APD 545 Project Report

Personal Finance Management Application

1. Application Overview

This application was a destop app deisnged to help tracking users expense and income. The tracking included the setting budget, recording transactions, recent income and expense and data visualization tool showing recent trsaction trend.

Key Features:

1. **Transaction Management**
    - Creat, edit, delete and update income and expense transactions
    - Categorized transactions with flexibility adding categories (expense on food entertainment, Income from investmetn or salary)
    - Date-based filtering of transactions by date
2. **Budget Management**
    - Set monthly, weekly, or yearly budgets for different expense categories
    - Track spending against budgets
    - Receive warnings when spending approaches or exceeds budget limits
3. **Financial Reporting**
    - Monthly summaries of income and expenses
    - Category-based analysis with pie charts and bar graphs
    - Trend analysis to track financial patterns over time
4. **Data Visualization**
    - Interactive charts for income/expense distribution
    - Budget progress visualization
    - Monthly trend comparisons

2. Architecture and Design

MVC Architecture

The application followed the Model-View-Controller design pattern to maintain a clear separation of concerns:

Model Layer

- **Data Models**:

Java classes representing entities like `Transaction`, `Category`, `Budget`, and `FinancialSummary`

- **Database Manager**:

Handles SQLite database operations through JDBC

- **Data Access Logic**:
- Provides methods to create, read, update, and delete data

View Layer

- **Using the components of JavaFX UI**: Forms, tables, and charts that display data to the user
- **Customizing the Styling using CSS**: CSS for consistent visual presentation
- **Adding User Input Elements for transaction and budget creation**: Text fields, date pickers, buttons, and other controls

Controller Layer

- For business logic, using the cotrollers for transactions, budgets, and reports
- For input validation, it is used to ensure data input are intact and following data integrity
- Using Async opertaions so that could manage the taks om the backgroup for the database operation

Database Design

I used the SQLite for this application. It is a lightweight embedded relational database. The following schema were adopted:

- **users**: Stores user information (for future multi-user support)
- **categories**: Stores transaction categories (income or expense)
- **transactions**: Records all financial transactions with their details
- **budgets**: Contains budget settings for different categories

3. Implementation Details

Threading

As required in the project details, I used threading oo ensure UI responsiveness while performing database operations. The implementation is like the following

1. For threadPool-based Execution, the application uses `ExecutorService` to manage a pool of worker threads for database operations.
2. For Task Framework, JavaFX's `Task` class is used to execute long-running operations off the UI thread.

3. For asynchronous Callbacks, all database operations use callback functions for success and error handling, allowing the UI to remain responsive.

Here is one of the examples:

```
public void getTransactions(LocalDate fromDate, LocalDate toDate,
                            Consumer<ObservableList<Transaction>> onSuccess,
                            Consumer<Exception> onError) {
    Task<ObservableList<Transaction>> task = new Task<>() {
        @Override
        protected ObservableList<Transaction> call() throws Exception {
            List<Transaction> transactions = dbManager.getTransactions(userId, fromDate, toDate);
            return FXCollections.observableArrayList(transactions);
        }
    };

    task.setOnSucceeded(event -> {
        onSuccess.accept(task.getValue());
    });

    task.setOnFailed(event -> {
        Throwable exception = task.getException();
        onError.accept(new Exception("Failed to load transactions", exception));
    });

    executor.submit(task);
}
```

Data Validation

I did the validation for all 3 following levels:

1. **UI Level**:
    - Text formatters for currency and numeric input validation
    - Required field checks before form submission
    - Date range validation
2. **Controller Level**:
    - Input validation before database operations
    - Validation of business rules, for example budget amount must be positive
3. **Database Level**:
    - Schema constraints (NOT NULL, CHECK, UNIQUE)

- Foreign key constraints

Notification System

I also set up a notification flow to alerts theusers for the following events:

1. Alerts when spending approaches or exceeds budget limits
2. Error messages for invalid inputs
3. Success or failure notifications for database operations

4. User Interface Design

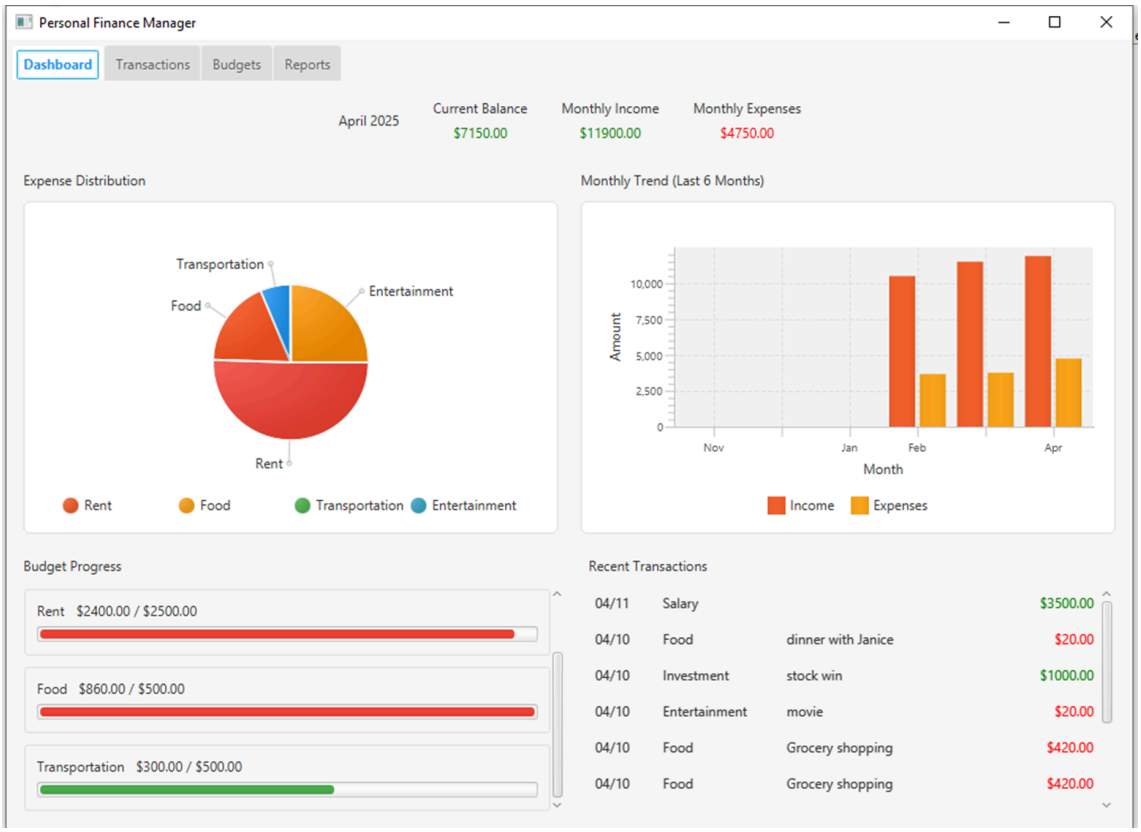
I divided the user interface into four main sections:

1. **Dashboard**: Provides an overview of financial status
2. **Transactions**: Allows users to manage their income and expense entries with filtering by date.
3. **Budgets**: Enables setting and monitoring budget limits.
4. **Reports**: Offer charts to present and summary the transaction data.

5. Data Visualization

I implemented the following data visualization tools with the avaFX chart components

1. **Pie Charts**: Show distribution of income sources and expense categories
2. **Bar Charts**: Display spending by category
3. **Line Charts**: Visualize income/expense trends over time
4. **Progress Bars**: Indicate budget utilization



6. Testing and Sample Data

To generate sample data for testing purposes. I created a object called insertSampleData() :

1. **Test Visualization Components**: See how charts and graphs display with realistic data patterns
2. **Evaluate Budget Features**: Test budget tracking with pre-populated expense data
3. **Try Filtering and Reporting**: Use date ranges to filter transactions and generate reports

In the sample data it contains

- Multiple months of transactions (income and expenses)
- Various transaction categories
- Different transaction amounts and descriptions
- Budget allocations for different expense categories, with the following as sample

```
private void addSampleTransaction(int userId, int categoryId, double amount,
                                  String description, LocalDate date) throws SQLException {
    String sql = "INSERT INTO transactions (user_id, category_id, amount, description, transaction_d
ate) " +
```

```
            "VALUES (?, ?, ?, ?, ?)";

    try (PreparedStatement stmt = connection.prepareStatement(sql)) {
        stmt.setInt(1, userId);
        stmt.setInt(2, categoryId);
        stmt.setDouble(3, amount);
        stmt.setString(4, description);
        stmt.setString(5, date.toString());

        stmt.executeUpdate();
    }
}

addSampleTransaction(1, salaryId, 3500.00, "Monthly salary", now.withDayOfMonth(5));
addSampleTransaction(1, foodId, 420.00, "Grocery shopping", now.withDayOfMonth(10));
addSampleTransaction(1, rentId, 1200.00, "Monthly rent", now.withDayOfMonth(1));
addSampleTransaction(1, entertainmentId, 85.00, "Movie night", now.withDayOfMonth(15));
addSampleTransaction(1, transportationId, 150.00, "Gas", now.withDayOfMonth(8));
addSampleTransaction(1, investmentId, 200.00, "Stock dividends", now.withDayOfMonth(22));
```

7. Known Limitations and Future Enhancements

Existing Limitations:

1. The current app only supports 1 user, as there is no login flow, although the database schema is prepared for multi-user support.
2. The app requirement manual input or creating programs for data import. It does not accept the external source data import like CSV file
3. This is only a local app and no server or cloud backup. Data will vanish after switch the device

Future Potential Enhancements:

I also come up with 3 ideas for the future enhancement

1. **User Login:** Implement user authentication and role-based access control so that mutiple users could sign in the application
2. **Data Import/Export**: Add support for CSVimport and PDF export.
3. **Recurring Transactions**: Support for scheduled recurring transactions for monthly bills like Hydro, insurace etc.