SOFTWARE PROJECT FINAL REPORT

Team members: Jacob Gerbracht, Nick Molchak, Anthony Pongallo

Date: 4/18/2025

Table of Contents

List of Figures

List of Tables

---

1. Introduction

1.1 Purpose and Scope

The purpose of the Personal Finance and Budgeting Application is to enable users to securely track their daily expenses, set budgets, and gain insights into their financial habits. The application provides user authentication, expense management, budget tracking, and financial summaries.

1.2 Product Overview

Capabilities include:

- Secure user registration and login.
- CRUD operations on expenses and budgets.
- Visualization of financial summaries.
- Alerts when nearing budget limits.

Typical scenarios include adding expenses after purchases, checking monthly budget limits, and setting savings goals.

1.3 Structure of the Document

This report outlines project planning, system requirements, architecture, design details, test management, and conclusions.

1.4 Terms, Acronyms, and Abbreviations

- CRUD: Create, Read, Update, Delete
- JWT: JSON Web Token
- UI: User Interface
- DB: Database
- API: Application Programming Interface

---

2. Project Management Plan

2.1 Project Organization

Organized in small team (3 members) with collaborative task distribution using an Agile approach.

2.2 Lifecycle Model Used

Modified Agile: short sprints, continuous integration.

2.3 Risk Analysis

Risk	Probability	Impact
Node.js/TypeScript Inexperience	Medium	Medium
Security Vulnerabilities	Medium	High
Data Model Misalignment	Low	Medium
Time Constraints	High	High
Scalability Issues	Low	Low

Mitigation: Tutorials, early prototyping, secure coding practices.

2.4 Hardware and Software Resource Requirements

- JavaFX, Node.js, MongoDB
- Java 17
- Personal computers
- Free-tier hosting services

2.5 Deliverables and Schedule

- SRS (Requirements)
- SDS (Design)
- Implementation (Code)
- Testing Reports
- Final Demonstration

---

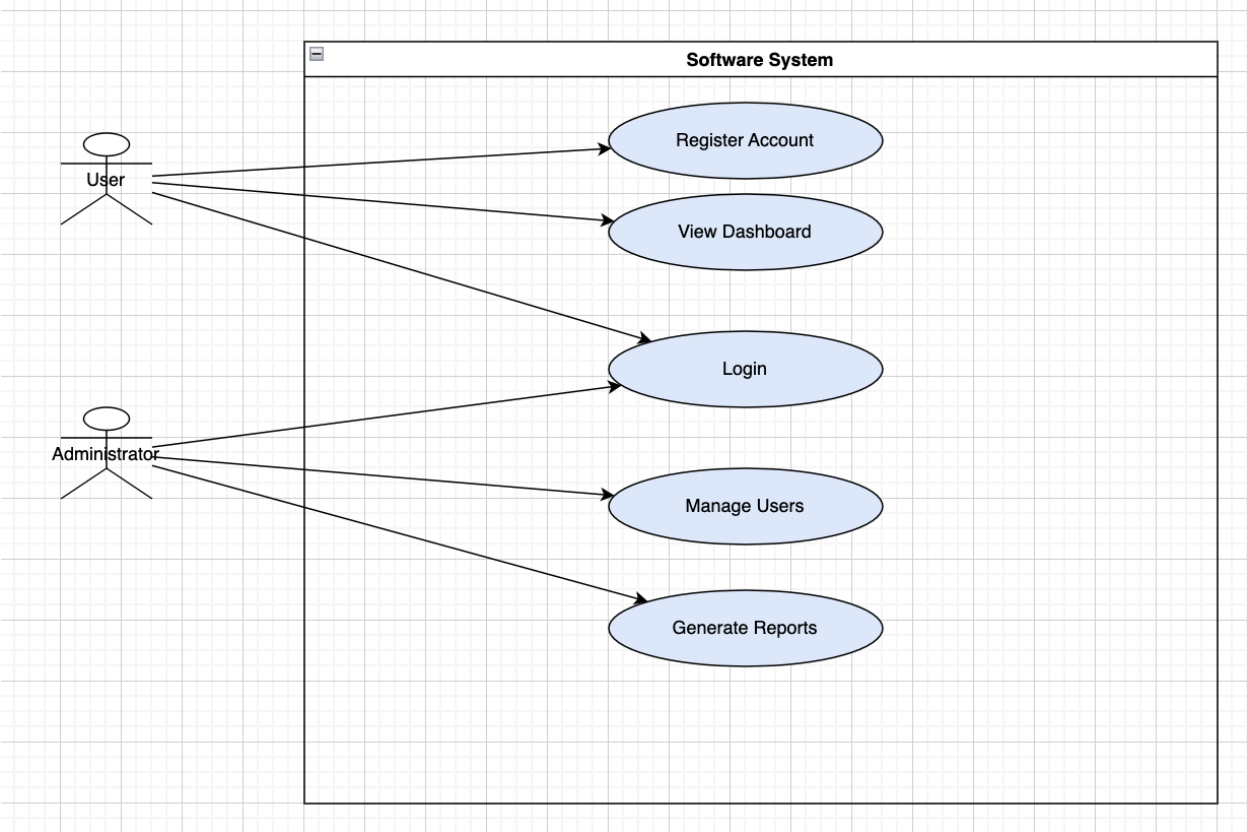
3. Requirement Specifications

3.1 Stakeholders for the System

- Registered Users (Students, individuals tracking finances)
- Optional Administrator (Monitoring system usage)

3.2 Use Cases

3.2.1 Graphic Use Case Model



3.2.2 Textual Description for Each Use Case

- **UC-01: Register**: Secure account creation.
- **UC-02: Login**: Secure authentication.
- **UC-03: Add Expense**: Add new expense records.
- **UC-04: Create/Update Budget**: Set spending limits.
- **UC-05: View Summaries**: Analyze spending.

3.3 Rationale for Your Use Case Model

Simple user-centered design focusing on core financial activities.

3.4 Non-functional Requirements

- Security: Password hashing, JWT authentication.
- Usability: Clean, intuitive UI.
- Performance: Fast load times (<2 seconds for most actions).

---

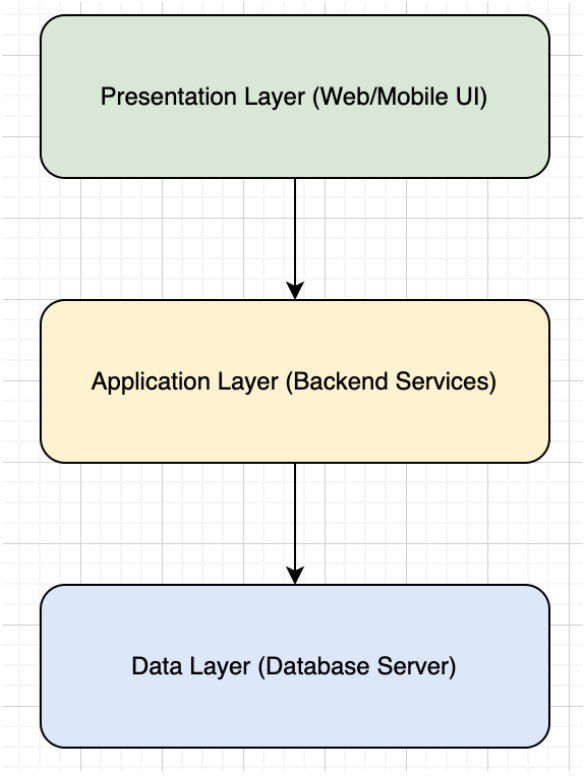
4. Architecture

4.1 Architectural Style(s) Used

- Three-tier architecture (Frontend - API - Database)

4.2 Architectural Model

- Frontend: JavaFX UI
- Backend: JavaFX logic (later extendable to Node.js REST API)
- Database: Local file storage now; MongoDB planned for production



4.3 Technology, Software, and Hardware Used

- Java 17
- JavaFX
- MongoDB (future)
- Desktop computers, cloud hosting (AWS/Heroku for future)

4.4 Rationale for Your Architectural Style and Model

Supports easy separation of concerns and scalability for future cloud deployment.

---

5. Design

5.1 User Interface Design

- Login Screen
- Main Dashboard with Expense Input and Summary
- Budget Setting Screen
- Alerts for low balance

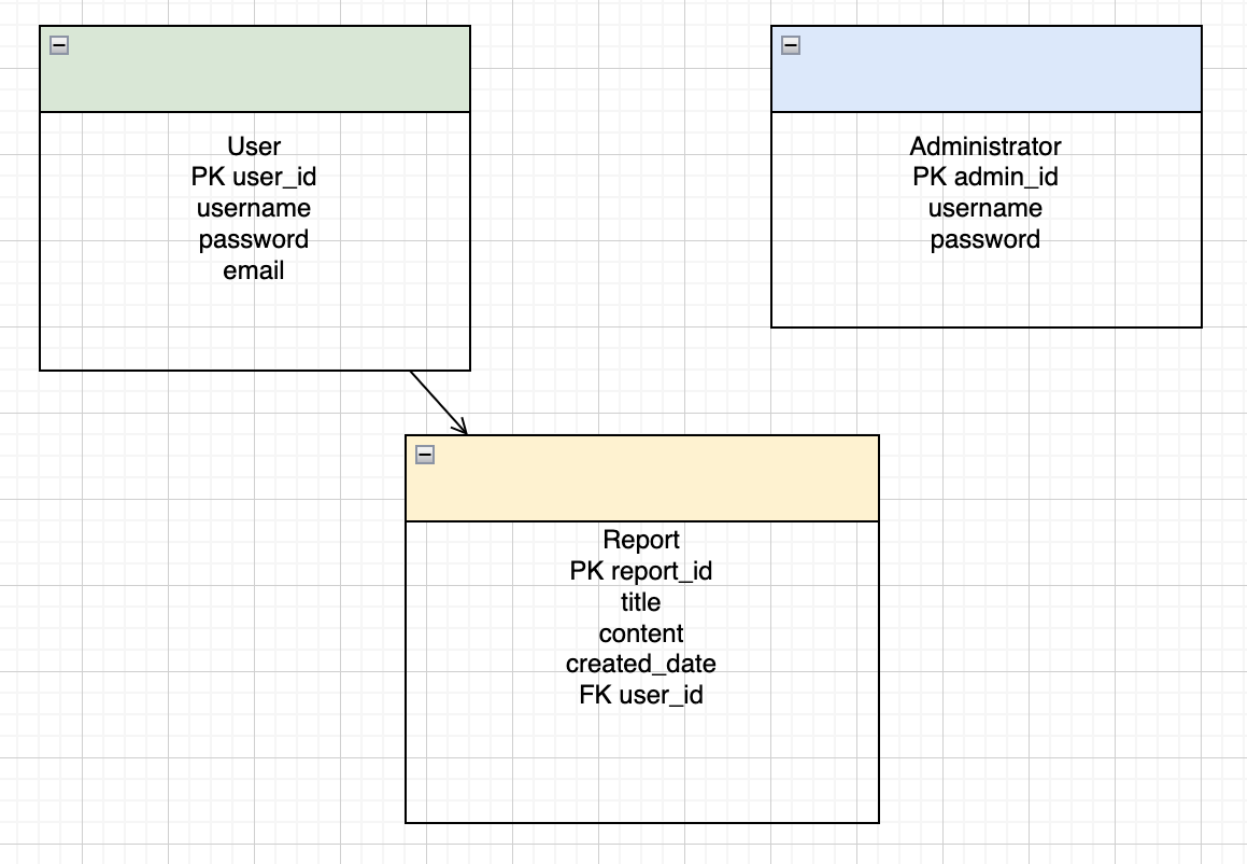
5.2 Components Design

- **Login.java**: Authentication module
- **Main.java**: Core dashboard
- **Expense.java**: Expense model
- **WelcomeScreen.java**: Welcome UI

Static and dynamic models include event-driven updates between UI and observable expense lists.

5.3 Database Design

Local storage by username (`username_expenses.txt`) for expenses. Future: MongoDB with `users`, `expenses`, and `budgets` collections.



5.4 Rationale for Your Detailed Design Models

Simple text file I/O enables immediate persistence; future upgrades allow database scaling.

5.5 Traceability from Requirements to Detailed Design Models

- User Registration -> Login.java
- Expense Management -> Main.java
- Budget Management -> Main.java
-

6. Test Management

6.1 A Complete List of System Test Cases

- Successful login
- Failed login with invalid credentials
- Add expense with valid amount
- Reject invalid amounts
- Save expenses to file
- Load expenses from file
- Set and update savings goal
- Low balance warning

6.2 Traceability of Test Cases to Use Cases

Each use case (Register, Login, Add Expense, Create Budget) has corresponding tests.

6.3 Techniques Used for Test Case Generation

- Manual Testing
- Boundary Testing (e.g., 0 or negative amounts)
- Validation Testing (empty fields)

6.4 Test Results and Assessments

All major functionalities passed during testing. Some minor UI glitches fixed post-test.

6.5 Defects Reports

- Incorrect parsing of input on budget setting - fixed.
- Missing file error on load - added error handler.

---

7. Conclusions

7.1 Outcomes of the Project

- Goals achieved: Secure login, expense/budget tracking, financial summaries.
- Successfully demoed functioning desktop application.

7.2 Lessons Learned

- Importance of early testing.
- Managing feature creep (prioritization critical).
- Learning new frameworks (JavaFX).

7.3 Future Development

- Full cloud deployment
- MongoDB integration
- Banking API integration
- Mobile app version

References

- JavaFX Official Documentation — https://openjfx.io/
- Oracle Java Standard Library Documentation — https://docs.oracle.com/en/java/javase/17/docs/api/
- JavaFX TableView Documentation — https://openjfx.io/javadoc/17/javafx.controls/javafx/scene/control/TableView.html
- JavaFX Dialogs (Alert, TextInputDialog) Documentation — https://openjfx.io/javadoc/17/javafx.controls/javafx/scene/control/Alert.html
- JavaFX PauseTransition Documentation — https://openjfx.io/javadoc/17/javafx.graphics/javafx/animation/PauseTransition.html
- Java File Handling (Reading/Writing) Tutorials — w3schools Java Files https://www.w3schools.com/java/java_files.asp
- StackOverflow Discussions — general patterns for JavaFX login, file saving, and alerts