Spenditure – The Art of Smart Spending

Memorial University of Newfoundland

Department of Engineering and Applied Sciences

ECE 5010 – Design Report

Prepared for:

Dr. Andrew Vardy, *P. Eng, B.Eng., M.Sc., PhD*

Professor, Principal Investigator – BOTS Group

Faculty of Electrical and Computer Engineering | Faculty of Computer Science

Memorial University of Newfoundland, St. John's, NL

Prepared by:

Waleed Mannan Khan Sherwani

Ahmad Hajahmad

Mohammed Al Taie

Contents

Introduction 3
Purpose 3
Scope 3
Definitions and Acronyms 4
Reference Material 4
JavaFX 4
PostgreSQL 4
Tesseract OCR 4
System Overview 5
Design Ideas and Class Diagrams 5
Class Diagram 6
Implementation – Use Cases 7
Use Case 1: Adding a Transaction 7
Use Case 2: Generating a Report 8
Data Categorization Design 9
Data Description 9
Receipt Scanning and Parsing 9
Data Extraction 9
Product Categorization 9
User Interface Design 10
Overview of User Interface 10
Sample UI 10
Usage of Design Patterns and Principles 11
Design Principles 11
Design Patterns 11
Conclusion 12

Introduction

Purpose

Spenditure is a financial management desktop app designed to simplify the organization of purchase records and help keep track of it. Users can easily upload their receipts onto the app and obtain weekly, monthly, and yearly expenditure analytics for their spending habits. It aims to empower users to make informed financial decisions by effectively managing their transactions and receipts and providing insightful analytics.

Scope

This section is intended to list more features of Spenditure while also diving into more details about each part.

- **Transaction Management**: Provide a seamless interface for users to enter and manage their financial transactions.
- **Analytics**: Offer comprehensive analytics that allows users to understand their spending patterns and make informed financial decisions.
- **Automated Reports**: Generate reports automatically, giving users periodic insights into their financial health.
- **Currency Conversion**: Incorporate a currency conversion feature directly into the dashboard for users dealing with multiple currencies.
- **Receipt Digitization**: Utilize Tesseract OCR technology to scan and parse data from physical receipts, reducing entry errors and saving time.
- **Multi-User Support**: Enable several users to create and manage their individual financial profiles within a shared environment.
- **Reminders**: Implement a reminder system for payments and subscriptions to help users avoid late fees and manage their financial commitments more effectively.

Definitions and Acronyms

- OCR - Optical Character Recognition
- SQL - Structured Query Language
- CRUD - Creating, Reading, Updating, and Deleting
- SRP - Single Responsibility Principle
- OOP - Object-Oriented Programming
- JavaFX - Java Graphics Framework
- FXML - JavaFX Markup Language
- CSS - Cascading Style Sheets
- XML - Extensible Markup Language
- UML - Unified Modeling Language

Reference Material

The following reference material was used:

JavaFX

- Overview (JavaFX 21) (openjfx.io)
- Introduction to FXML | JavaFX 21 (openjfx.io)
- JavaFX CSS Reference Guide (openjfx.io)
- jfx/doc-files/release-notes-21.md at jfx21 · openjdk/jfx · GitHub

PostgreSQL

- PostgreSQL 16.2 Documentation

Tesseract OCR

- github tesseract-ocr.
- Tesseract User Manual.
- TessJ4 package libraries.
- Tesseract OCR with Java

The Class and Sequence Diagrams for this document were created using *Mermaid.JS*

This document was organized and formatted in line with the teachings of *ECE 5010* and *IEEE STD 1016 – Software Design Document*.

System Overview

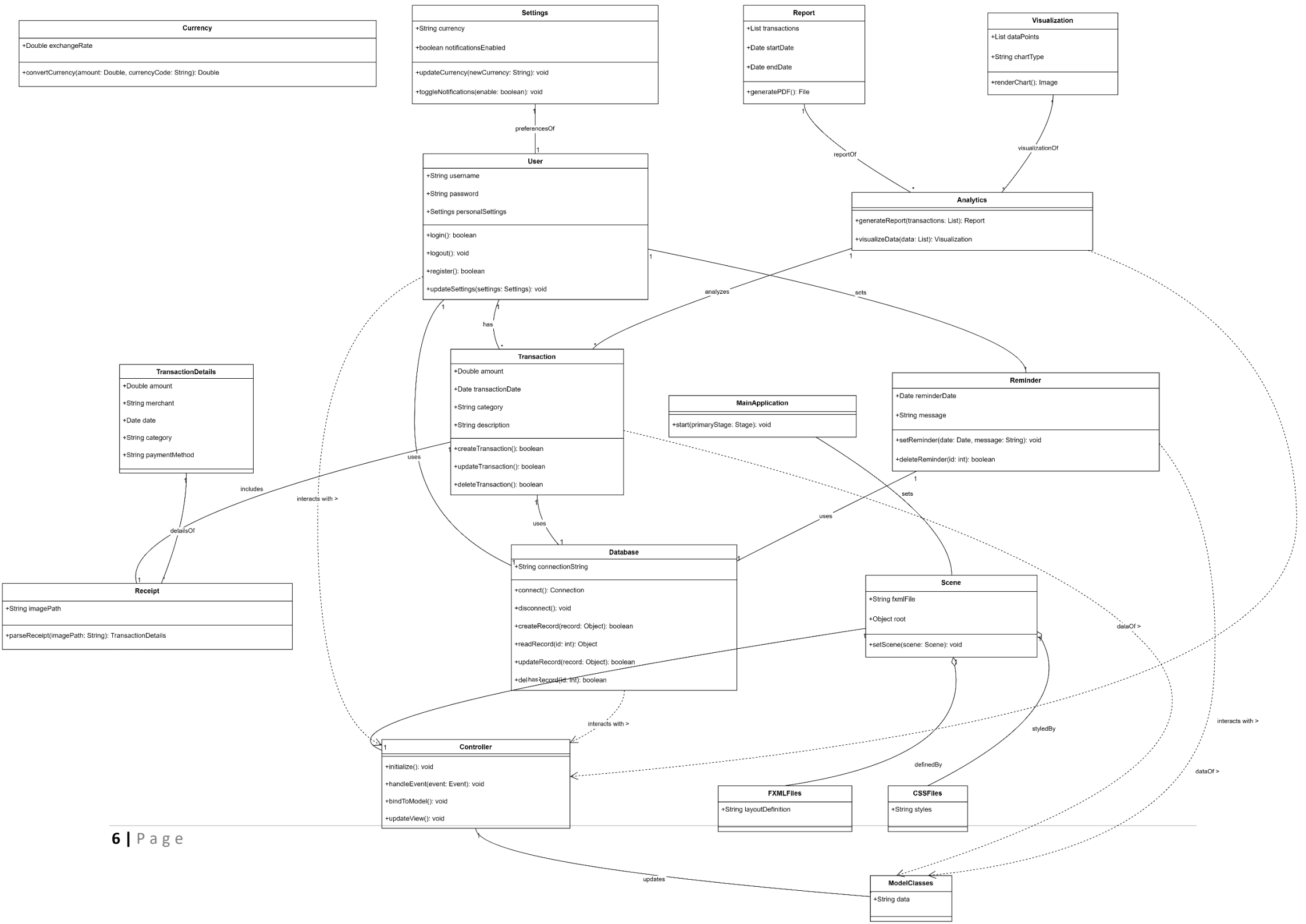
Design Ideas and Class Diagrams

The design of Spenditure is based on the principles of Object-Oriented Programming (OOP), which allows for modularity, reusability, and maintainability of code. The software is divided into several classes, each responsible for a specific functionality.

- **User Class**: This class represents the user of the application. It contains user-specific information such as username, password, and personal settings.
- **Transaction Class**: This class represents a financial transaction. It includes details such as the amount, date, category, and transaction description.
- **Receipt Class**: This class is responsible for handling the receipt scanning feature. It uses Tesseract OCR to parse the details from a physical receipt.
- **Database Class**: This class handles all interactions with the PostgreSQL database. It includes methods for CRUD (Creating, Reading, Updating and Deleting) transactions.
- **Analytics Class**: This class generates reports and analytics based on the user's transactions.
- **Currency Class**: This class handles the currency conversion feature, allowing users to view their transactions in different currencies.
- **Reminder Class**: This class manages the payment and subscription reminders.

The classes interact with each other to provide the overall functionality of Spenditure. For example, the User class interacts with the Transaction, Receipt, and Reminder classes to add transactions, scan receipts, and set reminders, respectively.

These interactions are shown better through the class diagram on the following page.


Currency
+Double exchangeRate
+convertCurrency(amount: Double, currencyCode: String): Double
Settings
+String currency
+boolean notificationsEnabled
+updateCurrency(newCurrency: String): void
+toggleNotifications(enable: boolean): void
Report
+List transactions
+Date startDate
+Date endDate
+generatePDF(): File
Visualization
+List dataPoints
+String chartType
+renderChart(): Image
preferencesOf
reportOf
visualizationOf
User
+String username
+String password
+Settings personalSettings
+login(): boolean
+logout(): void
+register(): boolean
+updateSettings(settings: Settings): void
Analytics
+generateReport(transactions: List): Report
+visualizeData(data: List): Visualization
analyzes
sets
has
Transaction
+Double amount
+Date transactionDate
+String category
+String description
+createTransaction(): boolean
+updateTransaction(): boolean
+deleteTransaction(): boolean
TransactionDetails
+Double amount
+String merchant
+Date date
+String category
+String paymentMethod
MainApplication
+start(primaryStage: Stage): void
Reminder
+Date reminderDate
+String message
+setReminder(date: Date, message: String): void
+deleteReminder(id: int): boolean
includes
interacts with >
uses
detailsOf
Database
+String connectionString
+connect(): Connection
+disconnect(): void
+createRecord(record: Object): boolean
+readRecord(id: int): Object
+updateRecord(record: Object): boolean
+deleteRecord(id: int): boolean
Receipt
+String imagePath
+parseReceipt(imagePath: String): TransactionDetails
Scene
+String fxmlFile
+Object root
+setScene(scene: Scene): void
dataOf >
interacts with >
styledBy
definedBy
Controller
+initialize(): void
+handleEvent(event: Event): void
+bindToModel(): void
+updateView(): void
FXMLFiles
+String layoutDefinition
CSSFiles
+String styles
updates
ModelClasses
+String data

Implementation – Use Cases

In this section, we dive into two practical applications of Spenditure's features through detailed use cases. With sequence diagrams for those specific functionalities, we demonstrate the process by which the software responds to user actions, offering a clearer understanding of its operation.

Use Case 1: Adding a Transaction

- The user initiates the process by choosing to add a transaction.
- The User class creates a new instance of the Transaction class.
- The Transaction class prompts the user to enter the transaction details.
- The user enters the details, which are then passed to the Transaction class.
- The Transaction class sends the transaction details to the Database class.
- The Database class updates the database with the new transaction.

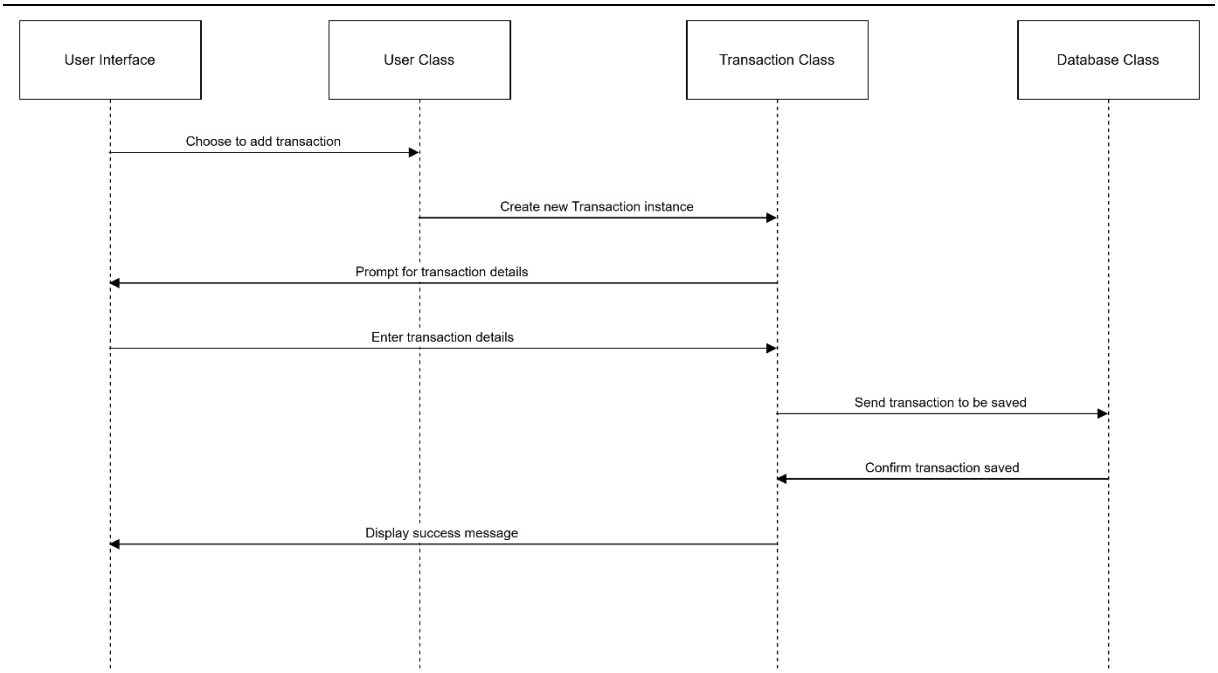


Figure - Sequence diagram illustrating the process of adding a transaction initiated by the user.

Use Case 2: Generating a Report

- The user initiates the process by requesting a report.
- The User class sends a request to the Analytics class.
- The Analytics class retrieves the user's transaction data from the Database class.
- The Database class returns the requested data to the Analytics class.
- The Analytics class processes the data and generates the report.
- The report is then presented to the user.

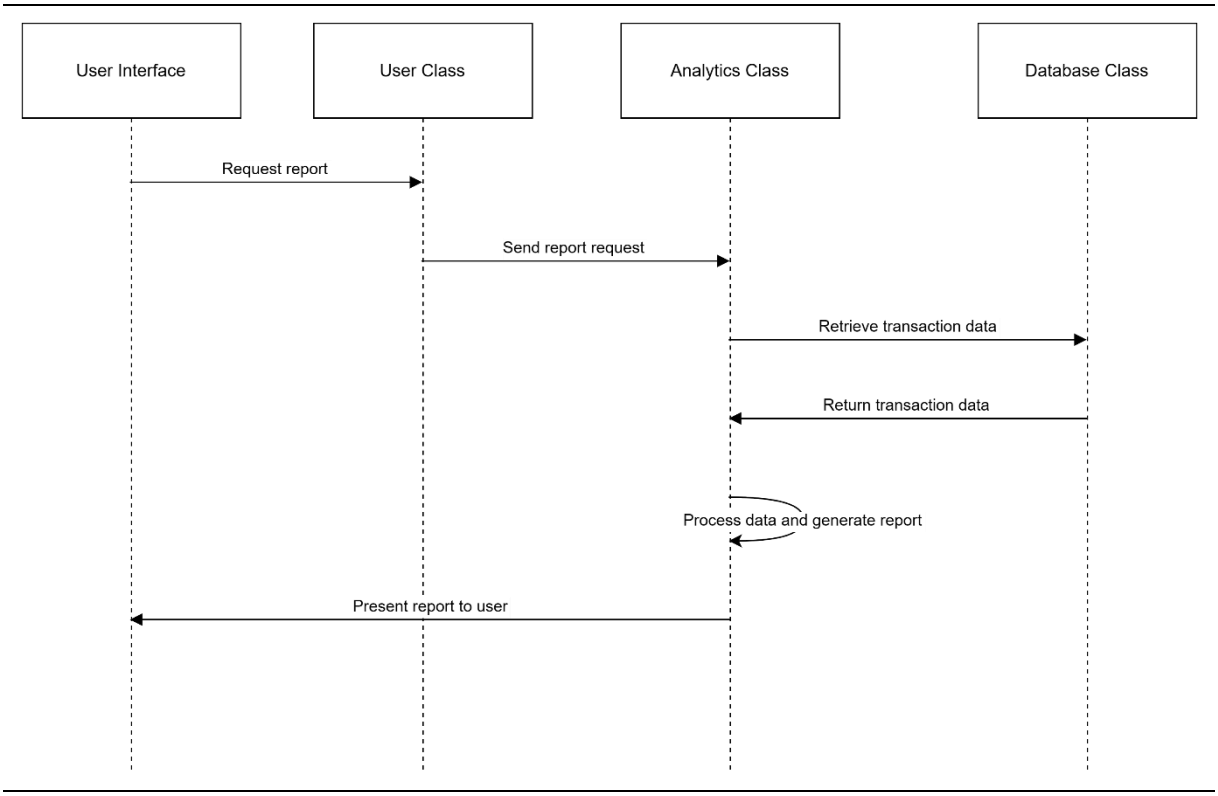


Figure - Sequence diagram illustrating the process of generating a report, initiated by user request

Data Categorization Design

Data Description

For database management, PostgreSQL was selected for its reliability and scalability, aligning with the essential requirements for handling complex data. This choice facilitates efficient storage, easy access, and effective handling of a wide range of data types.

Following the scanning and parsing of receipts with TesseractOCR, the extracted data is stored in PostgreSQL. This arrangement allows for the manipulation and analysis of data, enabling the generation of insightful reports and analytics. These features assist in tracking and understanding spending patterns, offering valuable insights to users.

Receipt Scanning and Parsing

The Tesseract OCR library will be utilized to scan the receipts and extract the text. This process involves pre-processing the image to improve the OCR results, such as converting the image to grayscale, applying a threshold, and resizing the image.

Data Extraction

Upon extracting the text from the receipt, regular expressions or a similar method will be employed to identify and extract the relevant information from the text. This includes the date, receipt number, and the list of purchased items along with their quantities and prices.

Product Categorization

To categorize the products, a mapping of product names to categories will be created. This could be a simple dictionary or a more complex machine learning model, depending on the variety and complexity of the products.

User Interface Design

Overview of User Interface

JavaFX is a software platform for creating and delivering desktop applications.

- **Main Application Class**: This class is the entry point for all JavaFX applications. It has a method where the is defined, and the is set. This class also handles application lifecycle events.
- **Scene Class**: This class represents the physical contents of a JavaFX application. It is a container for all content in a scene graph, which consists of nodes that represent all visual elements of the application's user interface. It handles the layout and styling of the UI components.
- **Controller Class**: This class is responsible for handling user input events. Each screen in the application has a corresponding controller class. The controller manipulates the data of the model and the view.
- **FXML Files**: These are XML files that define the user interface layout for each screen in the application. They provide the structure for the user interface and separate the application's logic from the layout.
- **CSS Files**: These files are used to style the JavaFX components. They separate the application's look and feel from the structure and behavior of the UI.
- **Model Classes**: These classes represent the data and are used by the JavaFX properties and bindings technology to respond to user events.

Sample UI

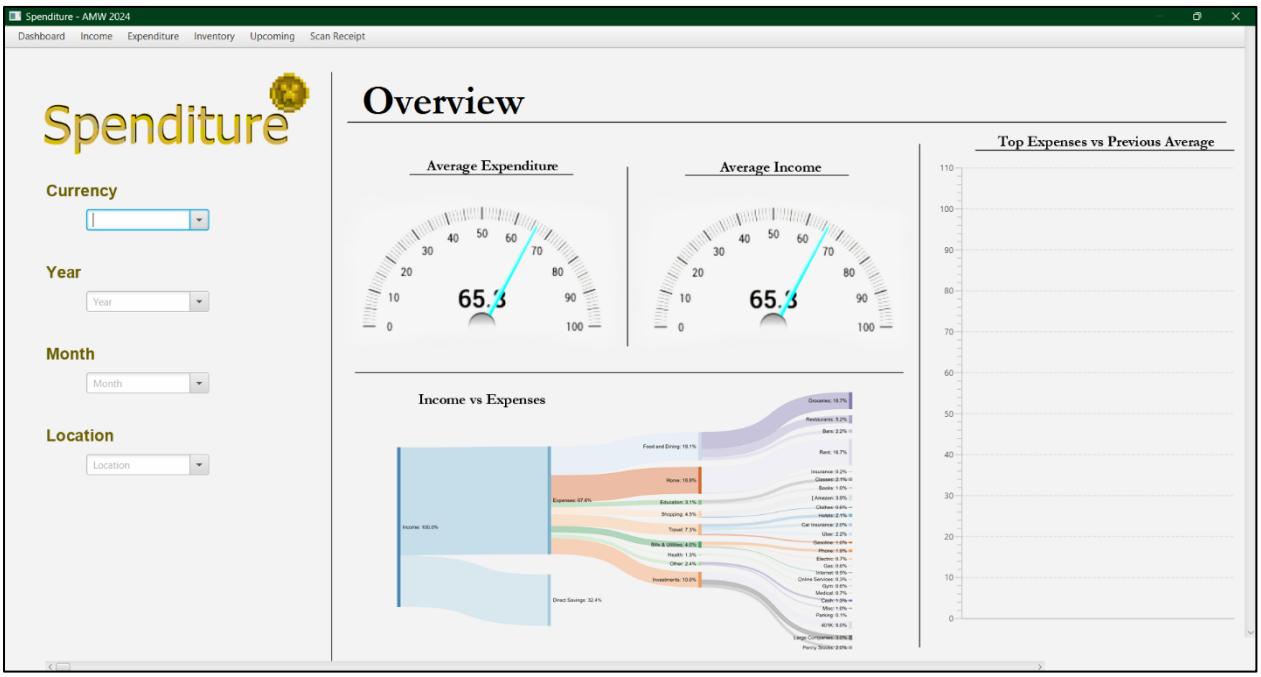


Figure – Sample Dashboard Main Page (Not Final)

Usage of Design Patterns and Principles

In developing Spenditure, high-level design choices were guided by established design principles and the use of design patterns to ensure a strong, maintainable, and scalable system. Below, an outline of how these principles and patterns are integrated is provided.

Design Principles

- **Single-Responsibility Principle (SRP)**: Each class in Spenditure has a single responsibility and a single reason to change. For instance, the *Reminder* class is solely responsible for handling reminder operations, whereas the *Currency* class deals with currency conversion. This adherence to SRP promotes modularity and makes the system easier to understand and maintain.

Design Patterns

- **Iterator**: To navigate through collections of transactions or user profiles, we implemented the Iterator pattern. This pattern provides a way to access the elements of an aggregate object sequentially without exposing its underlying representation, enhancing encapsulation and usability.
- **Strategy**: For features requiring algorithms that can be selected at runtime, such as different sorting or filtering of transaction records, the Strategy pattern is used. This allows the algorithm's behavior to be chosen depending on the user's preferences or specific conditions, promoting flexibility in data manipulation.
- **Factory**: The Factory pattern is employed to create objects without specifying the exact class of object that will be created. This is useful in Spenditure for creating different types of reports or analytics dynamically based on user input or specific criteria.
- **Singleton**: We use the Singleton pattern for classes where only a single instance should exist, such as the *Database* class. This ensures that only one connection to the database is active at any time, managing resources efficiently and preventing concurrent access issues.

Conclusion

In conclusion, this design report provides a comprehensive overview of the project, Spenditure – The Art of Smart Spending. The purpose and scope of the project are detailed, highlighting key features and benefits. Overall design ideas are discussed, including the use of Object-Oriented Programming principles and various design patterns to ensure a robust, maintainable, and scalable system.

Class and sequence diagrams illustrate the structure and interactions of the classes in the software, providing a clear understanding of how Spenditure operates. The data categorization design section explains how data, particularly in terms of receipt scanning and parsing, is handled.

There is an optimistic outlook on the potential of Spenditure. The commitment to good design principles and focus on user experience are expected to result in a quality software output. However, awareness of the risks and challenges that lie ahead is crucial, from future testing and implementation.

In closing, gratitude is expressed for the opportunity to work on this exciting project. The journey ahead is looked upon with anticipation and a commitment to making Spenditure a success.

Thank you for your time and consideration :)