Case Study

Finance Management System

Introduction

As part of the case study development in my Hexaware training program, I was assigned the task of developing a **Finance Management System** using **Java**. This system is designed to help individuals or businesses track and manage their expenses effectively. The project was implemented with a strong focus on key programming concepts such as SQL, control flow statements, loops, arrays, collections, exception handling, database interaction, and unit testing.

The **Finance Management System** allows users to manage their personal or business finances by providing a platform to track their expenses, categorize them, and generate detailed reports. The project involves designing and implementing the system with essential features such as user authentication, expense management, and database connectivity.

Functionalities Implemented

The Finance Management System is designed to provide different functionalities for Users and Admins, ensuring a seamless experience for both roles.

User Functionalities

The system allows users to perform a variety of functions to manage their finances.

- **Create Expense**: Users can create and add new expense records to the system.
- **View All Expenses**: Users can view a list of all their recorded expenses.
- **Filter Expenses**: Users can filter expenses by category, date range, or amount.
- **Update Expense**: Users can modify existing expense records.
- **Delete Expense**: Users can delete unwanted or incorrect expenses.
- **Generate Report**: Users can generate reports for their expenses over a specific time period (e.g., daily, weekly, monthly).
- **Suggestion**: Users can submit suggestions for system improvements.
- **Delete Account**: Users can delete their account and associated data from the system.
- **Logout**: Users can log out of the system securely.

Admin Functionalities

Admins have advanced controls to manage users, products, and orders.

- **View Suggestions**: Admin can view the suggestions submitted by users for system improvements.
- **Add Expense Category**: Admin can add new categories for expenses (e.g., groceries, rent, etc.).
- **View All Categories**: Admin can view all existing expense categories.
- **View All Users**: Admin can view a list of all registered users in the system.
- **View Specific User Details**: Admin can view detailed information of a specific user.

- **Delete Suggestion**: Admin can remove suggestions that are no longer needed or are irrelevant.
- **Delete Category**: Admin can delete an existing expense category.
- **Log Out**: Admin can log out of the system securely.

SQL Database Design

The backend of the Finance Management System is powered by a MySQL relational database named Financedb.

1. Table: admin

This table stores information related to the admin users who manage the system.

Column Name	Data Type	Description
admin_id	INT	Primary Key, Auto Increment
admin_name	VARCHAR(100)	Name of the admin
password	VARCHAR(100)	Admin password (hashed for security)

2. Table: expensecategories

This table contains the categories for classifying different expenses.

Column Name	Data Type	Description
category_id	INT	Primary Key, Auto Increment
category_name	VARCHAR(50)	Name of the expense category

3. Table: expenses

This table stores the details of each expense entered by users.

Column Name	Data Type	Description
expense_id	INT	Primary Key, Auto Increment
user_id	INT	Foreign Key referencing users.user_id
amount	DECIMAL(10,2)	The amount spent on the expense
category_id	INT	Foreign Key referencing expensecategories.category_id
date	DATE	Date when the expense was recorded
description	TEXT	Description about the expense

4. Table: suggestions

This table stores user-submitted suggestions for system improvements.

Column Name	Data Type	Description
suggestion_id	INT	Primary Key, Auto Increment
user_id	INT	Foreign Key referencing users.user_id
suggestion_text	TEXT	The content of the user's suggestion

5. Table: users

This table stores information about registered users of the system.

Column Name	Data Type	Description
user_id	INT	Primary Key, Auto Increment
username	VARCHAR(50)	Unique username for the user
password	VARCHAR(100)	User password (hashed for security)
email	VARCHAR(100)	User's email address

Project Structure

The project is organized into several key resource files and packages, which are crucial for the development and functionality of the Finance Management System.

- **db.properties**: This file contains database configuration properties (such as connection URL, username, and password) for establishing a connection to the database.
- **src** : The main source folder where all the Java source code is located.

entity/model

The entity classes only contain fields and getter/setter methods. They represent real-world data entities (e.g., users, expenses) but don't contain any business logic.

- **Purpose**: This package contains the entity classes, which represent the data objects in the system.
- **Details**:
    - **Admin**: Represents the admin user.

- **Expense**: Represents an individual expense record.
- **ExpenseCategory**: Represents categories for classifying expenses (e.g., food, transport).
- **Suggestion**: Represents suggestions submitted by users for system improvement.
- **User**: Represents the end user of the system.

dao (Data Access Object)

The IAdminRepository and IFinanceRepository interfaces provide a contract for the database operations.The AdminRepositoryImpl and FinanceRepositoryImpl classes implement these interfaces and provide the actual database interaction logic (e.g., querying, inserting, updating records).

- **Purpose**: This package handles the interactions between the application and the database.
- **Details**:
  - **IAdminRepository**: Interface that defines the contract for admin-related database operations.
  - **IFinanceRepository**: Interface for finance-related database operations (e.g., expenses).
  - **AdminRepositoryImpl**: Concrete implementation of IAdminRepository that interacts with the database for admin functionalities.
  - **FinanceRepositoryImpl**: Concrete implementation of IFinanceRepository that handles finance-related database operations.

Exception

Custom exceptions like ExpenseNotFoundException and UserNotFoundException are used to handle specific cases when an entity (expense or user) is not found in the system.

- **Purpose**: This package contains user-defined exceptions to handle errors and specific issues within the system.
- **Details**:
  - **ExpenseNotFoundException**: Exception thrown when an expense is not found.
  - **UserNotFoundException**: Exception thrown when a user is not found.

util (Utility)

DBConnUtil helps with database connectivity by creating a connection object from the provided connection string.

DBPropertyUtil helps load database properties (like username, password, and URL) from the db.properties file, ensuring flexibility in connecting to different environments (e.g., development, testing, production).

- **Purpose**: This package contains utility classes that aid in database connectivity and property file management.
- **Details**:
    - **DBConnUtil**: A utility class with a static method that takes a connection string and returns a database connection object.
    - **DBPropertyUtil**: A utility class that retrieves the connection string from the db.properties file.

Main

The MainModule class is responsible for displaying the system's menu and coordinating the various functionalities (e.g., creating expenses, viewing expenses, generating reports). It serves as the main entry point for the application, providing a user-friendly interface.

- **Purpose**: The main class and entry point of the application.
- **Details**:
    - **DBTest**: A test class for checking database connectivity and ensuring the system can interact with the database.
    - **FinanceApp**: The main application class that manages the flow of the finance management system.
    - **MainModule**: A class that demonstrates the system's functionalities in a menu-driven manner. It serves as the user interface for interacting with the system.

test.dao

- **Purpose**: This package contains unit tests for testing the DAO layer.
- **Details**:
    - **ExceptionHandlingTest**: A test class to verify exception handling in the system.
    - **FinanceRepositoryTest**: A test class for testing database interactions in the FinanceRepositoryImpl class.

Key Components

1. entity Package

This package includes all the core domain classes (entities) that directly represent the real-world objects in the system such as users,admin , expensecategor.

1. Admin Class

This class represents the **Admin** entity in the system. The admin is responsible for managing users, viewing suggestions, adding categories, etc.

Attributes:

- adminId: The unique identifier for the admin.
- adminName: The name of the admin.
- password: The password for the admin account.

Methods:

- **Constructor**: Initializes the adminId, adminName, and password.
- **Getters and Setters**: For accessing and modifying the adminId, adminName, and password.
- **toString()**: Returns a string representation of the admin details, including their adminId, adminName, and password.

2. Expense Class

This class represents the **Expense** entity, which holds information about an individual expense made by a user.

Attributes:

- expenseId: The unique identifier for the expense.
- userId: The ID of the user who created the expense.
- categoryId: The ID of the category to which the expense belongs.
- amount: The monetary value of the expense.
- date: The date the expense was incurred.
- description: A brief description of the expense.
- categoryName: The name of the category (can be set externally when fetching from the database).

Methods:

- **Constructors**:
  - One constructor initializes the expense with userId, categoryId, amount, date, and description.
  - Another constructor initializes all attributes, including expenseId, userId, categoryId, amount, date, and description.
- **Getters and Setters**: For accessing and modifying each of the attributes.

- **toString()**: Returns a string representation of the expense, including all its attributes.

3. ExpenseCategory Class

This class represents the **ExpenseCategory** entity, which categorizes different expenses (e.g., food, transportation).

Attributes:

- categoryId: The unique identifier for the category.
- categoryName: The name of the category (e.g., "Food", "Transportation").

Methods:

- **Constructor**: Initializes the categoryId and categoryName.
- **Getters and Setters**: For accessing and modifying categoryId and categoryName.
- **toString()**: Returns a string representation of the category, including its categoryId and categoryName.

4. Suggestion Class

This class represents the **Suggestion** entity, which holds user suggestions for system improvement.

Attributes:

- suggestionId: The unique identifier for the suggestion.
- userId: The ID of the user who made the suggestion.
- suggestionText: The text of the suggestion submitted by the user.

Methods:

- **Constructor**: Initializes the suggestionId, userId, and suggestionText.
- **Getters and Setters**: For accessing and modifying the suggestionId, userId, and suggestionText.
- **toString()**: Returns a string representation of the suggestion, including the suggestionId, userId, and the suggestion text.

5. User Class

This class represents the **User** entity, which holds the information about an individual user who uses the Finance Management System.

Attributes:

- userId: The unique identifier for the user.

- username: The username chosen by the user.
- email: The email address of the user.
- password: The password for the user's account.

Methods:

- **Constructor**: Initializes the userId, username, email, and password.
- **Getters and Setters**: For accessing and modifying the userId, username, email, and password.
- **toString()**: Returns a string representation of the user, including their userId, username, and email.

2. dao Package

The **DAO (Data Access Object)** package contains interfaces and their implementations for interacting with the database. The main goal of the DAO pattern is to provide an abstract interface to the data storage and manage operations like adding, retrieving, updating, and deleting records in the database.IAdmin Interface: Defines all operations an Admin can perform.

- **IAdminRepository** and **AdminRepositoryImpl**:
  - Interface and implementation for admin-related functionality.
  - Manages user information, expense categories, suggestions, and admin login.
- **IFinanceRepository** and **FinanceRepositoryImpl**:
  - Interface and implementation for finance-related functionality.
  - Handles user expenses, user login, suggestion management, and financial reports.

Both **AdminRepositoryImpl** and **FinanceRepositoryImpl** interact with the database, executing the necessary SQL queries to perform CRUD operations and return the results.

1. IAdminRepository Interface

This interface defines methods related to administrative operations in the system. Admins are responsible for managing users, suggestions, expense categories, and performing certain control functions in the application.

Methods:

- **adminLogin(String adminName, String password)**: Validates the admin credentials.
- **viewSuggestions()**: Allows the admin to view all user suggestions.
- **getUserById(int userId)**: Retrieves the user details based on their userId. Throws a UserNotFoundException if the user is not found.

- **deleteSuggestion(int suggestionId)**: Allows the admin to delete a specific suggestion by its ID.
- **addExpenseCategory(String categoryName)**: Allows the admin to add a new expense category.
- **isCategoryUsed(int categoryId)**: Checks if a given expense category is used by any expenses.
- **deleteCategory(int categoryId)**: Deletes a specific expense category by its ID.
- **viewAllCategories()**: Views all the existing expense categories.
- **viewAllUsers()**: Views all the users in the system.
- **getCategoryNameById(int categoryId)**: Retrieves the category name based on the category ID.

This interface is implemented by **AdminRepositoryImpl** to provide concrete database operations for admin-related functionality.

2. IFinanceRepository Interface

This interface defines methods related to finance operations, including user management, expense management, and reporting. It enables users to add and manage their expenses, view reports, and more.

Methods:

- **createUser(User user)**: Adds a new user to the system.
- **addExpense(Expense expense)**: Adds a new expense for a user.
- **deleteUserAndExpenses(int userId)**: Deletes a user and all their associated expenses.
- **deleteExpense(int expenseId, int userId)**: Deletes a specific expense for a user.
- **getAllExpensesByUserId(int userId)**: Retrieves all expenses for a given user by their userId.
- **updateExpense(Expense expense)**: Updates an existing expense for a user.
- **loginUser(String email, String password)**: Allows a user to log in using their email and password.
- **getExpensesByDateRange(int userId, Date fromDate, Date toDate)**: Retrieves all expenses for a user within a specific date range.
- **getExpensesByCategory(int userId, int categoryId)**: Retrieves all expenses for a user that belong to a specific category.
- **getExpenseById(int expenseId, int userId)**: Retrieves a specific expense by expenseId for a user. Throws an ExpenseNotFoundException if not found.
- **getCategoryWiseReport(int userId, Date start, Date end)**: Provides a category-wise report for a user in a given date range.

- **getMonthlySummary(int userId, Date start, Date end)**: Provides a monthly summary of a user's expenses within a given date range.
- **addSuggestion(Suggestion suggestion)**: Allows a user to add a suggestion to the system.

This interface is implemented by **FinanceRepositoryImpl** to provide concrete database operations related to finance management.

3. AdminRepositoryImpl Class

This class implements the **IAdminRepository** interface and provides concrete implementations for all the methods defined in the interface. It is responsible for interacting with the database to perform admin-specific operations, such as:

- Admin login and authentication.
- Viewing and deleting suggestions made by users.
- Managing expense categories (add, delete, check if in use).
- Viewing all users and categories.

This class will connect to the database, execute SQL queries, and return the results to the calling service or controller.

4. FinanceRepositoryImpl Class

This class implements the **IFinanceRepository** interface and provides concrete implementations for all the methods defined in the interface. It handles all finance-related operations for users, such as:

- Adding new users and expenses.
- Deleting users and their expenses.
- Updating existing expenses.
- Retrieving expenses by various criteria (e.g., user, date range, category).
- Generating reports (category-wise, monthly summary).
- Allowing users to add suggestions.

Just like **AdminRepositoryImpl**, this class interacts with the database to perform the finance-related CRUD operations and return the results.

3. exception Package

The exception package contains custom exception classes that are used to handle specific error scenarios within the application. By defining custom exceptions, the application can provide more detailed and specific error handling, making it easier to debug and manage error conditions.

1. ExpenseNotFoundException Class

This is a custom exception that is thrown when an expense cannot be found in the database. For example, when a user tries to retrieve an expense by its expenseId but it does not exist.

- **Constructor**: The constructor takes a String message parameter, which is used to pass a detailed error message about the exception. This message is passed to the parent Exception class through super(message).

```
package exception;

public class ExpenseNotFoundException extends Exception {
    public ExpenseNotFoundException(String message) {
        super(message);
    }
}
```

- **Usage**: This exception is likely thrown in scenarios like:
    - Attempting to get an expense by its ID that does not exist in the database.
    - Trying to update or delete an expense that is not present.

2. UserNotFoundException Class

This is a custom exception that is thrown when a user cannot be found in the system. For example, when a user attempts to log in with incorrect credentials or when a specific user is requested by their userId but does not exist.

- **Constructor**: Similar to ExpenseNotFoundException, the constructor of this exception takes a String message parameter, which is passed to the parent Exception class using super(message).

```
package exception;

public class UserNotFoundException extends Exception {
    public UserNotFoundException(String message) {
        super(message);
    }
}
```

- **Usage**: This exception would typically be thrown in situations such as: When a user tries to log in but no user with the provided email or username exists. When trying to retrieve or update a user's information using a non-existent userId.

4. util Package

This package contains utility classes used for managing database configuration and establishing database connections in a modular, reusable way.

DBPropertyUtil Class

- This utility class is responsible for reading the database configuration from an external db.properties file.
- It loads properties like db.url, db.user, db.password, and db.driver which are used to set up the connection.
- By using a properties file, the database settings are kept separate from the source code, making the application more configurable and secure.

```
public class DBPropertyUtil {
    public static Properties getDBProperties() {
        Properties properties = new Properties();
        try (FileInputStream input = new FileInputStream( name: "resources/db.properties")) {
            properties.load(input);
        } catch (IOException e) {
            System.err.println("Error loading database properties: " + e.getMessage());
        }
        return properties;
    }
}
```

DBConnUtil Class

- This class is responsible for establishing a connection with the database.
- It uses the properties loaded by DBPropertyUtil to retrieve connection parameters.
- Uses DriverManager.getConnection() to open a connection using the retrieved URL, username, password, and driver.

```
public class DBConnUtil {

    private static Connection connection = null;

    public static Connection getConnection() {
        try {
            Properties props = DBPropertyUtil.getDBProperties();

            String url = props.getProperty("db.url");
            String user = props.getProperty("db.user");
            String password = props.getProperty("db.password");
            String driver = props.getProperty("db.driver");

            Class.forName(driver);
            connection = DriverManager.getConnection(url, user, password);

        } catch (Exception e) {
            System.out.println("Connection Failed: " + e.getMessage());
        }
        return connection;
    }
}
```

5. main Package

Fianceapp serves as the entry point of the Finance Management System. It presents a menu-driven console interface that allows users and administrators to interact with the system. It that allows administrators and users to manage their expenses, categories, and suggestions. The

application offers features for viewing, adding, and deleting categories and suggestions, as well as managing user accounts and expenses.

Modules

1. **Admin Module** The admin has the highest level of access to the system, enabling them to manage categories, suggestions, users, and view various information regarding the users' expenses.
    - **Admin Login**: Admin must log in with a username and password. Successful login grants access to the admin dashboard.
    - Checks the validity of the provided credentials.

    Admin Actions Menu: After a successful login, the admin can perform various operations, such as:

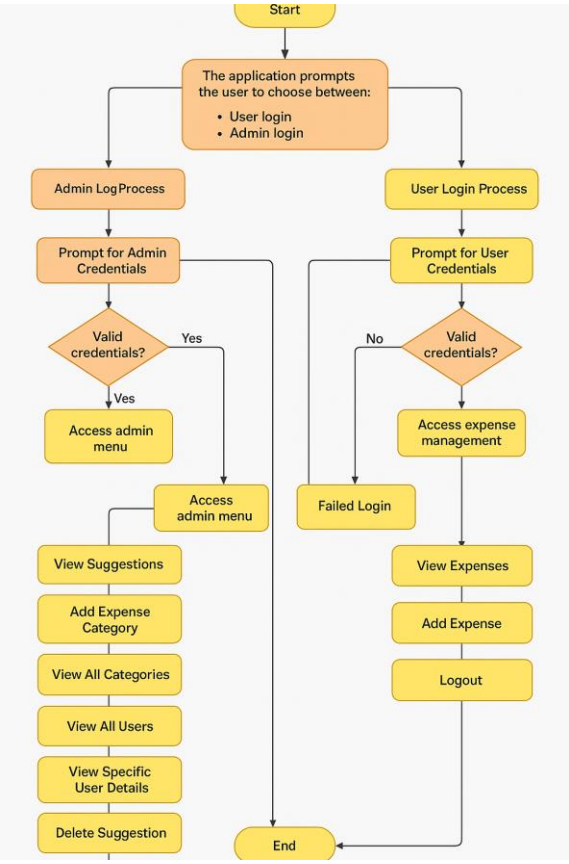
    1. **View Suggestions**: Displays all suggestions submitted by users.
    2. **Add Expense Category**: Allows the admin to add a new category for expenses.
    3. **View All Categories**: Displays all available expense categories.
    4. **View All Users**: Lists all users in the system.
    5. **View Specific User Details**: Allows the admin to view details of a specific user by entering their User ID.
    6. **Delete Suggestion**: Admin can delete a specific suggestion based on the suggestion ID.
    7. **Delete Category**: Admin can delete a specific category after ensuring that the category is not used in any expense.
    8. **Log Out**: Logs the admin out of the system.

    Handles the admin menu choices, interacting with the database (via adminRepo) to perform the actions mentioned.

2. **Expense Management**: Users can track their expenses based on categories.
    - **View Expenses**: Displays all expenses associated with the logged-in user. Each expense includes details like:
        - Expense ID
        - Category Name
        - Amount
        - Date
        - Description
3. **User Module**

- **User Management**: Admin can view the details of all users, including their usernames and emails.
- **Expense Category**: Each user can categorize their expenses using predefined categories. Admin manages the categories.
- **Suggestion Management**: Users can submit suggestions for the improvement of the app, and admins can view and delete them.

Work Flow



1. The application prompts the user to choose between:

- **User login**
- **Admin login**

2. Admin Login Process

1. **Prompt for Admin Credentials**:
   - Admin is asked to enter their **username** and **password**.
   - The credentials are verified using the adminRepo.adminLogin() method.
2. **Successful Login**:

- If credentials are valid, the admin gains access to the **admin menu** and is presented with various options to manage the app's features.

3. **Failed Login**:
   - If the credentials are invalid, the admin is informed of the failed login, and they are given the option to try again or quit the login process.

3. Admin Menu Actions

Once logged in, the admin can choose from several available actions. The admin can perform the following operations:

3.1. View Suggestions

- Admin can view all suggestions submitted by users for system improvements.
- The admin can see a list of suggestions using adminRepo.viewSuggestions().

3.2. Add Expense Category

- Admin can add new categories to the system for classifying expenses.
- Admin is prompted to input the **category name**.
- The category is added using the method adminRepo.addExpenseCategory().

3.3. View All Categories

- Admin can view all available categories within the system.
- The categories are retrieved and displayed using adminRepo.viewAllCategories().

3.4. View All Users

- Admin can view a list of all registered users in the system.
- The system retrieves user details via adminRepo.viewAllUsers().

3.5. View Specific User Details

- Admin can query the system to view the details of a specific user.
- Admin is prompted to enter a **User ID**.
- The user's information is retrieved via adminRepo.getUserById(id).
- If the user exists, their details (ID, username, email) are displayed.
- If the user doesn't exist, an error message is shown.

3.6. Delete Suggestion

- Admin can delete a suggestion from the system.
- Admin is prompted to enter the **Suggestion ID**.
- If the suggestion exists, it is deleted via adminRepo.deleteSuggestion(sugId).

3.7. Delete Category

- Admin can delete an expense category.
- Admin is prompted to enter the **Category ID**.
- The system checks if the category is in use in any expenses (using adminRepo.isCategoryUsed()).
- If the category is in use, the admin is informed that it cannot be deleted.
- If the category is not in use, it is deleted using adminRepo.deleteCategory(catIdToDelete).

3.8. Logout

- The admin can log out of the system, terminating the admin session and returning to the login screen.

4. User Login Process

1. **Prompt for User Credentials**:
    - The user is prompted to enter their **username** and **password**.
    - The system verifies the credentials against the user repository (e.g., repo.userLogin()).
2. **Successful Login**:
    - If the credentials are valid, the user is logged in and is able to access the system to view or manage their expenses.
3. **Failed Login**:
    - If the credentials are invalid, the user is informed and given the option to try again.

5. User Workflow

Once logged in, the user has access to their expense management features:

5.1. View Expenses

- The user can view all expenses that have been recorded against their account.
- The system retrieves the user's expenses (using repo.getAllExpensesByUserId(userId)).
- Each expense displays:
    - Expense ID
    - Category Name
    - Amount

- Date
- Description

5.2. Add Expense

- User can add a new expense by selecting a category, entering the expense amount, date, and description.
- The expense is then added to the system, and the user can track it along with other expenses.

5.3. Logout

- The user can log out of their account, terminating their session.

6. Exception Handling

The system includes various exception-handling mechanisms:

- **Admin Actions**: If an admin tries to perform an action with invalid data (like deleting a category in use or entering an incorrect suggestion ID), the system will display an appropriate error message.
- **User Details**: If a user is not found when the admin tries to retrieve their details, a custom exception (UserNotFoundException) is thrown, and a message is shown.

7. End

- After performing all the necessary operations, both **admin** and **user** can log out, returning to the initial login screen.
- The application exits when either the admin or user decides to log out, completing the workflow.

Output :

```
--------------------------------------------------------------------------------
                          Finance Management System
                         "Manage your money wisely!"
--------------------------------------------------------------------------------

1. User
2. Admin
3. Exit
Enter your choice: 1
--------------------------------------------------------------------------------

1. Login
2. Create User
3. Exit
Enter your choice: 2
---------------------------- CREATE USER --------------------------------

Username: Krish
Email: kresh@gmail.com
Password: eish
Account created successfully! Please login to continue.
UserId : 38, Username : 'Krish', Email : kresh@gmail.com'
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```

```
------------------------------- LOGIN ----------------------------------

Email: kresh@gmail.com
Password: eish

--------------------------------------------------------------------------------
Welcome Krish!

------------------------------ USER MENU ---------------------------------
1. Create Expense
2. View All Expenses
3. Filter Expenses
4. Update Expense
5. Delete Expense
6. Generate Report
7. Suggestion
8. Delete Account
9. Logout
```

```
Choose an option: 1
------------------------- Add New Expense -----------------------------
All Expense Categories:
Category ID     Category Name
1               Food & Dining
2               Transportation
3               Utilities
4               Entertainment
5               Healthcare
6               Education
7               Groceries
8               Shopping
9               Travel
10              Fitness & Sports
11              Insurance
12              Investment
13              Rent
14              Miscellaneous
17              Subscriptions
Enter Category ID: 4
Enter Amount: 500
Enter Date (yyyy-mm-dd): 2025-04-02
Enter Description/Note: Went theatre with friends

Expense Added!
Expense ID: 45
Amount: 500.0
Description: Went theatre with friends
--------------------------------------------------------------------------
```

```
Choose an option: 2
--------------------------- View All Expenses -------------------------------

Expense ID   Category        Amount     Date            Description
--------------------------------------------------------------------------
45           Entertainment   500.00     2025-04-02      Went theatre with friends
--------------------------------------------------------------------------
```

```
Choose an option: 3
------------------------- Filter Expenses ------------------------------
1. Filter by Date
2. Filter by Category
Enter choice: 1
Enter From Date (yyyy-mm-dd): 2025-04-01
Enter To Date (yyyy-mm-dd): 2025-04-10
Expense ID   Category        Amount     Date            Description
---------------------------------------------------------------------------
45           Entertainment   500.00     2025-04-02      Went theatre with friends
46           Groceries       4500.00    2025-04-10      monthly grocery
---------------------------------------------------------------------------
```

```
------------------------- Update Expense -------------------------------

Enter Expense ID to update: 46
All Expense Categories:
Category ID     Category Name
1               Food & Dining
2               Transportation
3               Utilities
4               Entertainment
5               Healthcare
6               Education
7               Groceries
8               Shopping
9               Travel
10              Fitness & Sports
11              Insurance
12              Investment
13              Rent
14              Miscellaneous
17              Subscriptions
Enter new Category ID (-1 to skip): -1
Enter new Amount (-1 to skip): 5000
Enter new Date (yyyy-mm-dd or 'na' to skip): na
Enter new Description ('na' to skip): na
Expense updated successfully!
ExpenseId : 46
Category : Groceries (ID: 7)
Amount : 5000.0
Date : 2025-04-10
Description : 'monthly grocery'
---------------------------------------------------------------------------
```

```
------------------------- Delete Expense ------------------------------

Enter Expense ID to delete: 48
Are you sure you want to delete this expense? (yes/no): yes
Expense deleted successfully!
--------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------
                        Finance Management System
                       "Manage your money wisely!"
--------------------------------------------------------------------------------

1. User
2. Admin
3. Exit
Enter your choice: 2

---------------------------------- ADMIN ---------------------------------
Enter admin name: admin2
Enter password: pass123
Admin login successful!
--------------------------------------------------------------------------------
------------------------------ ADMIN MENU ------------------------------
1. View Suggestions
2. Add Expense Category
3. View All Categories
4. View All Users
5. View Specific User Details
6. Delete Suggestion
7. Delete Category
8. Log Out
Enter your choice:
```

Conclusion:

I have successfully implemented all the functionalities assigned to me as part of the task, ensuring that the application meets the required objectives. The key features, such as user authentication, expense management, categorization, report generation, and database connectivity, have been completed and integrated seamlessly. Through this process, I have gained valuable experience in developing a secure, user-friendly application with a focus on financial data management. I also enhanced my skills in working with databases, handling user interactions, and implementing features that contribute to both functionality and user experience. This project has further strengthened my understanding of application development and the importance of maintaining clean, efficient code for real-world applications.