Development
of
MoneyExpense – Spending Tracker Application
for
Student

Development of MoneyExpense – Spending Tracker Application for Student

A programming in Java project Report

Submitted by

Quad Coder

Sl No.	ID	Name
01	**21303134**	**Md. Alamin**
02	22203004	Tonmoy Sarker
03	22203032	Hasan Imam
04	23203053	Farzana Yasmin
05	20203040	Al Amin Akash
06	22203184	Mahabuba Akter Mima

Department of Computer Science and Engineering

College of Engineering and Technology

IUBAT- International University of Business Agriculture and Technology

Table Of Contents

Table Of Figures

Abstract

The Project has been developed by us as a course requirement of Programming in Java Lab (CSC 384). By which we can prove our practical Java Development skill throughout the project development. We know, managing personal finances can be challenging for students who often struggle to track their spending. This project, *MoneyExpense – Spending Tracker*, aims to simplify expense tracking by providing a user-friendly application built using JavaFX and MySQL. Key features include categorizing expenses, filtering by date, and viewing spending summaries. In this project we used the MVC (Model View Controller) architecture ensures maintainability, and our all-functional testing has validated this MoneyExpense Application performance. Future improvements include data visualization and reporting features.

Letter of Transmittal

Date: 09 January, 2025.
To: Rubayea Ferdows

Subject: Submission of Project Report on *MoneyExpense – Spending Tracker*

Dear ma'am,

We are pleased to submit our project report for the *MoneyExpense – Spending Tracker*. This report details our journey through development, including design, implementation, and testing phases. We hope it meets your expectations; though we have some limitation and scope to improve the project functionality in future

We hope, this report will give you the insight about our project.

Sincerely,

Quad Coder

Sl No.	ID	Name	Phone	Email	Signature
01	**21303134**	**Md. Alamin**	**01822679672**	**alamin5g@yahoo.com**	
02	22203004	Tonmoy Sarker	01749240754	tonmoysarker7676@gmail.com	
03	22203032	Hasan Imam	01783174890	hasanimam72108@gmail.com	
04	23203053	Farzana Yasmin	01319961616	farzanayasmin199@gmail.com	
05	20203040	Al Amin Akash	01790770652	alaminakash550@gmail.com	
06	22203184	Mahabuba Akter Mima	01763496878	mahabubamima@gmail.com	

Section: A

Course Title: Programming in Java Lab

Course Code: CSC-384

Semester: Fall-2024

Introduction

Overview

The *MoneyExpense – Spending Tracker* is a JavaFX-based desktop application designed to help users track and categorize expenses. It features user-friendly data entry, filtering options, and summaries to aid in financial management.

Motivation

As a student we have to utilize (expense/cost) the money properly for the whole month. Sometimes, we are out of money at the end of last week of the month. Because we don't know where we expense/invest our money. So, we thought we will develop a spending tracker application where we can observe in which particular categories we have invest/expense our money. Also, we can see the current uses of money for the current month. Seems it will help us to find the how much money I spent already and what were the categories.

Objectives

- By using the MoneyExpense Application, users can track expenses by category and date.
- User can get monthly summaries for better budget control.
- For ease of use, we offer a simple user interface for seamless interaction with any user.

Methodology

To develop the *MoneyExpense – Spending Tracker*, we followed a structured approach to ensure the application is easy to maintain and performs well. Here's how we developed the project step by step:

1. **Used MVC Architecture**
   We used the **Model-View-Controller (MVC)** design pattern. This method divides the application into three parts:
   - **Model**: Manages the data and business logic (like handling expenses and categories).
   - **View**: The part of the application that users interact with, like buttons and tables in the JavaFX interface.

- **Controller**: Connects the user actions (like clicking buttons) with the logic in the model. This makes the code clean and easy to manage because each part has a specific role.

2. **JavaFX for the User Interface**
   We used **JavaFX**, which is a library in **JDKFX**, to create the user interface. JavaFX provides tools for building modern, interactive desktop applications. We designed the app using **Apache NetBeans IDE**, which made it easier to visualize the interface while developing. The interface was designed to be responsive, meaning it adapts to different window sizes and provides a smooth experience for the user.

3. **MySQL for Data Storage**
   To save user data (like expenses and categories), we used **MySQL**, a popular database system.

   - MySQL was set up using the **XAMPP Server**, a tool that combines MySQL, PHP, and Apache into one package. This made it easy to run and test the database on our computers. This setup ensures that data is saved permanently, so users can access their expense history even after restarting the app.

4. **Incremental Development Process**
   Instead of building everything at once, we used an **incremental development process**. This means:

   - We built one feature at a time (like adding categories first, then expenses, and then filtering). After completing each feature, we tested it thoroughly to make sure it worked before moving on to the next feature. This process allowed us to fix any issues early and ensured that the app worked smoothly at every stage.

Process Model

Reason For Using the Process Model

The **Incremental Process Model** is a method where a project is developed in small steps or "increments." Each step focuses on adding a specific part of the application.

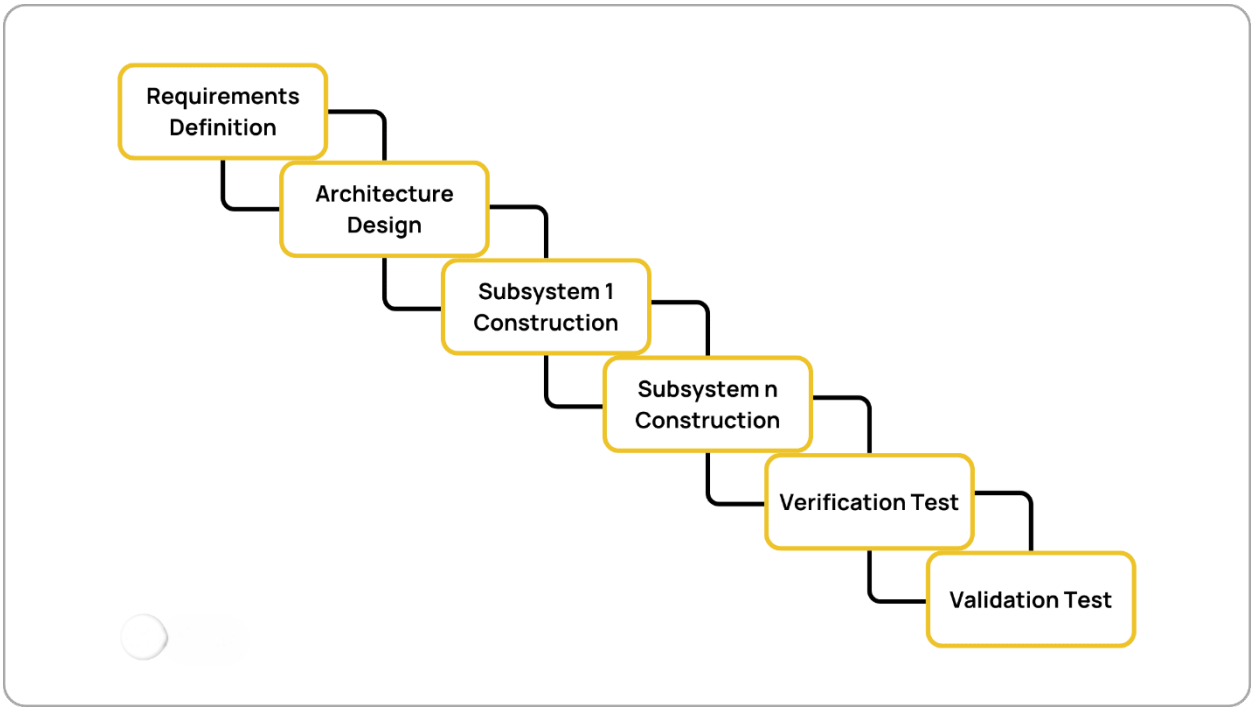

Figure 1- Incremental Process Model

Why We Chose the Incremental Process Model

We chose this model for the following reasons:

1. **Gradual Feature Implementation**
    - Instead of building the entire application at once, we worked on one feature at a time. For example, we started with the ability to add categories. Once that was working properly, we moved on to adding expenses, filtering options, and finally, viewing summaries. This step-by-step process allowed us to stay organized and focus on each feature separately, ensuring that it worked as expected before moving on to the next.

2. **User Testing and Feedback for Each Iteration**
    - After completing each step, we tested the new feature and got feedback from users. For instance, once the "Add Expense" feature was added, we asked a few users to try it out and tell us if it was easy to use. Their feedback helped us make improvements, such as ensuring better error handling (e.g., showing a warning if a category is left empty). This approach made the development process interactive and ensured that the application was user-friendly.
3. **A Working Product Early in the Development Cycle**
    - With the Incremental Process Model, even in the early stages of the project, we had a basic version of the app that worked. For example, after implementing the "Add Category" feature, we already had something that users could try. This approach is very helpful because:
        - It gives the team motivation by seeing visible progress early.
        - It ensures that if the project timeline becomes tight, we still have a working application with core functionalities ready for use.

Requirement Engineering

User and System Requirements

The requirements were gathered and refined using the **Incremental Process Model**, which allowed us to focus on specific needs step-by-step, test them with users, and improve them iteratively. Below are the detailed requirements:

User Requirements

The application is designed primarily for students, so simplicity and usability were prioritized. The requirements include:

1. **Log and Categorize Expenses Easily**
    - Users should be able to log their expenses without any confusion. The process should involve entering the date, category, and amount in an intuitive form.
    - The system must allow users to add custom categories (e.g., food, transportation, education) and manage them easily (e.g., delete unused categories).
    - This was implemented in an initial increment to ensure the core logging functionality worked well before moving to more advanced features.
2. **View Expenses Filtered by Category or Date**
    - Users should be able to filter their spending data based on categories or dates to get clear insights into their spending habits.
    - For instance, a user might want to see how much they spent on food in the last month. This requirement was addressed in a subsequent increment, after the logging functionality was stable.

System Requirements

The system was designed to ensure reliability and offline usability:

1. **Store Data Persistently in a Database**
    - All expense and category data must be saved permanently, even if the application is closed.

- This was implemented using MySQL in the first increment to establish a stable backend for the application.

2. **Work Offline**
    - The application should not rely on internet connectivity. All data processing and storage occur locally, making it convenient for users who do not have consistent internet access.
    - This requirement influenced the choice of technologies like JavaFX for the frontend and MySQL with XAMPP for local database storage.

Functional Requirements

The application's functionality was developed incrementally, starting with basic features and gradually adding more advanced capabilities based on user needs.

1. **Add, Delete, and Manage Categories**
    - **Increment 1:** Developed the feature to add and delete expense categories. This was a foundational feature that ensured users could customize their expense tracking.
    - **Testing:** Users tested the system to ensure categories could not be duplicated, and unused categories could be deleted.
    - **Improvement:** Based on feedback, input validation was added to prevent empty or duplicate categories.
2. **Log Expenses with Date, Category, and Amount**
    - **Increment 2:** Implemented the ability to log expenses, including the date, category, and amount.
    - **Testing:** Users entered various types of expenses to verify that the system stored them correctly.
    - **Improvement:** Error handling was added to ensure that invalid inputs (e.g., negative amounts) were rejected.
3. **Filter Expenses and Show Summaries**
    - **Increment 3:** Added filtering options to allow users to view expenses by category and date range.

- **Increment 4:** Developed summary features to display the total amount spent in a selected category or for a specific month.
- **Testing:** Verified that the filtering and summary calculations were accurate, especially for edge cases like overlapping date ranges or empty datasets.

Risk Management

Managing risks is an important part of any project because it helps avoid problems that can affect the success of the application. In our project, we followed a structured process to identify, understand, and reduce risks.

Stages of Risk

1. **Identification**
    - First, we thought about all the possible risks that could occur during the design and testing stages.
    - For example, what if the database stopped working? What if the user interface (UI) didn't work properly?
    - We wrote down all these risks to prepare for them.
2. **Analysis**
    - After identifying the risks, we analyzed how serious each risk was and how it could affect the project.
    - For example, if the database failed, users wouldn't be able to save their expenses, which would make the application useless. This is a big problem.
    - If the UI had small bugs, it might annoy users but wouldn't stop the app from working, so it's less serious.
3. **Mitigation**
    - Once we knew the risks and their impact, we planned ways to reduce or fix them.
    - For example, for database failure, we decided to use backup systems and proper error-handling techniques to reduce the chances of failure.

Categories of Risk

Here wee grouped risks into three types:

1. **Technical Risks**
    - These are related to the software or technology used in the project.
    - Examples:
        - **Data Integrity Issues:** Data might get lost or saved incorrectly.
        - **UI Responsiveness:** The app might not work smoothly or crash.
2. **Operational Risks**
    - These are related to how the project is managed.
    - Examples:
        - **Time Management:** The team might not finish the project on time.
        - **Resource Management:** Team members might not use tools and time efficiently.
3. **External Risks**
    - These come from outside the project and are beyond our control.
    - Examples:
        - **Technology Changes:** New updates or tools might create compatibility issues.
        - **Requirement Changes:** Users might request new features after the app is partly built.

RMMM Plan

RMMM stands for **Risk Mitigation, Monitoring, and Management**. Here's how we dealt with risks:

1. **Backing Up Data Regularly**
    - We made sure that all important files, like the database and project code, were saved in multiple locations. This way, even if something went wrong, we could recover quickly.
2. **Allocating Time Buffers**
    - We added extra time to our schedule for unexpected problems. For example, if testing took longer than expected, we had some extra time to finish it.
3. **Testing for Edge Cases**
    - We tested the app with unusual or extreme inputs (like negative amounts for expenses) to make sure it handled all possible situations.

Risk Analysis

Here's an example of a risk we identified and how we managed it:

- **Risk:** Database Connection Failure
    - **Impact:** If the app couldn't connect to the database, users wouldn't be able to save or retrieve their expenses. This would make the app useless.
    - **Mitigation:** We used a feature called **connection pooling**, which makes the database connection stable and efficient. We also added error-handling code to show a message to users if the database connection failed and allow them to retry.

RMMM Table

This table summarizes some of the key risks and how we managed them:

Risk	Impact	Mitigation Strategy
Data Loss	High	Regular backups, version control.
UI Bugs	Medium	Rigorous testing and debugging.
Database Issues	High	Connection pooling, error handling.

Analysis Modeling

Use Case Symbols

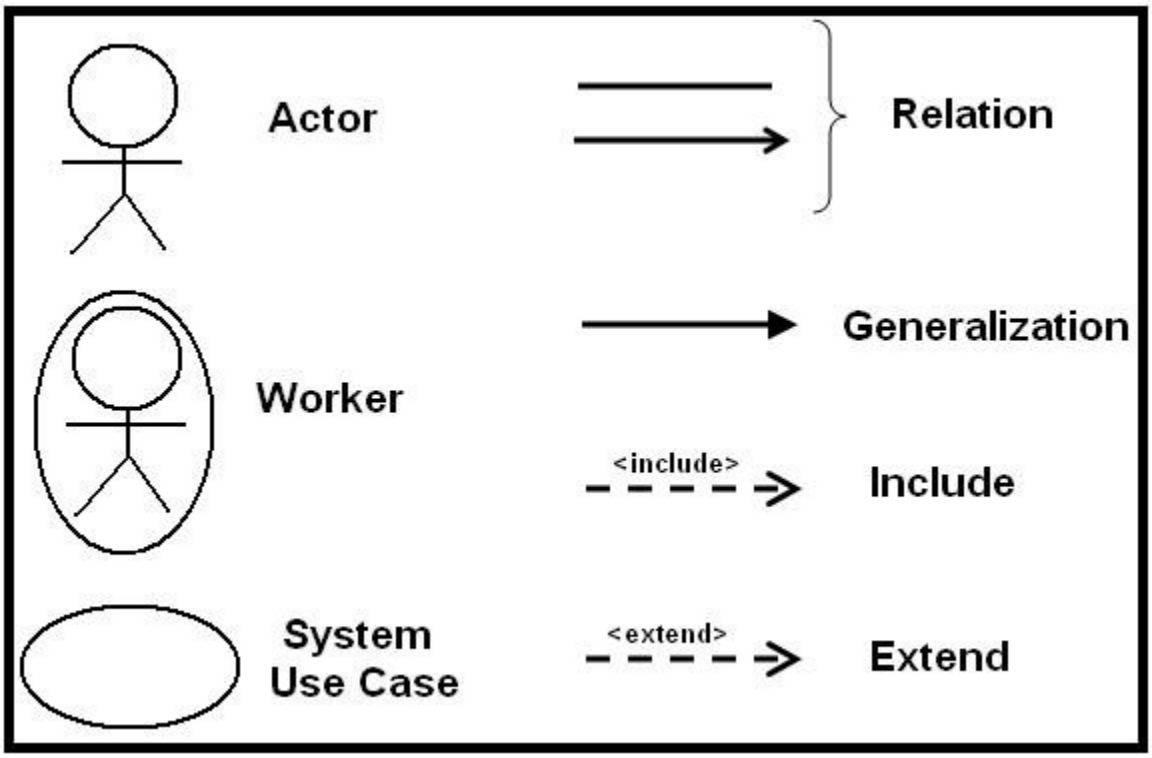

Figure 2 Use case Symbols

Actor: Represented by stick figures.

Use Case: Ellipses showing system functionality.

Relationships: Arrows connecting actors and use cases.

Use Case Diagram

The Use Case Diagram visually explains how the user interacts with the system. Here's a description of the relationships in our project.

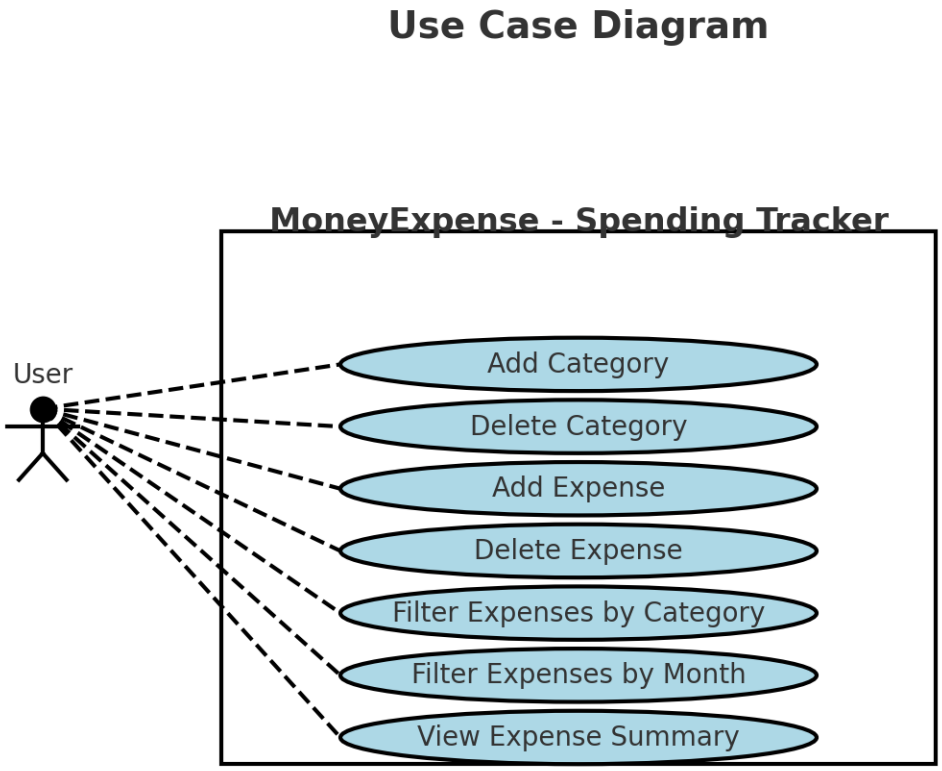

Figure 3 Use case Diagram

1. **Actors:**
    - **User:** The main actor who interacts with the system to perform tasks like adding categories, logging expenses, and viewing summaries.
2. **Use Cases (Functionalities):**
    - **Add Category:** Allows the user to add a new expense category.
    - **Delete Category:** Allows the user to remove an existing category.
    - **Log Expense:** Allows the user to record an expense.
    - **Delete Expense:** Allows the user to remove a specific expense entry.

- **Filter Expenses by Category:** Shows expenses filtered by the selected category.
- **Filter Expenses by Month:** Shows expenses filtered by a specific month.
- **View Summary:** Provides a summary of total expenses for the current month or all months.

3. **Relationships:**
   - Arrows connect the **User** to the functionalities (use cases) they interact with.
   - For example:
     - The **User** interacts with **Add Expense** to record a new expense.
     - The **User** interacts with **Filter Expenses by Category** to view expenses of a specific type.

Design

DFD (Data Flow Diagram)

1. **Context Level**
   The context-level DFD shows the interaction between the user and the system. It provides a high-level overview of how the user inputs data (e.g., categories, expenses) and receives outputs (e.g., filtered expenses, summaries).

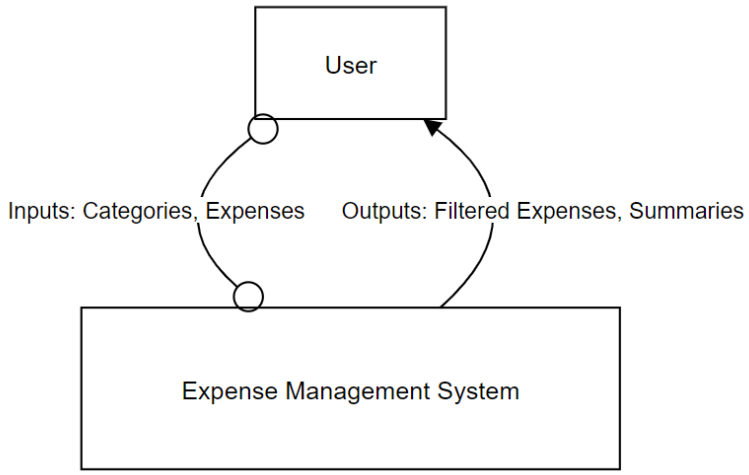

Figure 4 Context level Diagram

2. **Level 1**
   This level breaks down the system into its core functionalities, such as:
   - Managing categories (add/delete).
   - Logging expenses (date, category, amount).
   - Filtering expenses (by category or date).
   - Generating summaries (monthly or all-time totals).

3. **Level 2**
   This level adds detailed flows for specific functionalities. For example:
   - How the system validates inputs (e.g., ensuring categories are unique).

- The steps for retrieving filtered data from the database.
- The process of calculating and displaying summaries.

Database Design

- **Tables**:
  1. **category_info**: Stores unique expense categories.
  2. **spendings**: Stores individual expense entries, including the date, category, and amount.
- **Primary Keys**:
  - category in category_info ensures categories are unique.
  - sid in spendings is an auto-incrementing ID for each expense.

Entity Relationship Diagram:

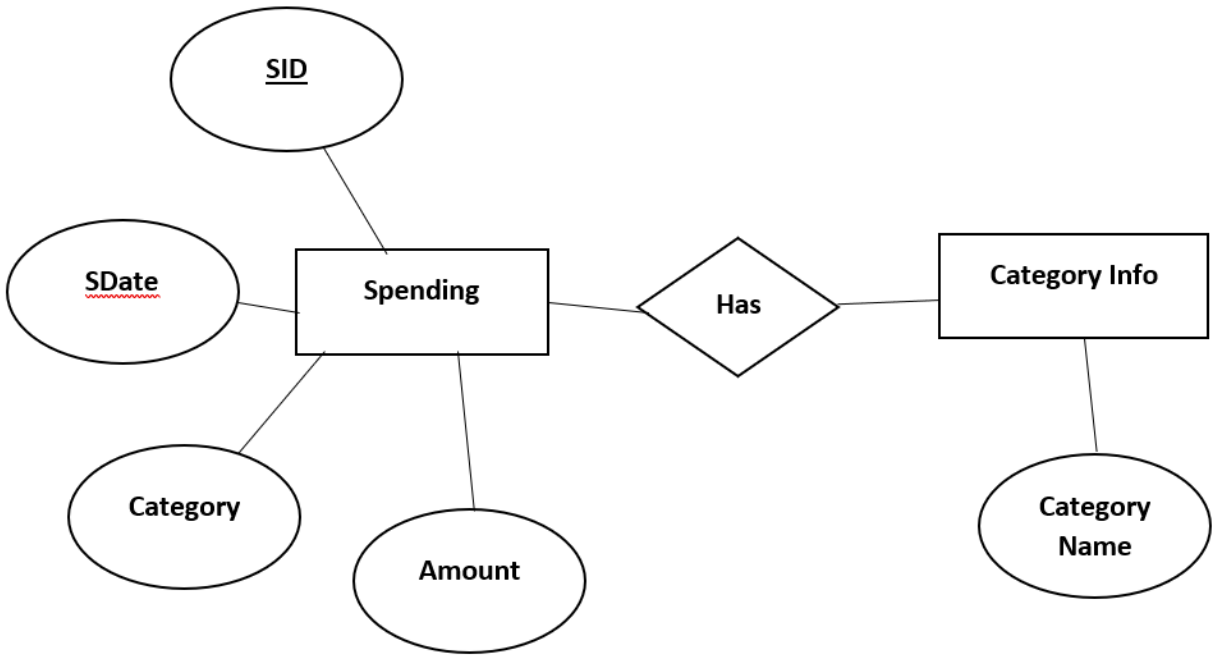

Figure 5 Entity Relationship Diagram

The ERD shows the relationships between the system's key entities:

1. **Categories**: Defines unique expense types.
2. **Spendings**: Represents individual expense records, linked to a category.

UML Diagram:

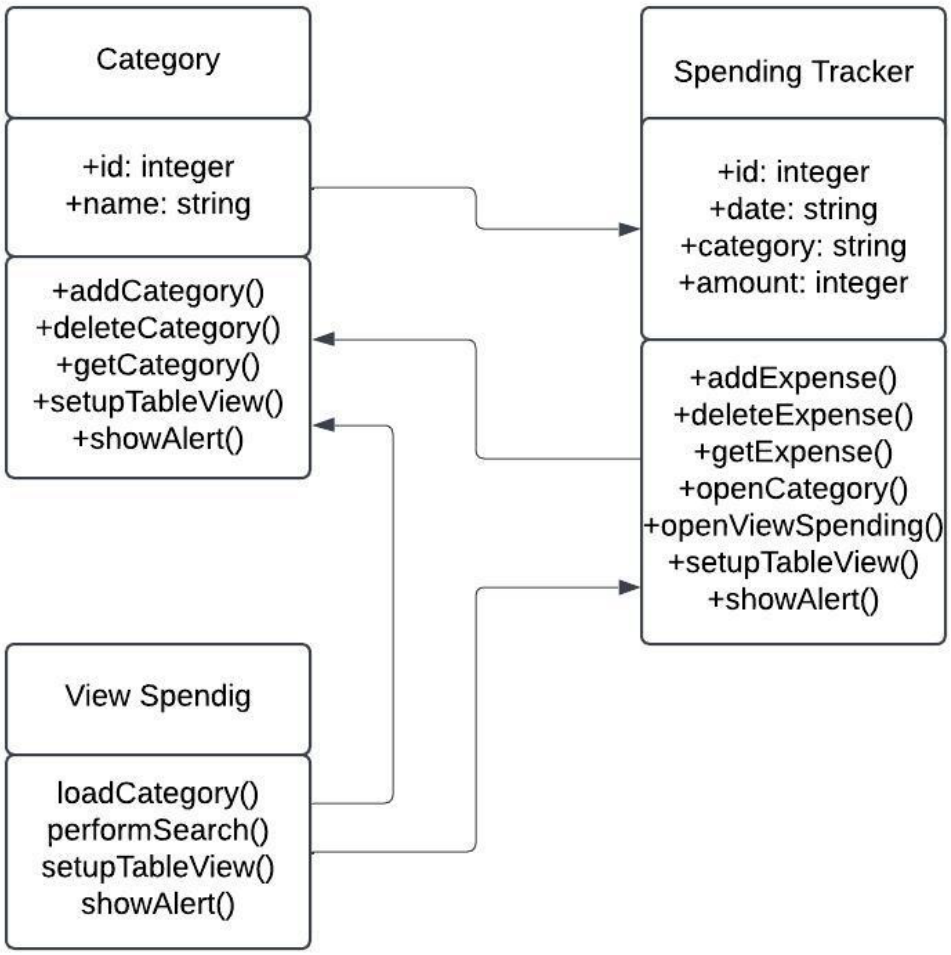

Figure 6 - UML Diagram

Quality Assurance

Unit Testing

- **Definition**: Unit testing focuses on testing individual components or methods of the application.
- **Purpose**: To ensure that each method performs as expected in isolation.
- **Example**:
    - **Add Category**:
        - Tested the addCategory() method to confirm that it correctly validates input and adds a category to the database.
        - Edge cases, such as empty input or duplicate categories, were tested to ensure appropriate error handling.
    - **Delete Expense**:
        - Verified that the deleteExpense() method successfully removes the selected expense from both the table view and the database.
        - Checked for scenarios where no expense is selected to ensure proper warnings are displayed.

Integration Testing

- **Definition**: Integration testing examines how different parts of the application work together.
- **Purpose**: To confirm that the components interact correctly, particularly with the database.
- **Example**:
    - **Database Operations**:
        - Verified that the loadCategories() method retrieves data correctly from the category_info table and updates the ComboBox.
        - Ensured the getEntries() method accurately fetched data from the spendings table within the specified date range and displayed it in the table view.

- **Error Handling**:
    - Tested database failure scenarios, such as unavailable connections, to ensure that meaningful error messages are shown to the user.

System Testing

- **Definition**: System testing ensures that all components of the application work together seamlessly.
- **Purpose**: To validate the overall functionality and performance of the application.
- **Example**:
    - **User Workflow**:
        - Simulated a complete user workflow:
            - Adding a category.
            - Logging multiple expenses.
            - Filtering expenses by category and date.
            - Deleting an expense.
        - Confirmed that data consistency was maintained across all operations.
    - **Cross-Component Testing**:
        - Verified that updates to categories in the Category module were immediately reflected in the SpendingTracker and ViewSpending modules.
    - **UI Responsiveness**:
        - Ensured the application's interface handled inputs efficiently and provided feedback in real time.

Table 1: Testing Result

Feature	**Test Case**	**Result**
Add/Delete Categories	Add/Delete multiple categories.	Passed.
Add Expense	Log expense with valid inputs.	Passed.
Filter by Category	Select categories from the list.	Passed.
Summarize Expenses	Show accurate totals.	Passed.

Limitations

Although the application performs its core functionalities effectively, there are certain limitations that could be addressed in future development:

1. **Graphical Representation of Data**
    - The current version lacks visual tools like graphs or charts to provide a clear picture of spending trends.
    - Adding features such as pie charts or bar graphs for expense visualization would make the app more intuitive and user-friendly.
2. **Exporting Features**
    - The app does not allow users to export their expense data to formats like CSV or PDF.
    - Exporting features would enable users to save and share their financial data for offline use or reporting.
3. **Multi-User Support**
    - The application is designed for single-user use and does not include user authentication or role-based access.
    - Adding multi-user support would allow multiple users to maintain separate accounts and securely track their personal expenses.

Scope

Current Scope

The application focuses on:

- **Expense Tracking for Individual Users**:
    - Users can add, delete, and manage expense categories.
    - Users can log expenses by date, category, and amount.
    - The app provides basic filtering and summaries of expenses based on categories and time periods.
- **Offline Functionality**:
    - The app works without internet connectivity, as all data is stored locally using a MySQL database.

Future Scope

The application can be expanded to include:

1. **Data Visualization**:
    - Introduce graphical representations such as pie charts and bar graphs for better insights into spending patterns.
    - Provide interactive dashboards for visualizing monthly and category-wise expenses.
2. **Reporting Features**:
    - Add options to export expense data in formats like CSV and PDF.
    - Allow users to generate detailed monthly or yearly expense reports.
3. **Multi-User Support**:
    - Implement user authentication (e.g., login functionality) to allow multiple users to maintain separate accounts.
    - Add role-based access for shared accounts, where users can track joint expenses (e.g., household expenses).

Conclusion

In the end, the **MoneyExpense – Spending Tracker** application successfully meets its goal of helping users manage their expenses. It allows students and anyone who wants to track their spending to categorize expenses, filter by month, and get a clear view of their financial situation.

We've learned a lot throughout this project—both in terms of JavaFX development and how to structure an app with MVC. If we had more time, we'd add features like graphs for better data visualization and maybe even an export option for expense reports.

References

- Oracle JavaFX Documentation: https://docs.oracle.com/javase/8/javafx/api/
- MySQL Documentation: https://www.w3schools.com/mysql/default.asp

Appendices

Task Contribution Table

Team Member	Task/Technology/Feature
Md. Alamin	JavaFX development, UI design & implementation.
Tonmoy Sarker	MySQL integration, UI Design
Hasan Imam	database schema design, Expense filtering.
Farzana Yasmin	Category management, error handling.
Al Amin Akash	Documentation, Requirement Analysis
Mahabuba Akter Mima	Use Case Diagram, testing, and debugging.