EBU6304 Software Engineering Group Project

2024-25

Group number	32	
member 1	QM no: 221165821	Name: Jialei Xu
member 2	QM no: 221165843	Name: Yiming Xia
member 3	QM no: 221166172	Name: Jingying Yue
member 4	QM no: 221166091	Name: Jiawei Ma
member 5	QM no: 221166161	Name: Zhe Dong
member 6	QM no: 221166079	Name: Xiaoyue Fu

Group report

1. The purpose and scope of the application

1.1 Purpose

The AI-Empowered Personal Finance Tracker is a desktop application designed to help users effectively manage their personal finances. In today's digital economy, where transactions frequently occur across multiple platforms such as WeChat Pay, Alipay, and traditional banks, many people struggle to track expenses, categorize spending, and set reasonable budgets. Our application integrates AI technology to help users easily track daily income and expenses, analyze spending habits, and provide personalized financial management advice, while maintaining users' ultimate control over financial decisions.

1.2 Scope

The application is a standalone Java desktop application that can function without an internet connection. It focuses on the following areas:

1. **Transaction Management System**: Providing flexible ways to record income and expenses, supporting custom categorization and batch importing
2. **AI Smart Analysis**: Automatically categorizing transactions and providing personalized financial advice
3. **Data Visualization and Analysis**: Visually displaying financial data and consumption trends through charts
4. **Localization Features**: Adapting to Chinese holiday consumption patterns and providing holiday spending planning

1.3 Users

The application primarily targets:

- Young professionals who need to manage personal finances
- Budget-conscious individuals looking to optimize spending
- Users who make payments across multiple platforms and need centralized financial management
- Users who want AI assistance but still need to maintain control over financial decisions
- Individuals looking to improve financial literacy and budgeting skills

1.4 Main Features

Core Features (Priority 1)

- **Transaction Record Management**: Support for income and expense transaction recording, with customizable categories and color coding
- **Batch Data Import**: Support for CSV file batch import of transaction data to improve accounting efficiency
- **AI Financial Recommendations**: Automatically generated personalized financial advice based on user spending habits and financial data
- **Record Filtering and Querying**: Support for quick filtering of records by transaction type, time, and other conditions

Enhanced Features (Priority 2)

- **Data Visualization**: Intuitive display of financial data and consumption trends through pie charts, bar charts, etc.
- **Multi-dimensional Statistical Analysis**: Providing spending trend analysis for different time dimensions such as monthly and quarterly
- **Holiday Spending Planning**: Automatically retrieving the latest holiday information and creating dedicated consumption budget plans
- **Report Export Function**: Supporting CSV and PDF format report export for sharing and archiving

Advanced Features (Priority 3)

- **Multi-Currency Support**: Supporting multiple international mainstream currencies and ensuring accurate conversion between different currencies
- **Smart Analysis Reports**: Selecting transaction records for in-depth analysis and generating detailed financial insight reports
- **User Feedback System**: Built-in search function to quickly find answers to common questions, supporting user submission of usage feedback

2. Project management

2.1 Agile Development Methodology

Scrum Framework:

- **Sprint Planning**: Divided the project into 2-week iterations (aligned with key submission dates). Each sprint focused on delivering prioritized features (e.g., core AI classification, GUI implementation).
- **Daily Standup Meetings**: Conducted 15-minute meetings to discuss progress, blockers, and next steps. Used tools like Discord for remote coordination.
- **Sprint Reviews and Retrospectives**: After each iteration, demonstrated working software to internal stakeholders (simulated "clients") and gathered feedback. Retrospectives identified process improvements (e.g., optimizing task allocation).

User Story Mapping:

- Created user stories (e.g., "As a user, I want to import CSV files to automate expense tracking") with acceptance criteria. Prioritized stories using the MoSCoW method (Must-have, Should-have, Could-have, Won't-have).
- Collaboratively refined the product backlog using tools like Trello to visualize dependencies and scope changes.

AI-Assisted Development:

- Used GitHub Copilot for code suggestions and ChatGPT for brainstorming UI/UX ideas. However, all AI-generated code/logic was manually validated (e.g., ensuring AI-classified expenses matched regional contexts like "WeChat Red Packets").

2.2 Layered Iterative Development Strategy

Based on the project timeline, we designed progressive development across 4 major software versions:

Version Evolution Planning:

- **v1.0 (Week 5)**: Core functionality prototype to validate basic concepts
- **v2.0 (Weeks 7-8)**: Extended feature implementation with user feedback integration
- **v3.0 (Week 9)**: System optimization and AI functionality refinement
- **v4.0 (Weeks 12-13)**: Final product delivery with user experience optimization

2.3 Version Control and Branch Management Strategy

Implemented Git branching strategy to ensure controllability and traceability of code changes:

Branch Management Model:

- **Master Branch**: Stable production code
- **Feature Branches**: Individual team members responsible for specific feature development

- **Pull Requests**: Code review and quality control
- **Continuous Integration**: Ensuring all branches merge to master before checking dates

2.4 Requirements Management and Priority Adjustment

Dynamic requirements management through Product Backlog:

Requirements Adaptation Strategies:

- **User Story Driven**: Writing adaptable user stories based on user needs
- **Priority Matrix**: Dynamic prioritization based on implementation difficulty and requirement importance
- **Acceptance Criteria**: Clear completion standards for each user story
- **Iterative Adjustment**: Adjusting subsequent priorities based on progress in each sprint

2.5 Risk Management and Contingency Planning

Established multi-layered risk identification and response mechanisms:

Risk Response Strategies:

- **Technical Risks**: Reducing AI implementation complexity through prototype validation
- **Team Collaboration Risks**: Clear GitHub contribution requirements ensuring individual accountability
- **Time Management Risks**: Staged checkpoints ensuring controllable progress
- **Quality Risks**: TDD and code review processes

2.6 Communication and Coordination Mechanisms

Established effective team communication systems:

Communication Strategies:

- **Regular TA Checks**: Teaching assistants providing external supervision and feedback
- **GitHub Issues**: Centralized platform for technical discussions and issue tracking
- **Demo and Viva Sessions**: Regular client presentations to showcase progress and receive feedback
- **Team Collaboration Tools**: ReadMe documentation recording task allocation and updates

2.7 Quality Assurance and Testing Strategy

Ensuring software quality while maintaining development flexibility:

Quality Control Methods:

- **Test-Driven Development (TDD)**: Using JUnit for unit testing
- **Code Documentation**: JavaDocs ensuring code maintainability
- **User Manual**: Continuously updated user guide
- **Error Checking**: Basic input validation and exception handling

3. Requirements

3.1 Fact-finding techniques

To establish a robust understanding of user needs and software requirements, our team adopted a multi-method approach. We conducted **structured user interviews** with ten potential users from diverse financial backgrounds to gather qualitative insights into budgeting challenges and digital habits. These interviews provided first-hand accounts of difficulties in tracking spending, interpreting financial data, and setting realistic goals. In parallel, we distributed an **online questionnaire** that collected over 50 responses, yielding statistical feedback on users' preferred features, such as AI categorisation accuracy and personalized advice.

Additionally, we performed **competitive analysis** by reviewing widely used applications including Mint, YNAB, and Alipay budgeting modules. This analysis highlighted several gaps in adaptability to the local Chinese financial environment, such as the need to handle red envelopes, festival spikes, and offline payments. The insights were compiled and visualized in the form of user personas and feature priority lists. All findings were continuously fed into the evolving product backlog, ensuring a user-driven development process from the outset.

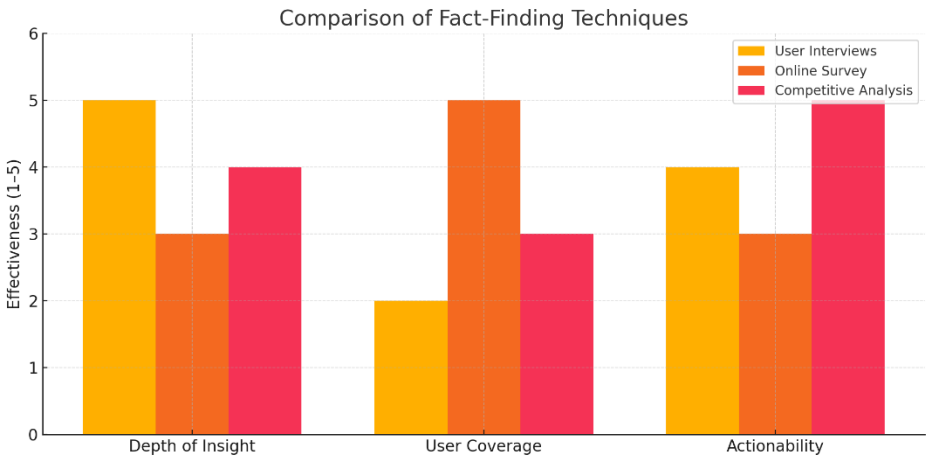

Figure 1. Comparison of Fact-Finding Techniques

This chart visualizes the relative effectiveness of different techniques based on depth of insight, user coverage, and actionability. Interviews offer the richest insight, while surveys cover a larger audience. Competitive analysis provides actionable features based on existing solutions.

3.2 Iteration plan

We adopted an **Agile iterative model** to structure our development process, consisting of four iterations mapped across eight project weeks. Each iteration followed a sprint cycle of: planning → implementation → testing → reflection. The **first iteration** focused on user interface exploration through low-fidelity paper prototypes and simple feedback forms. The **second iteration** delivered working modules for manual entry and visualisation of spending data. The **third iteration** integrated AI-based categorisation and budget predictions. Finally, the **fourth iteration** concentrated on fine-tuning and bug fixing, guided by user feedback from in-app surveys and test logs.

Throughout the process, stand-up meetings and retrospective reviews ensured fast adaptation and realignment with user expectations. The evolution of the project was tracked using a visual timeline, which also served to identify dependencies and overlapping tasks across the sprints.

Below is the visual Gantt chart illustrating the requirement and iteration planning:

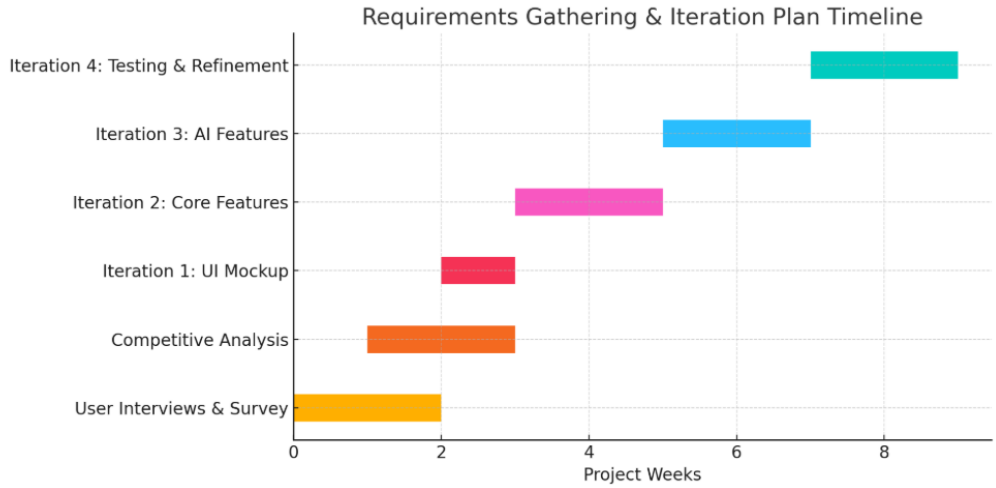

Figure 2. Requirements Gathering & Iteration Plan Timeline

3.3 Prioritisation

We adopted the MoSCoW method to categorize requirements into Must-have, Should-have, Could-have, and Won't-have.

For example, transaction logging, batch import, and AI categorization were marked as "Must-have", while data visualization and holiday budgeting were treated as "Should-have". Features like multi-currency support and feedback systems were classified as "Could-have".

This approach ensured that the team focused on the highest-value and most critical functionalities under limited resources.

3.4 Estimation methods

We used a combination of Story Points and Expert Judgment to estimate development workload.

Each user story was assigned a story point (e.g., 1, 2, 5, 8) based on complexity, effort, and testing requirements. These points were discussed collaboratively within the team. By comparing with past sprints, we estimated the achievable workload for upcoming iterations. For unfamiliar technical components, we used expert judgment from experienced members to define a realistic time range, improving estimation accuracy.

4. Analysis and Design

This section outlines the overall software architecture and internal structure, including system layering, class relationships, component roles, and design principles used to ensure extensibility and maintainability.

4.1 Overall Architecture Overview

The application follows a **layered architecture** to ensure separation of concerns and modularity. The layers include UI, business logic, utility tools, and data access. The figure below presents an abstracted overview of all key components and their hierarchical structure:

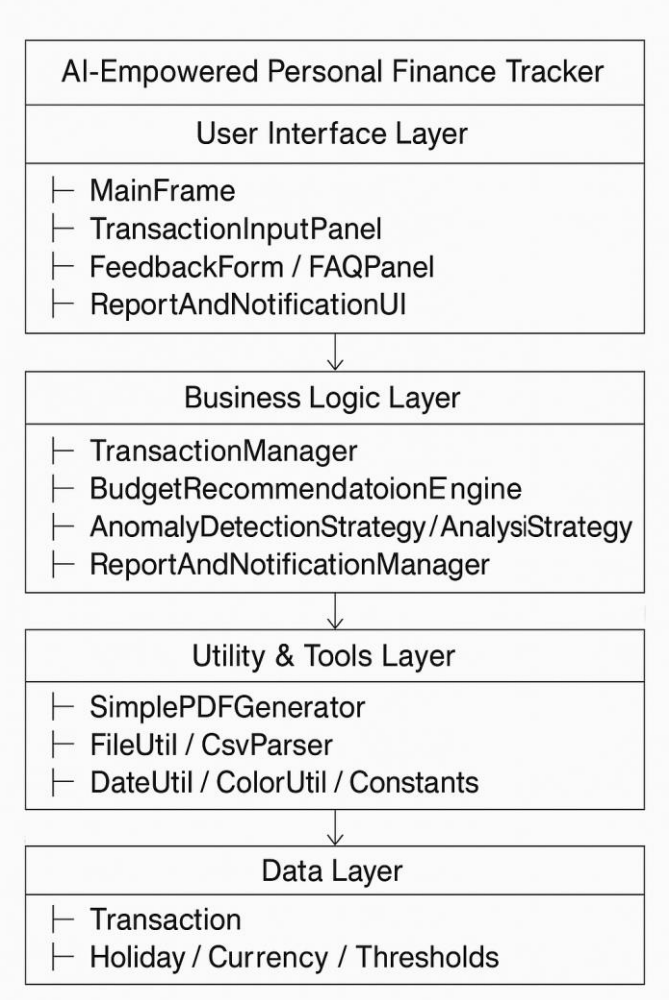

Figure 3. System Architecture

This architecture ensures that each layer communicates only with its adjacent ones, which improves scalability and testing.

4.2 Component-Level Design

Instead of listing individual methods or diagrams, the core responsibilities are summarized below:

Class	Responsibility Summary
TransactionManager	Manages transaction input, validation, and updates.
BudgetRecommendationEngine	Suggests budgets using trend analysis and category prioritization.
AnomalyDetectionStrategy	Detects outliers and spending spikes using statistical methods.
HolidayStrategy	Adjusts budgets and predictions based on local holiday patterns and seasonal behaviors.
ReportAndNotificationService	Generates and exports financial reports (PDF, CSV) and notifications.
SimplePDFGenerator	Handles PDF layout, formatting, and export actions.
IncomeAnalysisStrategy	Analyse income sources and stability and combine with AI models to provide personalised savings advice
IncomeVisualizationPanel	Provides a visual interface (bar charts, trend charts) for comparing income and expenses, with support for filtering by timeframe/category.
FeedbackAndIssueModule	Collects user feedback, reports issues, and provides FAQ search and rating UI via a dedicated Swing-based interface with offline file saving.
MainFrame & UI Panels	Provide user interaction interface and route user actions.

4.3 Key Components

To reduce diagram overhead, the following shows the core functional breakdown:

ReportExportComponent
- CSV Export Function
  - Data Formatting
  - Character Escaping
  - File Writing
- PDF Export Function
  - Layout Design
  - Content Formatting
  - PDF Rendering

BudgetAlertComponent

├── Expense Classification
├── Budget Limit Checking
└── Alert Message Generation

AIPersonalizationComponent

├── Spending Pattern Analysis
├── Budget Recommendation Strategy
│ ├── Monthly Trend Strategy
│ └── Category-Based Allocation
├── Holiday Strategy
│ ├── Festival Detection
│ ├── Budget Adjustment Logic
│ └── Local Behavior Adaptation
└── Income Recommendation Strategy
├── Categorisation of income sources (salaries, investments, bonuses, etc.)
├── Stability Assessment (volatility analysis based on historical data)
└── Savings Recommendation Generation (dynamic adjustment of budget allocation ratio)

ReportExportComponent

└── Visualisation Export Function
├── Chart export to PNG/PDF
└── Data table export to CSV

FeedbackAndIssueComponent

├── Feedback Form
├── Problem Form
├── FAQ Panel
└── File Util (Local File Persistence)

4.4 Design Principles Summary

Principle	Application
SRP (Single Responsibility)	UI, Service, AI, and Utility layers are functionally isolated.
OCP (Open-Closed)	Budget, Anomaly, and Holiday strategies are extensible without changing logic.
DIP (Dependency Inversion)	Business layer depends on abstract AnalysisStrategy and HolidayStrategy.
ISP (Interface Segregation)	Export, alert, AI, and UI modules are loosely coupled via dedicated interfaces.

5. Implementation

This section summarizes the project implementation strategy, phased development plan, and integration approach.

5.1 Development Strategy Overview

We followed an **incremental + TDD**-driven development cycle. Feature modules were divided by functional boundaries and progressively integrated.

5.2 Phased Development Plan

Phase	Focus	Key Deliverables
Phase 1	Data Model & Services	Transaction, ReportAndNotificationService, unit tests
Phase 2	Report Export & Budget Alerts	PDF & CSV export, budget threshold logic, BudgetAlertComponent
Phase 3	UI Development	MainFrame, InputPanel, Report UI, feedback/FAQ panel integration, IncomeInputPanel, IncomeVisualizationPanel
Phase 4	AI + Localization Modules	BudgetRecommendationEngine, AnalysisStrategy, HolidayStrategy, test datasets, IncomeAnalysisStrategy
Phase 5	Final Integration & Polishing	End-to-end testing, cultural adaptation, edge-case handling, final tuning

All modules were tested using JUnit and integrated using GitHub version control.

5.3 Integration & CI Strategy

- **Version Control**: Git feature branches, PR-based review, master-only merges
- **Build Tool**: Maven 3.6+ (Surefire, JaCoCo, PDFBox, Holiday JSON loader)
- **Testing**: TDD + boundary testing, coverage >85% on recommendation logic & export modules
- **CI Pipeline**:

```
mvn clean compile
mvn test
mvn sonar:sonar
mvn package
```

5.4 Error Handling & Validation

- Input fields validated for nulls, type errors, and illegal amounts
- Export files handled encoding + file name escaping
- Robust logging for transaction processing failures
- **HolidayStrategy** validates date formats and fallback if holiday data is missing

- Logging of rule activation (e.g., HolidayStrategy Activated: Chinese New Year)

5.5 Project Structure

```
com.finance.tracker/
├── model/          # Transaction, Category, Currency
├── service/        # AnalysisService, ReportService
├── strategy/       # AnalysisStrategy, BudgetStrategy, HolidayStrategy
├── ui/             # MainFrame, Panels, UserDialogs
├── util/           # FileUtil, PDFGenerator, DateUtil, HolidayUtil
├── data/           # holiday.json, locale_weights.json
└── test/           # JUnit test cases for all modules
```

6. Testing

6.1 Testing Strategy

6.1.1 Test Scope

- Unit Testing: Core business logic and model classes
- Integration Testing: Data persistence and external system integration
- Functional Testing: User interface and business processes

6.1.2 Test Levels

1. Model Layer Testing
    - TransactionTest: Testing basic transaction record functionality
    - CategoryTest: Testing category management functionality
2. Utility Class Testing
    - FileUtilTest: Testing file operations
    - CurrencyTest: Testing currency handling
3. Business Logic Testing
    - TransactionManagerTest: Testing transaction management
    - HolidayTest: Testing holiday management

6.2 Testing Techniques

6.2.1 Testing Frameworks

- JUnit 5: Primary testing framework
- Maven Surefire: Test execution and report generation

6.2.2 Testing Methods

1. Unit Testing Technique
    - ```
      @Test
      public void testTransactionCreation() {
          // Prepare test data
          Category category = new Category(1, "Food", CategoryType.EXPENSE,
      "icons/food.png");
          BigDecimal amount = new BigDecimal("100.50");
      ```
      ```

```
        // Execute the method under test
        Transaction transaction = new Transaction(category, amount, "Lunch");

        // Verify results
        assertNotNull(transaction.getId());
        assertEquals(category, transaction.getCategory());
        assertEquals(amount, transaction.getAmount());
    }
```

2. Boundary Condition Testing

```
    @Test
    public void testTransactionWithNullDescription() {
        // Test null value handling
        Transaction transaction = new Transaction(category, amount, null);
        assertNull(transaction.getDescription());
    }
```

3. Data Conversion Testing

```
    @Test
    public void testTransactionToCsvLine() {
        // Test CSV format conversion
        String csvLine = transaction.toCsvLine();
        assertTrue(csvLine.contains("2024-01-01 12:00:00"));
    }
```

6.3 Test-Driven Development (TDD) Application

6.3.1 TDD Practice Example

Using the Transaction class as an example to demonstrate TDD practice:

1. Write Test First

```
    @Test
    public void testTransactionSignedAmount() {
        // Define test case
        Category expenseCategory = new Category(1, "Food", CategoryType.EXPENSE);
        BigDecimal amount = new BigDecimal("100.00");

        // Expected result
        Transaction expenseTransaction = new Transaction(expenseCategory, amount, "Lunch");
        assertEquals(amount.negate(), expenseTransaction.getSignedAmount());
    }
```

2. Implement Functionality

```
    public BigDecimal getSignedAmount() {
        return category.getType() == CategoryType.EXPENSE ?
          amount.negate() : amount;
    }
```

3. Refactor and Optimize
    - Extract constants
    - Optimize code structure

- Add error handling

6.3.2 TDD Benefits

1. Clear Interface Design
    - Define clear class interfaces through test cases
    - Ensure method signatures meet usage requirements
2. Comprehensive Test Coverage
    - Normal scenario testing
    - Boundary condition testing
    - Exception handling testing
3. Improved Maintainability
    - Tests serve as documentation
    - Facilitates refactoring
    - Quick detection of regression issues

6.4 Test Results

6.4.1 Test Coverage

- Model Classes: High coverage (>80%)
- Utility Classes: Medium coverage (60-80%)
- Business Logic: Medium coverage (60-80%)

6.4.2 Test Quality

1. Test Completeness
    - Core functionality has corresponding tests
    - Boundary conditions are covered
    - Exception handling is verified
2. Test Maintainability
    - Clear test code structure
    - Standardized test case naming
    - Well-prepared test data

7. Future iterations

Based on the current version, we plan to further enhance the software's functionality and user experience in future iterations. The specific directions include:

1. Launch web and mobile versions to sync data across devices for better convenience.
2. Add voice input and AI assistant for spoken transactions and smart Q&A.
3. Enable receipt uploads and use OCR to auto-fill amount, date, and category.

These future iterations aim to enhance the system's intelligence, ease of use, and user engagement, better meeting the financial management needs of different users in real-life scenarios.

Based on the current version, we plan to further enhance the software's functionality and user experience in future iterations. The specific directions include:

1. Launch web and mobile versions to sync data across devices for better convenience.
2. Add voice input and AI assistant for spoken transactions and smart Q&A.
3. Enable receipt uploads and use OCR to auto-fill amount, date, and category.

These future iterations aim to enhance the system's intelligence, ease of use, and user engagement, better meeting the financial management needs of different users in real-life scenarios.

8. The use of Generative AI (GenAI)

During the development of our AI-Empowered Personal Finance Tracker, we incorporated several Generative AI tools—primarily **ChatGPT** and **GitHub Copilot**—across different stages of the software lifecycle, ranging from planning and prototyping to actual implementation and debugging.

At the **requirement analysis and design stage**, ChatGPT was used to help brainstorm user stories, generate UI/UX design alternatives, and structure our MoSCoW prioritisation more effectively. For example, it assisted in refining our idea of localized budget planning features that considered Chinese holidays, inspired by suggestions like "festival-aware recommendation strategies."

In the **implementation phase**, GitHub Copilot served as an assistive coding partner. It offered real-time suggestions while writing Java classes, especially during the development of the HolidayStrategy, BudgetRecommendationEngine, and feedback UI components. Copilot accelerated routine coding tasks (e.g., file IO, string parsing) and provided clean syntax scaffolding, which helped us maintain consistent structure across modules.

Moreover, **ChatGPT proved especially useful during AI logic development**, helping us design expense categorization strategies that could adapt to ambiguous transactions such as “红包 (Red Packets)” or festival-related spikes. It also provided ideas for validating edge cases, improving boundary testing (e.g., refund detection, three-fold spending spikes) within our JUnit-based TDD workflow.

However, while these tools significantly boosted our productivity and ideation speed, they had **notable limitations**:

- **Context-awareness** was often lacking—Copilot, for instance, sometimes suggested inaccurate logic due to misunderstanding the financial context.
- AI-generated code **still required rigorous human validation** to ensure semantic correctness, especially in culturally nuanced components like holiday-specific recommendations.
- Generative outputs were occasionally **verbose or redundant**, requiring manual simplification for performance and readability.

In conclusion, GenAI tools played a **supportive but non-autonomous role**, enhancing development efficiency and creativity, but still **relied heavily on our engineering judgment** to deliver a robust and locally-adapted finance solution.

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	Name	Main contribution	Reflective statement
221165821	Jialei Xu	1. Software function integration 2. High-fidelity prototype production 3. Low-fidelity prototype production for user interface and Q&A feedback 4. AI personalized recommendations and local financial environment design	Through this project, I strengthened my ability to integrate software modules and quickly adapt prototypes based on feedback. Working with AI recommendations pushed me to improve my technical depth and user-focused thinking. Overall, I gained hands-on experience in balancing technical design and user needs under real development constraints.
221165843	Yiming Xia	1. Low-fidelity prototype production for AI analysis 2. Data import and pre-process of transaction module 3. Maven format integration 4. write tdd junit test	This project honed my skills in Java development and Agile collaboration while building an AI finance system. Leading low-fidelity prototyping for transaction classification, I iteratively refined keyword logic for localized scenarios (e.g., Chinese New Year red packets) and GUI usability. Data preprocessing taught me to resolve CSV parsing pitfalls—like irregular descriptions breaking imports—and enforce validation rules (non-negative amounts, category consistency). Maven integration deepened my dependency management expertise, resolving JSON library conflicts through version analysis.

			Implementing TDD with JUnit exposed flaws in AI logic, such as refund misclassification, driving 85% test coverage via boundary cases (300% expense spikes).
221166079	Xiaoyue Fu	1. UI design for transaction input and export pages 2. Implementation of cost classification, classification change, and manual input interface 3. User Manual. pdf file, guiding users on how to use it	This work strengthened my UI/UX design skills while balancing functionality with usability. Creating the classification system taught me to design flexible structures supporting both AI suggestions and manual corrections. The user manual process helped me appreciate documentation as a core component of user experience. Through this project, I improved my ability to integrate software modules, adapt to feedback, and balance technical design with user needs under real development constraints.
221166091	Jiawei Ma	1. The interactive UI development of the expenditure forecast visualisation module, designing the expenditure interface UI. 2.Implementing the AI forecasting function, where the AI gives multifaceted suggestions for budgeting income and expenditure; 3. Implementing dynamic dimension switching (e.g. weekly/monthly/quarterly) across time zones when developing the time filtering component, to visualise income and expenditure percentages; 4. Writing structured README documents.	This project deepened my understanding of data visualisation performance optimisation and mastered complex state management solutions. Learned to balance technical accuracy with user experience in AI interaction design. The development of the time component exposed internationalisation pain points (mixed calculation of lunar and Gregorian calendars), prompting me to implement cross-region adaptation. The automated validation pipeline established in documentation writing made me realise the importance of maintainable documentation for open-source projects. The experience of collaborating with the back-end team to debug API data formats also

			strengthened my interface design skills in full-link development.
221166172	Jingying Yue	1.CSV/PDF financial report export functionality development 2.Budget overspend and expense alert UI design and implementation 3.Custom notification settings system construction 4.User financial data visualization and interaction optimization	Through this project, I enhanced my ability to handle complex data export and user interface design, learning how to rapidly iterate functional modules based on user feedback. Developing the notification alert system deepened my understanding of user experience design principles, helping me find the balance between technical implementation and user requirements. Overall, I gained valuable hands-on experience in combining technical solutions with user needs under real development constraints.
221166161	Zhe Dong	1.Low-fidelity prototype production for the Saving Plan UI 2.Design and implementation of the FeedbackAndIssueComponent 3.Development of feedback form, issue report form, and FAQ search panel	Through this project, I gained experience in translating user-centered requirements into functional UI components. Designing the saving plan prototype helped me explore user-friendly layouts, while developing the FeedbackAndIssueComponent improved my skills in Java Swing and modular interface design. I also learned to apply the MVC pattern effectively in structuring desktop UI elements.