EBU6304 Software Engineering Group Project

2024-25

Group number	58	
member 1	QM no: 221169368	Name: Haoran Sun
member 2	QM no: 221169379	Name: Yudian Wang
member 3	QM no: 221169391	Name: Zhengxuan Han
member 4	QM no: 221169483	Name: Muchi Wei
member 5	QM no: 221169357	Name: Wei Chen
member 6	QM no: 221169645	Name: Kaiyu Liu

Group report

1. The purpose and scope of the application

1.1 Purpose & Introduction

The **AI-Empowered Personal Finance Tracker** is a desktop application that helps users manage their personal finances more efficiently. Its main purpose is to simplify tracking of expenses and provide personalized spending analysis using AI. By combining automated analysis with user-driven validation, the application aims to improve financial literacy and support smarter spending decisions for all users.

1.2 Scope

1) User Definition

The application is designed for three main user roles:

- **Administrators:** Manage user accounts, oversee data import, and maintain system integrity.
- **End Users:** Register, edit profiles, record and categorize transactions, visualize financial data, and access standard membership features.
- **VIP Users:** Enjoy advanced functions such as AI-powered expense forecasting, financial health scoring, and custom cost-saving suggestions.

2) Main Features

The product scope is shaped by user needs, agile requirements analysis, and the need for extensible design. The main features include:

- **Account Management:** Secure registration, profile editing, multi-factor authentication, and account deletion.
- **Transaction Tracking:** Manual or CSV import of spending records, automatic and user-correctable categorization, and editable transaction history.
- **Financial Visualization:** Real-time expense analysis with charts, budget comparison, and consumption trend dashboards.

- **Membership System:** Display of member benefits, points management, expiry and birthday reminders, and exclusive VIP features.
- **Reminders & Notifications:** Automated alerts for key events, such as over-budget warnings and important holidays.
- **Card Management:** Add, edit, and validate multiple payment cards with secure storage.

3) Technical Notes:

The application is built with Java and JavaFX, works offline, and stores data in CSV/JSON files for portability and ease of use. It is developed using agile methods, with features prioritized and iteratively improved based on user feedback.

2. Project management

2.1 Project Planning and Iterative Development

2.1.1 First Phase: Initial Planning and Design

1) Initial Planning and Requirements Gathering

At the very beginning of the project, we clarified major constraints and deliverables, such as the overall timeline, minimum quality standards, and essential milestones (prototype, user testing, etc.), following the structured project planning process outlined in our lectures.

The team systematically gathered and analyzed requirements by **background reading**, **interviewing**, **observation**, **document sampling** and **questionnaires**, converting them into user stories using the **INVEST** criteria. Early-stage user and stakeholder interviews, as well as reference to similar software, provided a foundation for an initial product vision and backlog.

2) Task Breakdown and Scheduling

After the requirements were established, we broke down high-level user stories into smaller, actionable tasks. This allowed for parallel development and ensured that every team member had a clear focus.

Using **MoSCoW** prioritization and story point estimation, we identified must-have features for the first iterations and organized the project schedule accordingly.

2.1.2 Second Phase: Software Development

1) Sprint Planning and Execution

The project was divided into multiple sprint cycles, each beginning with a sprint planning meeting. During these meetings, we selected high-priority user stories from the backlog, estimated their complexity, and assigned them to team members.

Daily stand-ups and frequent check-ins throughout each sprint enabled the team to synchronize progress, identify and resolve blockers, and make adjustments as necessary to keep the sprint on track.

2) Review, Feedback, and Backlog Refinement

At the end of each sprint, we conducted sprint review and retrospective meetings. Completed features were demonstrated, and feedback was collected from both team members and stakeholders.

This feedback, together with findings from prototype testing and usability sessions, was incorporated into the product backlog. The backlog was regularly refined and re-prioritized to ensure our development roadmap remained flexible and responsive to user needs.

3) Continuous Improvement and Adaptation

Iterative development allowed us to quickly adapt to new requirements or unforeseen challenges. As the project progressed, we were able to update the backlog, adjust sprint goals, and shift development focus without significant disruption.

Throughout each sprint, we maintained a flexible development roadmap. Regular backlog refinement and sprint retrospectives ensured that new user insights and stakeholder feedback were promptly incorporated into future iterations, keeping the team focused on the highest-value features.

2.1.3 Third Phase: The Project Closure

After five iterations spanning about two months, we successfully completed the development of the AI Empowered Personal Finance Tracker, fulfilling all core user and system requirements. In the closure phase, the team finalized the comprehensive project report, user manual and summarized key lessons learned.

2.2 Techniques and Tools

2.2.1 Agile Methodology (Scrum)

The team adopted Scrum, following iterative development cycles and embracing change at every stage. Each sprint was time-boxed and delivered a working increment of software.

User stories were the central unit of requirement expression and tracking. These were prioritized and managed in a dynamic product backlog.

2.2.2 Version Control & Collaboration:

Git and **GitHub** were utilized for version management and team collaboration (Click here to view our repository). Each major feature or bug fix was developed in a dedicated branch and merged after review to maintain code quality and project integrity.

Communication tools (such as WeChat and online meetings) were used for instant feedback, clarifications, and team discussions.

2.2.3 Backlog and Task Management:

The product backlog was maintained as an Excel file(Click here to download our backlog). It served as both a living requirements document and a planning board, ensuring transparent task allocation and progress tracking.

2.3 Quality and Risk Management

2.3.1 Code Quality and Documentation:

The project enforced code reviews, pair programming on complex features, and comprehensive **Javadoc** documentation to maintain high quality and facilitate knowledge sharing.

JUnit tests were written for core logic, while manual testing covered integration and UI.

We also tracked metrics such as feature completion rate, and test coverage to evaluate sprint success and identify improvement areas.

2.3.2 Adaptable Architecture:

The adoption of the **Model-View-Controller** (MVC) and Loader-Controller patterns, as documented in our **README**, promoted modularity and simplified feature updates or changes in requirements.

3. Requirements

3.1 Fact-Finding Techniques

To ensure our AI Empowered Personal Finance Tracker effectively meets real user needs, we employed multiple fact-finding technique.

3.1.1 Background Reading:

To gain a comprehensive understanding of the personal finance management app market, user needs, and product trends, we conducted an in-depth literature review across multiple sources:

1) Market environment and competition:

We analyzed the rapid development of mobile personal finance tools in China, the increasingly complex financial needs of users, and the importance of differentiated product strategies [1]. This study highlighted the shift from simple bookkeeping to comprehensive financial management, integration with financial services, and the growing demand for intelligent features and scenario-based design. It also underlined the critical role of user experience, security, and responsive design in meeting the needs of a tech-savvy generation.

2) User experience and feature prioritization:

We have found an intelligent accounting robot provided valuable insights into user needs through Kano model analysis and extensive questionnaires. and interviews [2]. Key findings include the high priority users place on easy categorization, financial overview, and responsive interaction. Features such as AI-powered budget reminders, visual analytics, and engaging interactions (like voice bookkeeping and gamification) were identified as "attractive" or "must-have" for enhancing user satisfaction. The study also clarified that students and young professionals desire both practical and innovative features, which guided our focus on data visualization, AI analytics, and interactive UI in our own requirements.

3) Systematic design approaches:

From pertinent literature, we learned best practices in system architecture, modular design, and the implementation of core functionalities like secure authentication, categorized expense tracking, and budget management [3].

4) Benchmarking and functional coverage:

We referenced the competitive landscape and functionality lists provided by open-source university projects, identifying foundational and advanced features suitable for student users [4]. Our core features, including secure authentication, categorized expense/income management, AI-driven analytics, data visualization, and cross-device responsiveness, were defined in line with market and academic best practices.

Based on our literature review and analysis of industry reports, research papers, and benchmarking documents, several key requirements emerged for a modern personal finance tracker:

- Users expect the system to provide automated budget suggestions and personalized savings goals, leveraging AI to enhance financial planning and self-discipline.
- Cost-cutting and savings recommendations are highly valued, helping users identify unnecessary expenses and optimize their financial habits.
- Data visualization and analytics are essential, as users want to easily interpret their financial situation through interactive charts and reports.
- Cross-platform compatibility and responsive design are increasingly important, as users access financial tools from various devices.
- Data privacy and security must be a top priority, ensuring that sensitive information is protected in line with best practices.

3.1.2 Observation & Brainstorming:

We observed typical user behaviors and pain points through informal usability testing and group brainstorming sessions. Team members shared their own finance management challenges, further informing requirements related to UI simplicity, ease of onboarding, and reminder mechanisms.

After observing user behaviors and conducting team brainstorming sessions, we identified the following functional needs and usability improvements:

- Users want the ability to review and correct expense categorizations, increasing accuracy and trust in their financial records.
- Interface simplicity and easy onboarding were repeatedly highlighted, motivating us to streamline workflows and provide clear instructions for new users.
- Users frequently desire reminders about upcoming payments, budgets, or unusual spending, so notification features and timely alerts were prioritized.
- Many users expressed a desire for the app to detect seasonal or location-based spending spikes—such as during festivals or holidays—and to adjust budget suggestions accordingly.
- Visualization of spending history and savings trends through interactive dashboards greatly improves user engagement and satisfaction.

3.1.3 Document Sampling:

Core requirements were extracted from the project handouts, marking criteria, and feature lists described in our backlog and requirements documentation. This ensured our implementation would cover all essential use cases, security expectations, and compliance needs.

- The system must support both manual and automated data entry, including the ability to import CSV files from other platforms.
- Expense categorization should use AI for automation but must allow manual verification and correction by users.
- Spending insights, predictions, and reminders are expected, with the system providing users with monthly budget and savings suggestions as well as cost-cutting tips.
- The tracker should consider local financial contexts and seasonal variations, such as higher expenses during national holidays.
- Security, flexibility, and offline operation are mandatory: the app must run as a stand-alone Java desktop application, store data in plain files (not databases), and enable easy maintenance and future extensibility.
- Administrative functions must include user lookup, benefit assignment, and record editing to ensure accurate data management.

3.1.4 Questionnaires

Building on our initial observations and interviews, we created a user questionnaire in Chinese to better understand our target audience's needs (Click here to view questionnaire). Most respondents were students or young adults aged 18–35, our main user group. The key findings include:

- **Financial Management Habits:** Most participants manage their finances by themselves, with 70%+ indicating they handle their own financial records and budgeting.
- **Challenges Encountered:** The most common issues users face are unclear budget planning, overspending, and the lack of effective reminders. Many respondents highlighted difficulties in keeping track of consumption and distinguishing between necessary and unnecessary spending.
- **Feature Demand:** There is a strong demand for functions such as budget planning, spending warnings/alerts, personalized financial suggestions, category management, and the ability to view detailed spending statistics and trends. Visualizations and interactive dashboards are highly preferred for a clear financial overview.
- **Preferences for AI/Intelligent Features:** Many users expressed interest in smart reminders, automatic categorization, savings suggestions, and intelligent consumption analysis. Key words from feedback include "budget," "recommendations," "overspending warning," "financial planning," and "consumption analysis."
- **Expectations on Usability:** Users emphasized the importance of simplicity and ease of use, clear categorization, input validation, and convenient data entry and export. Some also hoped for community/interaction functions and more personalized insights.

They highlight the need for:

- Budget and savings planning features
- Cost-cutting suggestions
- Spending classification and review
- Alerts for overspending and reminders
- Interactive data visualization
- Simplicity and user-friendly design

3.2 Iterative Requirement Gathering and Planning

We adopted an **iterative and incremental approach** for requirement discovery and refinement, aligning with Agile and Scrum methodologies taught:

1) User Stories and INVEST Principles:

All requirements were structured as user stories using the INVEST criteria (Independent, Negotiable, Valuable, Estimable, Small, Testable), making each feature clear and actionable.

2) Backlog Prioritization and Estimation Planning:

Our product backlog, maintained in Excel, was continuously updated and prioritized using MoSCoW (**Must-have**, **Should have**, **Could have**, and **Want to have**) prioritization rules to systematically assign priority levels. Features identified as must-have were given very high priority, ensuring they would be delivered first to meet core user needs and essential system requirements such as secure authentication, expense categorization, budget reminders, and AI analytics received top priority. Conversely, features classified as want to have were assigned low or very low priority, as these elements were considered non-essential and would only be developed if time and resources permitted. We then used story points to estimate user stories and facilitate sprint planning.

Table 1 Prioritization and Estimation Planning

Story ID	Story Name	MoSCoW	Priority	Iteration Plan	Story Point
1	Open Account	Must Have	very high	1	3
2	Add or Modify User Information	Must Have	high	1	3
3	Security Verification	Must Have	very high	1	5
4	Querying User ID	Must Have	high	1	2
5	Import Consumption Data	Must Have	very high	2	5
6	Classify Transactions	Must Have	very high	2	8
7	View Categories	Must Have	very high	2	2
8	Expense Editing	Must Have	high	2	3
9	Visualization Report	Must Have	high	2	5
10	Display of Member Benefits	Must Have	high	3	2
11	Consumption habit Forecasting	Must Have	high	3	8
12	Manage Cards	Should Have	high	1	5
13	Query User Level	Should Have	medium	1	2
14	View the remaining membership time	Should Have	high	1	2
15	Cash Flow Visualization	Should Have	high	2	5
16	Comparison of Budget with Actual Expenditure	Should Have	medium	3	5
17	Consumption Reduction Suggestion System	Should Have	medium	4	8
18	View membership rewards points	Could Have	medium	4	2
19	Membership expiration reminder	Could Have	medium	4	3
20	Holiday Reminder	Could Have	medium	4	3
21	Financial Health Scoring	Could Have	medium	4	8
22	Smart over Budget Alerts	Could Have	low	5	5
23	Holiday Budget Proposal	Want Have	very low	5	5
24	Main Menu Navigation	Should Have	high	3	2
25	VIP Membership Recharge	Should Have	medium	4	3
26	Financial Analysis System	Should Have	medium	4	8

3) Iteration and Feedback Loops:

Requirements were revisited at the end of each sprint based on prototype feedback, usability tests, and bug reports. This agile cycle enabled rapid response to changing needs and ensured continuous alignment with user expectations.

3.3 Requirement Prioritization and Final Feature Set

By combining **Kano model analysis**, interviews, and survey results, we were able to distinguish between must-have, expected, and attractive features:

- **Must-have requirements:** Secure login, expense and income tracking, category management, responsive UI, and clear data visualization.
- **Expected features:** Budget setting, spending reminders, quick editing/deletion, and report export.
- **Attractive (delighter) features:** AI-powered expense prediction, financial health scoring, interactive dashboards, and personalized saving suggestions.

This process enabled us to deliver a robust, user-centred financial tracker while maintaining the flexibility to adapt as requirements evolved during development.

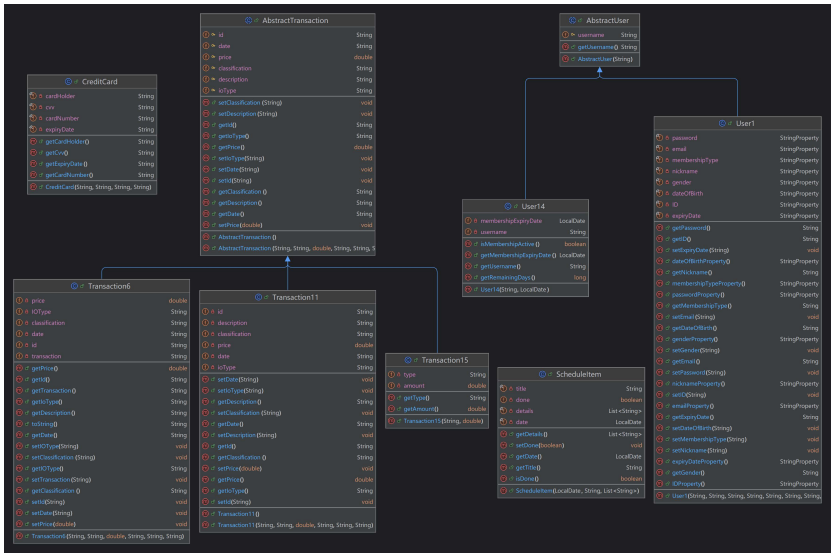
4. Analysis and Design

4.1 Identify Entity, Controller, and Boundary Classes

We separated the source codes of the Java file into 5 parts- Entity, Boundary, Controller, model, delayer classes.

4.1.1 Entity Classes

The code should follow the principle of high cohesion and loose coupling to increase the quality of code.

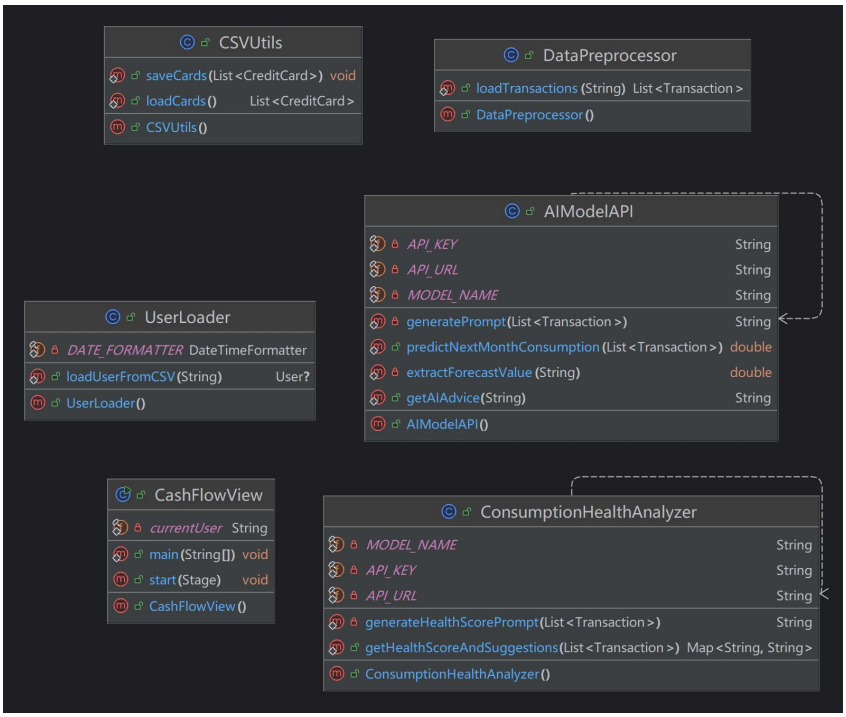


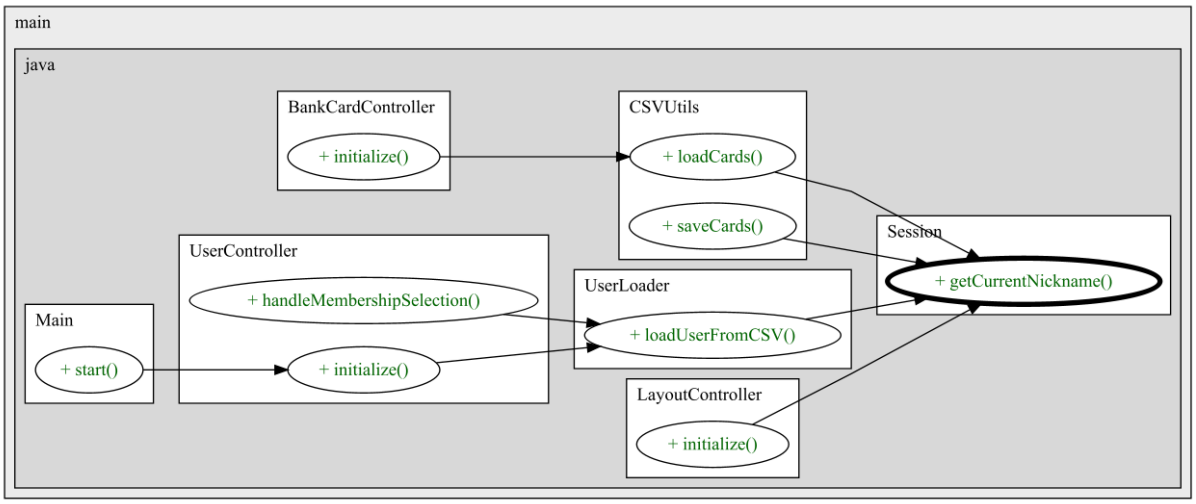
4.1.2 Control Classes

We use the control class to link the Boundary class to the database and the Entity class. It is primarily responsible for processing data requests sent by the Boundary class and linking to the corresponding database based on the content of the request. After getting the data from the database, it parses the return packet and returns the specific data to the Boundary class.

4.1.3 Boundary Classes

The boundary class is responsible for presenting our UI to the users, listening to their actions, and interacting with them. The boundary class is at the top of our software architecture and only interacts with the user and control classes, making our software less coupled.





4.2 Design Principles

4.2.1 DRY (Don’t repeat yourself)

The DRY principle discourages code duplication to improve maintainability. Throughout our project, we identified repeated logic—especially related to FXML loading for various UI windows—and extracted it into a utility class.

For instance, the logic for loading and displaying FXML files (FXMLLoader, load(), getController(), Stage.show(), etc.) was reused across multiple features. We centralized this code into a utility method like FXMLUtils.loadAndShow(String fxmlPath,

Consumer<Controller> initializer), making future updates or error handling adjustments easier to manage in a single location.

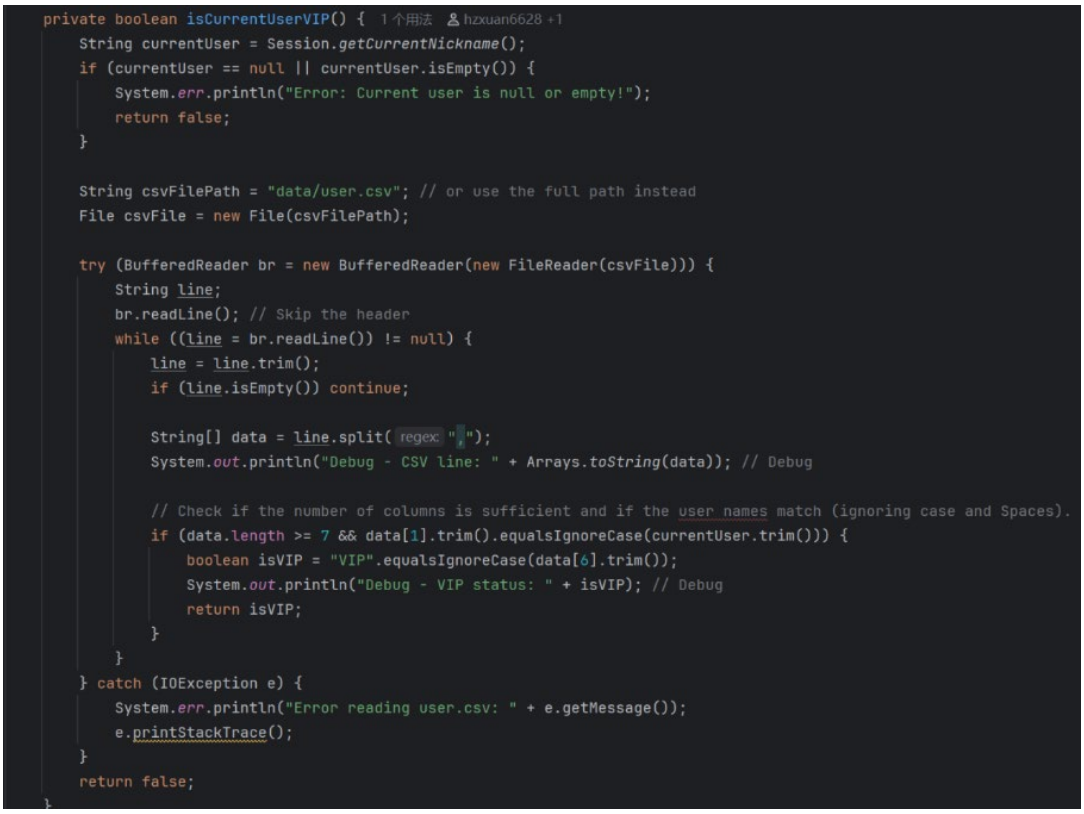
```
private void showAnalysisReport(String reportText) {  1 个用法   Wei13461 +1
    try {
        FXMLLoader loader = new FXMLLoader(getClass().getResource( name: "/src/main/resources/Category_story6_7/AnalysisReport.fxml"))
        Pane reportPane = loader.load();
        AnalysisReportController reportController = loader.getController();
        reportController.setReportText(reportText);

        Stage stage = new Stage();
        stage.setScene(new Scene(reportPane,  v: 1200,  v1: 675));
        stage.setResizable(false);
        stage.showAndWait();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

4.2.2 SRP (Single Responsibility Principle)

The Single Responsibility Principle is a guideline for achieving high cohesion and low coupling.

We discovered the different responsibilities of classes and separated them, so that one class is responsible for the corresponding responsibilities in only one functional area, giving high cohesion. The code inside a module will not be affected when other functions change, giving loose coupling.

For example, we created a method called isCurrentUserVIP(). This method is only responsible for verifying the VIP status of the currently logged-in user by reading the user information from a CSV file. It does not deal with any UI or unrelated logic, which makes the method highly cohesive and focused on a single responsibility.



```
private boolean isCurrentUserVIP() {  1 个用法   hzxuan6628 +1
    String currentUser = Session.getCurrentNickname();
    if (currentUser == null || currentUser.isEmpty()) {
        System.err.println("Error: Current user is null or empty!");
        return false;
    }

    String csvFilePath = "data/user.csv"; // or use the full path instead
    File csvFile = new File(csvFilePath);

    try (BufferedReader br = new BufferedReader(new FileReader(csvFile))) {
        String line;
        br.readLine(); // Skip the header
        while ((line = br.readLine()) != null) {
            line = line.trim();
            if (line.isEmpty()) continue;

            String[] data = line.split( regex: ",");
            System.out.println("Debug - CSV line: " + Arrays.toString(data)); // Debug

            // Check if the number of columns is sufficient and if the user names match (ignoring case and Spaces).
            if (data.length >= 7 && data[1].trim().equalsIgnoreCase(currentUser.trim())) {
                boolean isVIP = "VIP".equalsIgnoreCase(data[6].trim());
                System.out.println("Debug - VIP status: " + isVIP); // Debug
                return isVIP;
            }
        }
    } catch (IOException e) {
        System.err.println("Error reading user.csv: " + e.getMessage());
        e.printStackTrace();
    }
    return false;
}
```

4.2.3 OCP (Open-Closed Principle)

The Open-Closed Principle means that a software entity should be open for extension and closed for modification. That is, the software entity should extend as much as possible without modifying the original code.

In our design, we create a utility method called formatDate to convert a LocalDate object into a string with the pattern "yyyy-MM-dd". When other parts of the system require date formatting, they can reuse this method directly instead of rewriting the same logic. Moreover, if in the future we want to support different date formats (e.g., "MM/dd/yyyy"), we don't need to modify the original method — we can add an overloaded method that accepts a custom pattern, thus extending the functionality without changing the existing code. This approach follows the Open-Closed Principle by allowing format logic to evolve while keeping the original method untouched and stable.

```
private String formatDate(LocalDate localDate) {  1 个用法  Wei13461
    if (localDate == null) {
        return "";
    }
    DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd");
    return localDate.format(formatter);
}
```

This allows the system to handle new formatting styles by extension rather than modification, fulfilling the goal of being open for extension and closed for modification.

4.2.4 LSP (Liskov Substitution Principle)

In LSP, replacing an object of the base class with an object of its subclass in the software will not generate any errors and exceptions. The reverse is not true. If a software entity uses a subclass object, it may not be able to use the base class object. Hence, we define objects by using the base classes as much as possible and subclass types are determined at run time to replace superclass objects with subclass objects.

For example, in our saveTransactionToCSV() method, we operate on the Transaction object, which can be designed as a base class. If we later introduce subclasses like IncomeTransaction or ExpenseTransaction, they can still be passed into this method without changing any logic, because all required methods (such as getId(), getTransaction(), getPrice() etc.) are defined in the base class. This follows LSP since these subclasses can replace Transaction without causing errors, and the method continues to work correctly at runtime.

```
private void saveTransactionToCSV(Transaction transaction) {  1 个用法  SUN Haoran +2
    try (BufferedWriter bw = new BufferedWriter(new FileWriter( fileName: "data/" + currentUser + "_transaction.csv",  append: true))) {
        // Append the new record to the file
        bw.write( str: transaction.getId() + "," + transaction.getTransaction() + "," + transaction.getPrice() + "," +
                transaction.getClassification() + "," + transaction.getDate() + "," + transaction.getIOType());
        bw.newLine();
    } catch (IOException e) {
        e.printStackTrace();
        // Show an error alert if saving fails
        Alert alert = new Alert(Alert.AlertType.ERROR,  s: "Error saving to CSV file.");
        alert.showAndWait();
    }
}
```

5. Implementation

5.1 Configuration Management

We adopted **Git** and **GitHub** as our configuration management tools to support collaborative development, ensure traceability, and manage version control. Each group member was assigned some feature-specific branch to develop independently. This strategy facilitated parallel development and minimized merge conflicts.

When a team member develops a new feature, they first create a **dedicated feature branch** from the main branch. Development and testing are performed entirely within this isolated branch to prevent instability in the main codebase. Once the feature has passed all relevant unit tests and peer reviews, the branch is merged into the main branch using a **pull request**, ensuring traceability and accountability of changes.

Our project repository is structured for modular development and includes proper documentation and JavaDocs for maintainability (see README.md for details).

5.2 Build Plan and Integration Strategy

Following the **incremental build strategy**, we divided the implementation process into manageable iterations. Each iteration delivered a working build that passed predefined JUnit tests and was suitable for demonstration.

Our **Integration Build Plan** included:

- **Iteration 1**: Core features like user login, registration, and transaction import.
- **Iteration 2**: Data visualization (PieChart, BarChart), user information editing, and card management.
- **Iteration 3**: AI-based modules such as forecasting, savings advice, and financial health scoring.
- **Iteration 4**: GUI refinement, VIP system, reminders, and full system integration.
- **Iteration 5**: Bug fixing, functionality polishing, code cleanup and optimization.

Each build was followed by integration testing to ensure compatibility between modules and stable performance.

Table 2 Build Plan

Build Plan	Functionality	Implementation
1	Function modules configuration	Login_story1_3, Category_story6_7, financial_story9, card_management_story12, ViewMembershipTime_story14, CashFlowVisualization_story15
	Entity Implementation	Class Transaction, Class CreditCard, Class User, Class ScheduleItem
2	Controller classes	Class BankCardController, Class CashFlowController, Class GifDisplayController, Class AddController, Class EditController, Class MainController, Class FinancialController, Class LayoutController, Class LoginController, Class RegistrationController, Class UserManagementController, Class UserController
	Basic UI pages	card_manager.fxml, ui.fxml, main.fxml, main_view.fxml, MainLayout.fxml
	Utility Classes	Class CSVUtils
3	Function modules configuration stage II	AI_story11_21_22, Query_story4_13, Reminder_story_19_20, Suggestion_story_17
	Controller classes stage II	Class ConsumptionForecastController, Class AnalysisReportController, Class UserSearchController, Class ScheduleListController, Class SuggestionController, Class RechargePopupController
	Basic UI pages stage II	ConsumptionForecast.fxml, UserSearch.fxml, ReminderView.fxml
	Utility Classes stage II	Class AIModelAPI, Class ConsumptionHealthAnalyzer, Class DataPreprocessor, Class UserLoader
4	Personal UI Pages	add.fxml, AnalysisReport.fxml, edit.fxml, suggestion.fxml, Login.fxml, Registration.fxml, UserManagement.fxml, UserSearch.fxml, recharge_popup.fxml
	External style definition	consumption-forecast.css, styles.css(CashFlowVisualization_story15, Login_story1_3, Reminder_story_19_20, ViewMembershipTime_story14), rechargestyle.css

5.3 Mapping Design to Code

We followed the **Model-View-Controller (MVC)** and **Boundary-Entity-Controller** architecture:

- **Model (Entity)**: JavaBeans such as User, Transaction, and ScheduleItem, responsible for business logic and data.
- **View (Boundary)**: FXML-based JavaFX interfaces with responsive layout and styling.
- **Controller (Controller)**: JavaFX @FXML-annotated controllers that handle logic, data binding, and user interactions.

Class diagrams and use-case mappings were directly translated into implementation classes using object-oriented principles such as encapsulation, inheritance (extends), and interface (implements).

5.4 Refactoring

Throughout the development process, we continuously identify new challenges in the software architecture. These challenges often arise not from flaws in the initial design, but from evolving requirements that render the existing architecture less suitable.

When we detect bloated or non-reusable code, we prioritize refactoring to streamline the architecture. Although refactoring may temporarily slow project progress, it is essential for maintaining a robust and maintainable codebase.

5.5 Pair Programming and Agile Practices

We implemented **pair programming** during complex module development (e.g., AI-based analysis and financial forecasting), which served as a lightweight substitute for formal code review. This practice improved error detection and knowledge sharing.

Our agile practices also included:

- Short development iterations with working software at the end of each cycle.
- Frequent team meetings and sprint retrospectives.
- Continuous testing using JUnit to verify all functional and AI components.

6. Testing

6.1 Testing Strategy

This project adopted a hybrid testing strategy that systematically integrates unit testing and system testing to ensure rigorous and repeatable testing.

The testing implementation was structured in following phases:

① **Unit Testing Implementation:** Individual components were tested in isolation. JUnit annotations (e.g., @Test, @BeforeEach) were used to validate constructors, getter/setter methods, and business logic, ensuring each unit met functional specifications. For example, TransactionTest verified data initialization, while LoginControllerTest examined email validation and password encryption logic. Conducted concurrently with module development, spanning 8 weeks (2 weeks per sprint).

② **System Testing Integration:** Post-unit testing, integrated system tests were conducted to evaluate end-to-end workflows across modules (e.g., user registration via RegistrationController, for details, see 6.5.). Scenario testing was employed to simulate real-

user operations. Performed over 2 weeks post-integration, including regression testing after each code modification to ensure existing features remained unaffected

Testing criteria:

① **Functional Coverage:** 100% test case execution for all specified requirements, including white-box testing (code path coverage), black-box testing (input/output validation), and scenario-based validation (real-world usage simulation).

② **Quality Metrics:** A pass rate of 100% for all test cases, with no defects identified during testing.

6.2 Testing Techniques

The design of test cases adhered to a combination of testing principles, which are elaborated as follows:

① White-Box Testing

Focused on verifying the internal logic and code paths of the software. By intricately examining the internal structure of the code, this testing method aimed to ensure that all code segments executed correctly and as intended, thereby identifying any potential issues within the codebase.

② Black-Box Testing

Centered on validating the correctness of functional inputs and outputs. From an end-user perspective, it checked whether the system functions aligned with the specified requirements, without delving into the internal workings of the code. This method ensured that the system behaved as expected under various input conditions.

③ Regression Testing

Conducted to ensure that the introduction of new features or modifications to the existing codebase did not inadvertently disrupt the functionality of existing features. It played a crucial role in maintaining the integrity of the system as it evolved over time.

6.3 Testing Environment

Development Environment: Java JDK 21.0.1

Testing Framework: JUnit 5.10.0

Auxiliary Tools: JavaFX was utilized for certain UI-related tests, facilitating the examination of user interface components and their interactions.

6.4 Testing Results

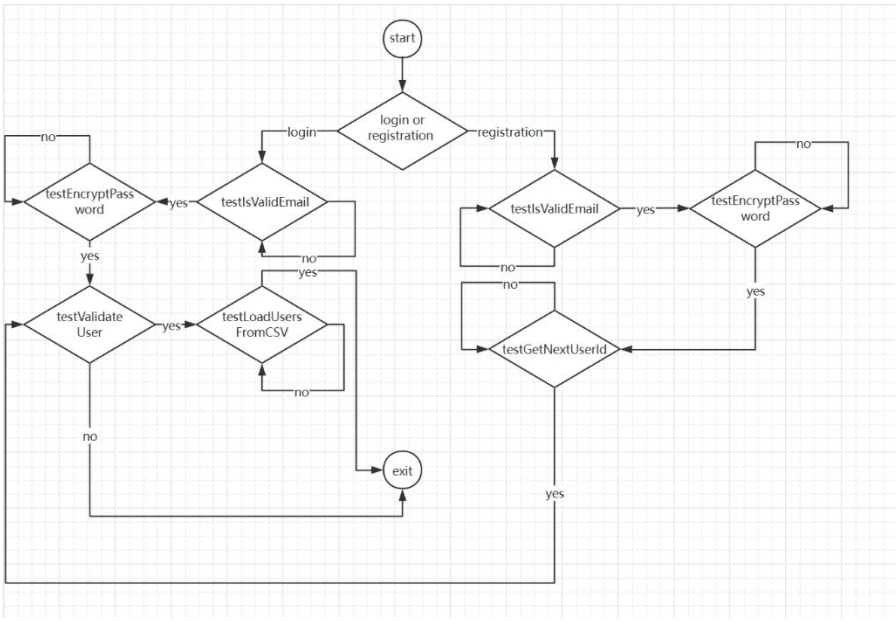
Achieved a remarkable 100% pass rate, with all test cases executed successfully and meeting the expected criteria.

Table 3 Test Items and Results

Test ID	Test Module	Test Case	Testing Result	Pass/Fail	Comments
1	TransactionTest	Testing constructors and getter methods	Passed	P	Verifies the initialization and property retrieval of the Transaction model
2	TransactionTest	Testing setter methods	Passed	P	Verifies the attribute setting of the Transaction model
3	CreditCardTest	Testing constructors and getter methods	Passed	P	Verifies the initialization and property retrieval of the CreditCard model
4	CSVUtilsTest	Testing saving and loading of credit card data	Passed	P	Verifies CSV file operations for credit card data
5	MainAppTest	Testing main method	Passed	P	Verifies the startup of the JavaFX application in a headless environment
6	AddControllerTest	Testing getNextId method	Passed	P	Verifies transaction ID generation logic
7	EditControllerTest	Testing getNextId method	Passed	P	Verifies ID generation logic when editing transactions
8	LoginControllerTest	Testing email validation	Passed	P	Verifies the correctness of email format
9	LoginControllerTest	Testing password encryption	Passed	P	Verifies password encryption functionality
10	LoginControllerTest	Testing user validation	Passed	P	Verifies user login logic
11	RegistrationControllerTest	Testing password encryption	Passed	P	Verifies password encryption during registration
12	RegistrationControllerTest	Testing email validation	Passed	P	Verifies email format during registration
13	RegistrationControllerTest	Testing user ID generation	Passed	P	Verifies user ID generation logic
14	UserSearchControllerTest	Testing user model	Passed	P	Verifies the correctness of user data
15	UserSearchControllerTest	Testing VIP and regular user classification	Passed	P	Verifies user classification logic
16	ScheduleItemTest	Testing constructors and getter methods	Passed	P	Verifies the initialization and property retrieval of the ScheduleItem model
17	ScheduleItemTest	Testing completion status switching	Passed	P	Verifies the switching logic of ScheduleItem completion status
18	SuggestionControllerTest	Testing generation of advice prompts	Passed	P	Verifies AI suggestion generation logic
19	SuggestionControllerTest	Testing CSV file parsing	Passed	P	Verifies CSV file parsing for transaction data
20	UserModelAndLoaderTest	Testing VIP user loading	Passed	P	Verifies loading of VIP user data
21	UserModelAndLoaderTest	Testing non-VIP user loading	Passed	P	Verifies loading of non-VIP user data
22	UserModelTest	Testing user model	Passed	P	Verifies the initialization and property retrieval of the user model

6.5 Test case design

We carried out the basis path test method for every aspect of our system, and analyzed each test result, and produced the corresponding test matrix. Due to the limitations of the report, we now show one demonstration as follows:



7. Future iterations

To further enhance user experience and value, the following features are planned for future development. Each builds on current architecture and leverages existing AI/ML capabilities, ensuring seamless integration with the system's core modules.

7.1 Smart Over-Budget Alerts (Priority: Low)

This feature will provide intelligent, personalized spending alerts, helping users avoid exceeding their budgets and encouraging healthier financial habits.

7.1.1 Integration with Existing Features

The smart alert system will leverage real-time **transaction categorization (Feature 6)** and **cash flow visualization (Feature 15)** to continuously monitor spending by category. Interactive alert overlays will be added to the **main visualization dashboard (Feature 24)**, enabling users to instantly see and respond to budget warnings directly within their financial charts.

7.1.2 Innovations

Adaptive Thresholds: Instead of static alerts, the system will use ML models (from Feature 11) to analyze user behavior and adjust thresholds based on spending patterns, distinguishing, for example, between one-off large purchases and recurring overspending.

Personalized Recommendations:

- Normal users receive basic budget tips.
- VIP users get AI-generated "what-if" scenarios (e.g., “Reducing dining-out by 10% keeps you within budget”).

7.1.3 Technical Synergy

- Reuses existing **ML infrastructure (Feature 11)** for real-time processing.
- Integrates with the **notification system (Feature 19 membership expiration reminders)** for consistent alerts across multiple channels (email, in-app).

7.1.4 Workflow Impact

- Users can link alert insights to **expense editing (Feature 8)** by directly modifying transaction categories or amounts from the alert prompt, streamlining error correction.
- Administrators will gain anonymized aggregate alert data to refine **budget comparison algorithms (Feature 16)** for future iterations.

7.2 Holiday Budget Proposal (Priority: Very Low)

7.2.1 Overview

Enables users to plan, simulate, and optimize holiday-specific budgets based on historical spending data and seasonal trends.

7.2.2 Integration with Existing Features

The holiday budget proposal feature will utilize **imported consumption data (Feature 5), transaction categorization (Feature 6), and holiday reminders (Feature 20)** to identify and analyze seasonal spending trends. It further extends the **budget comparison system (Feature 16)** by supporting temporary, scenario-based “holiday budgets” tailored to specific seasonal events.

7.2.3 Innovations

Dynamic Scenario Planning: Users can simulate holiday budgets using "**sandbox mode**", allowing risk-free adjustments to allocations (e.g., increasing "gift" spending by 30% while reducing "transport" by 15%) without affecting real budgets.

ML and GenAI:

- Leverages ML from Feature 11 to highlight high-impact holidays.
- VIP users receive AI-powered gift or saving recommendations tied to their proposals (cross-linking with Feature 17).

7.2.4 Technical Approach

- Develop a **holiday-specific data model** that inherits from the existing transaction schema, enabling easy integration with **Feature 9's visualization tools** (e.g., generating holiday-specific pie charts).
- Use generative AI (**GenAI**) (as explored in Feature 8's potential future use) to craft natural-language budget rationales (e.g., "Your proposed Lunar New Year budget aligns with 2024 spending on family gifts").

7.2.5 User Impact

- Bridges the gap **between financial health scoring (Feature 21)** and proactive management by translating scores into actionable holiday plans (e.g., users with low savings rates receive stricter holiday budget guidelines).
- Enhances **VIP exclusivity** by offering early access to holiday budget tools, leveraging Feature 10's membership benefit hierarchy.

7.3 Implementation Roadmap

These enhancements will be delivered in **Iteration 5 (May 19–30, 2025)** using a phased approach:

Phase 1: Launch core alert and holiday budget features, reusing existing logic.

Phase 2: Roll out ML-driven personalization and generative AI-powered recommendations for advanced users.

Strategic Vision:

These iterations reinforce the product's positioning as a holistic financial companion by:

1. Extending ML/AI capabilities (already used in Feature 11) to real-time decision-making.
2. Strengthening user engagement through context-aware interactions (e.g., alerts and proposals that evolve with spending habits).
3. Capitalizing on existing data infrastructure (transaction history, membership tiers) to deliver low-cost, high-value features.

8. The use of Generative AI (GenAI)

8.1 Generative AI in Development

Tools Used: ChatGPT o4-mini, Figma AI (for interface design assistance), Deepseek R1

Throughout the development of the *AI Empowered Personal Finance Tracker*, GenAI tools were selectively applied across various stages to assist with ideation, draft generation, and debugging. However, all outputs were carefully reviewed, modified, or restructured by the team to ensure technical feasibility, project alignment, and maintain human-led decision-making.

Planning & Requirement Stage:

ChatGPT was used as a brainstorming partner during early planning. It helped refine vague ideas (e.g., "budget reminder") into more concrete stories and inspired additional features like card management and AI-powered predictions. However, its suggestions were critically assessed. For instance, we downgraded its proposed "membership points system" due to workload concerns and rejected third-party API integration due to offline constraints.

Design Stage:

Figma AI and ChatGPT were used to generate GUI mockups and preliminary class diagrams. While these drafts accelerated the design process, they often required iterations due to aesthetic mismatches or architectural oversights. The final designs reflected team preferences and practical limitations.

Testing Stage:

ChatGPT helped generate basic JUnit test templates and edge case considerations. It accelerated test case coverage, but for complex UI scenarios and bulk data tests, we relied on manual design and execution.

Critical Reflection:

Generative AI significantly improved efficiency during routine tasks and documentation drafting. However, its limitations—such as occasional overengineering, incomplete local context understanding, and lack of domain specificity—reinforced the importance of human oversight and domain knowledge. Ultimately, GenAI served as a collaborative assistant rather than a replacement for systematic software engineering.

8.2 Runtime Integration of AI Models

Tools Used: Qwen2.5 72b

Beyond development assistance, our application integrates lightweight AI models at runtime to power intelligent features such as **consumption forecasting** and **financial health analysis**. This integration is encapsulated in the AI module (Story11, 21).Model Invocation via Encapsulated API handles the actual call to the external AI model. It prepares the input vector and invokes a local or remote prediction interface. This design ensures separation of concerns and allows model replacement or offline fallback without affecting the controller logic.

This can realize our Financial Health Scoring System. Financial Health Scoring uses rule-based logic combined with the AI model outputs to assess users' financial behavior. For example, users with continuously rising expenditures and low category diversity are flagged as "at risk," with personalized suggestions generated accordingly.

This runtime integration reflects our commitment to blending GenAI capabilities with robust user-centered control, ensuring AI enhances experience without reducing transparency or usability.

Reference

[1] Tian, H. (2020). Competitive strategy research on SUISHOU Technology's personal finance management app (Master's thesis, Guangxi Normal University). (in Chinese)

[2] Tao, J. (2022). Product design research of intelligent bookkeeping robot based on KANO model. Design. (in Chinese)

[3] Ma, H., & Liu, Q. (2018). Design and implementation of personal finance management system. Information and Computer, 30(6), 50–52. (in Chinese)

[4] Muzammil0777. (2023). Personal-Finance-Tracker [GitHub repository]. https://github.com/Muzammil0777/Personal-Finance-Tracker

To here - No more than 18 pages including tables, charts, figures and diagrams you may have.

Individual contribution and reflection

Each group member should provide one paragraph outlining their key contribution to the project (this must be agreed by the group) and another paragraph reflecting on their learning experience (this should include thoughts on the overall process, challenges encountered, and skills developed). Maximum 300 words for each member. Copy the template below.

QM no: 221169368

Name: Haoran Sun

Main Contribution:

As the team leader, I was responsible for overall project planning, coordination, and key feature development. I played a major role in designing the system's modular architecture, establishing our **use of the MVC** pattern, and **maintaining the main GitHub repository** with **branch management** and **code review**. During the requirements stage, I led **background research** on existing finance applications, **conducted stakeholder** interviews, and analyzed user needs by results of reading and questionaires, **shaping our initial product backlog and scope**. I also took charge of **writing the Project Management and Requirements sections of the report**, **generated Javadoc** documentation, and **refined the final report** for clarity and completeness. In implementation, I developed several core modules: **Membership Expiry Reminder** (Feature 14), **Holiday Reminder** (Feature 20), **Cash Flow Visualization** (Feature 15), and **Main Menu Navigation** (Feature 24), which provides a centralized and intuitive user interface. Additionally, I integrated JUnit-based unit testing, wrote test cases, and **assisted teammates in debugging and cross-module optimization**.

Reflective Statement:

This project gave me practical experience in agile software engineering and team leadership. I learned to balance technical work with management—coordinating sprints, organizing meetings, and supporting the team. **Key challenges** included designing UI style, aligning coding styles and resolving merge conflicts. Early ambiguity in requirements led to rework, highlighting the need for communication and ongoing validation. By implementing financial features and reminders, I **improved my JavaFX and MVC skills**. Integrating JUnit tests deepened my understanding of test-driven development. Working with teammates also strengthened my code review and documentation abilities. Overall, this experience showed me the importance of adaptability, clear communication, and feedback in agile projects.

QM no: 221169379

Name: Yudian Wang

Main Contribution:

In this group project, my contributions mainly include: designing high-fidelity prototypes with Instant Design and defining interaction logic for core pages, sorting out user requirements to complete functional architecture design and specification documentation. In development, I focused on designing VIP module's benefit rules and activation processes, developing information forms and three-element verification functions for the bank card entry module with added security verification links, and implementing multi-type visual chart modules based on JavaFX that support interactive operations and data analysis. Additionally, I designed unit testing plans, wrote test cases, built an automated testing environment, developed test matrices and flowcharts, and assisted in fixing code defects, providing systematic support from prototype design to module development and testing assurance for the successful completion of the group project.

Reflective Statement:

During the requirements grooming process, I failed to fully consider the bank interface rules, which led to code modifications that affected the project timeline. The initial unit testing didn’t cover all edge cases, indicating that the testing plan lacked rigor. When developing multiple modules simultaneously, my time allocation was unreasonable, as I spent excessive hours on chart optimization.

To address these issues, I will: conduct requirements analysis with more scenario-based considerations and prepare in advance for potential changes; learn professional testing methodologies to improve test coverage; apply time management techniques to allocate work reasonably and avoid task overtime. This experience has taught me that project execution requires both global vision and attention to details, and I will implement these reflections in future practices.

QM no: 221169391

Name: Zhengxuan Han

Main contribution:

In this group project, my main contributions focused on interface navigation, layout optimization, user security enhancement, and the implementation of intelligent financial analysis. I optimized the **VIP Center interface** by clearly presenting all VIP-exclusive benefits, helping non-members make informed decisions. I also improved the layout of the login, registration, and user information editing interfaces, enhancing the user experience. I designed the **Security Verification Module**, introducing multi-factor authentication during login, and applied SHA-256 encryption to store user passwords during registration. In the **Budget Compare Module**, I integrated an AI model to analyze users' historical transaction data and generate insightful reports. Lastly, I implemented the **Main Menu Navigation**, which consolidates major functional modules and improves user interaction. All these features were thoroughly tested using JUnit to ensure system stability and accuracy across different user scenarios.

Reflective Statement:

During the project, I spent excessive time on interface aesthetics, which limited the validation of the AI analysis module and impacted result accuracy. Additionally, the initial design of the navigation interface lacked consideration of user habits, leading to suboptimal usability in early testing. In future projects, I will balance design and development efforts more effectively and prioritize core functionality. I also plan to involve user feedback earlier in the design phase to ensure that interfaces are both visually appealing and user-friendly. These experiences have highlighted the importance of time management and user-centered thinking in software development.

QM no: 221169483

Name: Muchi Wei

Main contribution:

My key contribution to the project was focusing on the user login and AI - function components. Regarding the login interface, my main contribution is designing a login interface that is still connected to the backend and data (CSV files), and allows users to modify their personal information after logging in. Regarding the AI functions, I called the interface of the qwen2.5: 72b model and successfully implemented sending our users' transaction records to the large model with a fixed prompt and received feedback: health score, consumption forecast, and consumption suggestions.These features can greatly enhance the user's experience.In addition, during the first phase, I also participated in the story writing and was responsible for my part. In the final section, I was also in charge of the corresponding user manual and report.

Reflective Statement:

Through this group collaboration, the most important thing I learned is teamwork; breaking down a challenging task among everyone makes it more efficient. Secondly, it also reflected some issues, such as being unfamiliar with using GitHub and the programming content not being modular enough. Of course, this experience also enhanced my familiarity with interface design and taught me how to call APIs instead of relying on local deployment.

QM no: 221169357

Name: Wei Chen

Main contribution:

In this project, my main contributions centered around the development and enhancement of the transaction system, data interaction and interface design. I designed and implemented the Transaction class, which enables seamless interaction with CSV files, ensuring efficient data storage and retrieval. Additionally, I created the Transactions Table Page, which visually presents transaction data in an intuitive and user-friendly format. I developed methods for adding, modifying, and deleting transaction records, providing users with full control over their

data. Furthermore, I designed several UI interfaces, optimizing the overall user experience and interaction flow. All these features were rigorously tested to ensure reliability and accuracy in various scenarios.

Reflective Statement:

Through this project, I gained valuable insights into both technical and collaborative aspects of software development. Working on the transaction system was particularly rewarding, as it allowed me to apply my knowledge of Java and data structures to create a functional and user-friendly module. Interfacing with CSV files and developing methods for data manipulation honed my problem-solving skills, especially when dealing with data integrity and performance optimization. Designing the UI interfaces was a creative challenge, requiring me to balance aesthetics with usability. The process of rigorous testing taught me the importance of attention to detail and the iterative nature of development. Collaborating with my team members was an enriching experience, as we shared ideas, resolved conflicts, and leveraged each other's strengths. The challenges we faced, such as integrating different modules and managing time effectively, pushed me to improve my communication and adaptability. Overall, this project significantly enhanced my technical abilities and teamwork skills, and I am excited to apply these learnings to future endeavors.

QM no: 221169645

Name: Kaiyu Liu

Main contribution:

In this group project, I was mainly responsible for implementing the user search and smart cost-cutting suggestion features. I developed the User Search Module, namely, the admin interface, enabling Administrators to search by nickname and view detailed information such as email, gender, and date of birth. The Administrators can also search for VIP or Normal users by inputting "VIP" or "normal", and loads user data from a CSV file during initialization. Additionally, I implemented the Smart Suggestion Module, which utilizes an AI model to analyze user expense records from CSV files and provide personalized advice for reducing unnecessary spending. These features were rigorously tested using JUnit to ensure accurate data handling and reliable AI interaction.

Reflective Statement:

Reflecting on my learning experience, I realized that while I focused heavily on ensuring feature completeness and backend accuracy, I underestimated the importance of early usability testing and interface intuitiveness. Some initial feedback revealed that the admin interface, though functional, could have been more intuitive for first-time users. In future projects, I plan to adopt a more iterative design approach, integrating user testing earlier in the development cycle. I also aim to improve my ability to balance technical implementation with user experience design. This project has reinforced the value of thorough planning, user feedback, and maintaining a holistic view throughout the development process.