ASSIGNMENT GROUP WORK

Qualification	**Pearson BTEC Level 5 Higher National Diploma in Computing**		
Unit number and title	**Unit 22: Application Development**		
Submission date	12/9/2025	**Date Received 1st submission**	
Re-submission Date		**Date Received 2nd submission**	
Group number:	**Student names & codes**	**Final scores**	**Signatures**
	PHU TUONG LONG – BD00772		LONG
	HO DUC DUONG – BD00535		DUONG

Class	SE07202	**Assessor name**	DO TRUNG ANH

Plagiarism

Student Declaration

P1	P2	P3	P4	P5	P6	M1	M2	M3	M4	M5	D1	D2

OBSERVATION RECORD

Student	**PHU TUONG LONG – BD00772**
Description of activity undertaken	

P1 Description of activity undertaken:
I produced a comprehensive problem definition statement for the "CampusExpense Manager" project. I identified the primary stakeholders (Students, Developers, Administrators) and their specific needs. I clearly defined the system's functional requirements, such as CRUD capabilities for expenses and budget setting, as well as non-functional requirements like offline functionality and cross-platform support. I also acknowledged potential management difficulties, including the strict 12-week timeline and the team's limited experience.

P3 Description of activity undertaken:

I conducted research into various software development tools and techniques suitable for the project. I analyzed different SDLC models, comparing the linear approach of Waterfall against the iterative nature of Agile. I researched development tools, specifically selecting Android Studio for its intelligent code editor and visual layout editor, and SQLite for its embedded local data storage capabilities. I also researched version control techniques using Git to manage collaborative development.

M2 Description of activity undertaken: I provided a critical justification for the selection of specific tools and methodologies based on the project scenarios. I justified the use of the Agile methodology to mitigate the risk of the strict 12-week deadline, allowing for iterative testing. I explicitly justified the choice of SQLite over cloud alternatives (like Firebase) to meet the mandatory "offline functionality" constraint without incurring costs. Furthermore, I justified the use of Android Studio and Java to align with the team's existing skill set and minimize the learning curve.

P4 Description of activity undertaken: I conducted a peer review of the proposed solution by distributing a survey that received 51 responses to gather quantitative data on user needs. I also conducted semi-structured personal interviews with four distinct personas (David, Emily, Joseph, Mary) to gather qualitative feedback regarding UI speed and feature requirements. I documented all feedback, including requests for features like "Dark Mode" and "Export to PDF".

P5 Description of activity undertaken: I developed the functional "BudgetWise Solutions" business application using Java and Android Studio. I created comprehensive support documentation, specifically a User Guide, which includes step-by-step instructions and visual screenshots for key features such as User Registration, Expense Tracking, Budget Management, and using the built-in Calculator.

M3 Description of activity undertaken: I critically interpreted the feedback received from the peer review to identify gaps in the initial design. I analyzed the survey results, noting that while 80.4% of users

understood the core problem, there were clear demands for improved accessibility and automation. I identified specific opportunities not previously considered, such as implementing "Dark Mode" to reduce visual discomfort and adding "Data Export" features for professional users, and summarized these in a feedback improvement table. **M4** Description of activity undertaken: I developed the functional application strictly adhering to the software design document, specifically utilizing the MVC (Model-View-Controller) architecture. I provided technical evidence of this implementation through code snippets of the ChartHelper class (Controller) and DatabaseHelper (Model). I also demonstrated the successful integration of the preferred third-party tool, MPAndroidChart, to render dynamic visualizations of expense data.
Assessment & grading criteria
How the activity meets the requirements of the criteria

Student signature:	**LONG**	**Date:**	**12-09-2025**
Assessor signature:		**Date:**	
Assessor name:			

❐ Summative Feedback: ❐ Resubmission Feedback:

Grade:	Assessor Signature:	Date:

Internal Verifier’s Comments:

Signature & Date:

TABLE OF CONTENT

LIST OF FIGURE 8

LIST OF TABLE 10

INTRODUCTION 11

CHAPTER 1: LO1 PRODUCE OF A SOFTWARE DESIGN DOCUMENT FOR A BUSINESS-RELATED PROBLEM BASED ON REQUIREMENTS 12

1. (P1) Produce a well-defined problem definition statement, supported by a set of user and system requirements for a business problem. 12

1.1. Identify stakeholders and what they want in this scenario. 13

1.2. Management (Potential Difficulties) 16

2. (P2) Review areas of risk related to the successful development of a proposed application. 17

3. (M1) Analyse a business- related problem using appropriate methods to produce a well-structured software design document. 19

3.1. Problem Analysis using UML Diagrams 19

3.2. Test Plan Construction 21

CHAPTER 2: LO2 RESEARCH DESIGN AND DEVELOPMENT TOOLS AND METHODOLOGIES FOR THE CREATION OF A BUSINESS APPLICATION 24

4. (P3) Research the use of software development tools and techniques for the development of a proposed application. 24

4.1. SDLC models 24

4.2. Development tools and methodology selected 25

5. (M2) Justify the software development tools and development methodology selected. 30

5.1. Software Development Methodology: Agile (Scrum) 30

5.2. Integrated Development Environment (IDE): Android Studio 31

5.3. Programming Language: Java 32

5.4. Database Management System: SQLite 33

5.5. Project Management Tool: Trello .... **Error! Bookmark not defined.**

5.6. Version Control: Git & GitHub 34

CHAPTER 3: LO3 PLAN AND PRODUCE A FUNCTIONAL BUSINESS APPLICATION WITH SUPPORT DOCUMENTATION 36

6. (P4) Conduct a peer review of the problem definition statement, proposed solution and development strategy, documenting any feedback given 36

6.1. Survey Question 36

6.2. Survey Answer 40

6.3. Personal Interview 47

7. (P5) Develop a functional business application with support documentation based on a specified business problem. 49

7.1. Application Overview 65

7.2. Technical Specifications 65

7.3. User Guide and Features 65

8. (M3) Interpret peer-review feedback and identify opportunities not previously considered. 50

9. (M4) Develop a functional business application based on a specific software design document, with supportive evidence of using the preferred tools, techniques and methodologies 55

9.1. Feature Selection 55

9.2. Architecture Explanation 55

9.3. Code Evidence 55

9.4. Functional Evidence 58

9.5. Critical Review & Justification 59

CHAPTER 4: LO4 EVALUATE THE PERFORMANCE OF A BUSINESS APPLICATION AGAINST ITS SOFTWARE DESIGN DOCUMENT AND INITIAL REQUIREMENTS. 75

9. (P6) Review the performance of the business application against the problem definition statement and initial requirements 75

9.4. Review against Functional Requirements 75

9.5. Review against Non-Functional Requirements 79

9.6. Summary of Requirement Completion Table 82

10. (M5) Critically review the design, development and testing stages of the application development process including risks. 84

LIST OF FIGURE

Figure 1: Survey Question 1 36
Figure 2: Survey Question 36
Figure 3: Survey Question 3 37
Figure 4: Survey Question 4 38
Figure 5: Survey Question 5 39
Figure 6: Survey Question 6 39
Figure 7: Survey Answer 1 40
Figure 8: Survey Answer 2 40
Figure 9: Survey Answer 3 41
Figure 10: Survey Answer 4 41
Figure 11: Survey Answer 5 42
Figure 12: Survey Answer 6 42
Figure 13: Survey Answer 7 43
Figure 14: Survey Answer 8 43
Figure 15: Survey Answer 9 44
Figure 16: Survey Answer 10 44
Figure 17: Survey Answer 11 45
Figure 18: Survey Answer 12 45
Figure 19: Survey Answer 13 46
Figure 20: Survey Answer 14 46
Figure 21: User Registration Screen 66
Figure 22: User Registration Screen 67
Figure 23: Dashboard Interface 68
Figure 24: Add Expense Screen 69
Figure 25: Budget Management 70
Figure 26: Financial Report Card 71
Figure 27: Notes and Calculator 71
Figure 28: In-App Calculator Utility 72

Figure 29: Settings and Profile 73
Figure 30: Survey Answer 5 50
Figure 31: Survey Answer 6 51
Figure 32: Survey Answer 7 52
Figure 33: Survey Answer 8 52
Figure 34: Requesting Data from the Model 56
Figure 35: loadChart 56
Figure 36: Processing Logic for Visualization 57
Figure 37: View Initialization and Event Listening 58
Figure 38: Android Emulator (Pixel 9, API 36) 58
Figure 39: Secure User Registration and Authentication 75
Figure 40: Expense Tracking (CRUD) 76
Figure 41: Budget Setting 77
Figure 42: Expense Overview and Visualization 78
Figure 43: Recurring Expenses 78
Figure 44: Budget Alerts 79
Figure 45: Offline Functionality 80
Figure 46: Feedback and Support 81
Figure 47: Performance and User Interface 82

LIST OF TABLE

Table 1: Add Expense Entry 19
Table 2: Set Monthly Budget 20
Table 3: Summary of Selected Development Tools and Methodologies 35
Table 4: Personal survey 1 47
Table 5: Personal survey 2 47
Table 6: Personal survey 3 48
Table 7: Personal survey 4 48
Table 8: Summary of Feedback and Improvement Opportunities 53
Table 9: Summary of Requirement Completion Table 82

INTRODUCTION

The rapid digitization of daily life has created a significant need for tools that assist university students in managing their personal finances effectively. This report documents the end-to-end development of "BudgetWise Solutions" (initially conceptualized as "CampusExpense Manager"), a mobile application designed to help students track expenses, manage budgets, and analyze spending habits without relying on constant internet connectivity.

This assignment fulfills the requirements for Application Development by detailing the four key stages of the Software Development Life Cycle (SDLC):

- **Chapter 1 (LO1)** defines the business problem, identifying "Students" as the primary stakeholders and establishing critical constraints such as a strict 12-week timeline and the need for offline functionality.
- **Chapter 2 (LO2)** details the research and justification for selecting specific tools and methodologies, specifically the choice of Agile (Scrum) for management, Android Studio for development, and SQLite for local data storage.
- **Chapter 3 (LO3)** presents the execution of the project, including peer reviews, the application of the MVC (Model-View-Controller) architecture, and the production of a functional application with support documentation.
- **Chapter 4 (LO4)** evaluates the final application against its initial requirements and critically reviews the development process, highlighting risk management strategies and lessons learned.

The primary objective of this project was to deliver a "Model-View-Controller" (MVC) that ensures data privacy, cross-platform potential, and financial awareness for its users, while navigating the limitations of a junior development team.

CHAPTER 1: LO1 PRODUCE OF A SOFTWARE DESIGN DOCUMENT FOR A BUSINESS-RELATED PROBLEM BASED ON REQUIREMENTS

1. (P1) Produce a well-defined problem definition statement, supported by a set of user and system requirements for a business problem.

1.1. Management Difficulties for BudgetWise Solutions

When initiating the "CampusExpense Manager" (later BudgetWise Solutions) project, the development team faces several significant management hurdles that could impact the successful delivery of the application. These difficulties stem primarily from resource limitations and the specific nature of the team.

- Limited Development Experience & Strict Timeline: The team consists of "junior developers" who may struggle with complex features such as secure authentication and cross-platform deployment. This lack of experience significantly increases the risk of technical debt. Furthermore, the project is constrained by a "strict 12-week timeline," which creates immense pressure for rapid development and limits the time available for thorough testing and refinement.
- Budget Constraints: The project operates with "limited funds," which restricts the ability to purchase advanced tools, hire external consultants, or conduct extensive marketing. This forces the team to rely entirely on open-source and free tools, potentially affecting the "completeness" of the final product.
- Data Security & Compliance: A major management difficulty is ensuring data privacy. The team must implement "encrypted storage and secure authentication" to comply with regulations. However, the junior team lacks specialized knowledge in security implementation, making this a complex and risky task.
- Cross-Platform Compatibility Risks: Developing for multiple platforms (Android and iOS) simultaneously with a limited skillset creates a high risk of platform-specific bugs and "inconsistent user experience".

1.2. Issues to be Addressed in the Scenario

The core business problem centers on the financial management struggles of university students. The digitization of daily life has created a need for digital tools, yet students face specific constraints that current solutions often fail to address. The following specific issues must be resolved:

- Financial Awareness and Management: University students need an intuitive way to "track expenses," "manage budgets," and "analyze spending habits". Without a dedicated tool, they struggle to monitor limited funds across categories like tuition, rent, and food.

- Lack of Internet Connectivity: A critical issue identified is "inconsistent internet access" in locations such as lecture halls or basements. Existing cloud-based solutions fail in these environments. Therefore, a mandatory requirement is that the system "must work offline" and store data locally.
- Data Privacy: Students are concerned about the safety of their financial data. The solution must ensure "secure user registration" and authentication to protect sensitive information.
- Usability and Speed: User research indicates that students need to log costs quickly (e.g., "in under 5 seconds") without navigating through boring lists. The current lack of "Dark Mode" also causes visual discomfort for users at night, which is a key usability issue to address.

1.3. Consideration of Business Application Solutions

To solve the problems outlined above, different types of business applications were considered. The nature of the user's lifestyle and the technical constraints dictated the final choice.

- Desktop Application: A desktop solution would offer powerful processing for generating detailed reports and analyzing large datasets, which would benefit users like the "Department Manager" or "Senior Accountant" who require printable PDF reports or Excel exports. However, a desktop app fails to address the primary "on-the-go" needs of the student demographic. Students need to track expenses immediately at the point of purchase (e.g., buying pizza or coffee). A desktop application lacks the portability required for this real-time data entry.
- Mobile Application (Selected Solution): A mobile application was selected as the optimal solution for BudgetWise Solutions.
  - Portability: It aligns with the lifestyle of university students, allowing for "quick expense entry" anytime and anywhere.
  - Offline Capability: A native mobile app using an embedded database (SQLite) can easily fulfill the requirement to "function effectively without an internet connection," ensuring data availability even in unconnected areas.
  - Hardware Integration: Mobile apps can eventually leverage device features (like cameras for receipt scanning), which addresses the user desire for automation.

1.4. Identify stakeholders and what they want in this scenario.

The CampusExpense Manager app aims to provide university students with an easy-to-use mobile solution for managing their expenses and budgets. The core objectives are secure user registration, expense tracking with categorization, budget setting and adjustment, visualization of expense summaries and trends, support for recurring expenses, and timely budget notifications. The app must work offline, be compatible with both Android and ensure data privacy and security. The development team is small, relatively inexperienced, and constrained to a strict 12-week timeline and limited budget.

Figure 1: Budget

Features (Functional Requirements)

- Secure user registration and authentication mechanism to protect user accounts.
- Expense tracking with create, read, update, delete (CRUD) capabilities on expense entries.
- Expense categorization (e.g., rent, shopping, dining).
- Budget setting and dynamic adjustment of expense limits based on user input.
- Recurring expenses management with automatic addition to monthly budgets.
- Real-time summaries and visual reports on expenses and budgets by category and time period.
- Budget alert notifications based on pre-set thresholds.
- Feedback mechanism to report issues and suggest improvements.

Non-Features (Non-Functional Requirements)

- Offline functionality with local data storage and processing, with synchronization when connectivity is restored.
- Cross-platform support for both Android and iOS, ensuring a consistent user experience.
- Strong data encryption for sensitive information both in transit and at rest, complying with data privacy regulations.
- Application performance optimized to stay responsive even with large volumes of data.
- Maintainability and extensibility for future enhancements, such as monetization features.
- Stakeholders and Their Needs

Students (Primary Users)

Intuitive and simple interface for quick expense entry. Ability to track and categorize all expenses accurately. Tools to set, adjust, and monitor budgets. Notifications when nearing or exceeding budgets. Access to detailed reports and trends to understand spending habits. Offline functionality due to inconsistent internet access. Data privacy and security to protect financial information. BudgetWise Solutions Team (Developers) Clear functional and non-functional requirements to guide development. Manageable project scope within limited time (12 weeks) and budget.Training and skill development opportunities to improve mobile app development expertise. Access to tools and frameworks that simplify cross-platform development. Ability to maintain and update the app post-launch.

Figure 2: Students (Primary Users)

Assurance that the app promotes responsible financial habits among students. Possible integration or collaboration opportunities. Data privacy compliance to meet institutional policies and regulations.

System Requirements

The system must provide secure user registration and authentication mechanisms to protect user accounts. The system should allow CRUD (create, read, update, delete) operations on expense entries with appropriate data validation. The system must support budget setting and dynamically adjust expense limits based on user input. The app should generate real-time summaries and visual reports on expenses and budgets by category and time period. The system must handle recurring expenses by automatically adding them to monthly budgets during their active periods. The system should trigger budget alert notifications based on pre-set thresholds. The app must store and process data locally to ensure offline functionality, with synchronization capabilities when connectivity is restored. The system must support deployment and operation on both Android and iOS platforms, ensuring consistent user experience. The system must implement encryption for sensitive data both in transit and at rest to comply with data privacy regulations. The app should include a feedback mechanism for users to report issues and suggest improvements. Performance should be optimized to maintain responsiveness even with large volumes of data.

The system should be maintainable and extensible for future enhancements, such as optional monetization features.

1.5. Management (Potential Difficulties)

Limited Development Experience & Timeline

Junior developers may face challenges with complex mobile app features, cross-platform deployment, and secure authentication, increasing risk of delays or technical debt.

The 12-week deadline pressures rapid development, limiting time for thorough testing, refinement, and user feedback incorporation.

Budget Constraints & Data Security

Limited funds restrict use of advanced tools, external consultants, or extensive marketing, which might affect feature completeness and app reach.

Ensuring encrypted storage and secure authentication in compliance with regulations adds complexity, possibly requiring specialized knowledge or external resources.

Cross-platform Compatibility

Developing and testing for both Android and iOS within limited time and skillset may lead to platform-specific bugs or inconsistent user experience.

User Interface and Experience

Designing a user-friendly interface suitable for diverse student needs and device types may require UX expertise.

The outcome

The app aims to help university students manage expenses and budgets simply and securely, supporting features such as secure user registration, expense tracking with categorization, budget setting and dynamic adjustment, visualization of expense summaries and trends, support for recurring expenses, and budget alert notifications.

It must work offline, be compatible with Android and ensure data privacy and security.

Stakeholder needs are clearly identified: Students want ease of use and accurate budgeting tools; developers want clear requirements and manageable scope; administrators want financial habit promotion and data privacy compliance.

System requirements cover secure authentication, CRUD operations on expenses, real-time reporting, recurring expense handling, offline storage with sync when online, cross-platform support, encryption, feedback mechanisms, performance, and maintainability.

Potential management difficulties are acknowledged, including limited developer experience, tight 12-week timeline, limited budget, data security complexities, and cross-platform development challenges.

2. (P2) Review areas of risk related to the successful development of a proposed application.

Below is an analysis of the potential risks facing the "CampusExpense Manager" project and the corresponding management strategies

Technical Risks

Risk Identification

- Limited Team Experience: The development team consists of junior developers who may face challenges with complex mobile app features and secure authentication, increasing the risk of technical debt

- Platform Compatibility: Developing and testing for mobile platforms with a limited skillset may lead to platform-specific bugs or inconsistent user experiences

Management Strategy

- Utilize Android Studio: Use Android Studio's intelligent code editor and smart code completion to assist junior developers in writing correct code and detecting errors in real-time

- Simulation Testing: Leverage the built-in Android Emulator to test the application on multiple device configurations without needing physical hardware, ensuring compatibility

Simplify Data Management: Use SQLite (embedded database) to manage local data, as it requires minimal configuration and integrates directly with Android, reducing technical complexity

Security and Legal Risks

Risk Identification

- Data Compliance: The project requires ensuring encrypted storage and secure authentication to comply with data privacy regulations. Failure to do so could compromise student financial data

- Implementation Complexity: Implementing these security measures adds complexity that may require specialized knowledge the team lacks

Management Strategy

- Secure Storage: Utilize SQLite, which supports transactions and data integrity, to manage expense data securely and efficiently on the local device

- Standardized Tools: Rely on the official Android development environment (Android Studio) to implement standard security practices rather than building custom solutions from scratch

Budget Risks

Risk Identification: Resource constraints the project has limited funds, restricting the use of advanced paid tools, external consultants, or extensive marketing

Management Strategy

- Open Source and Free Tools: Select robust, free tools to keep costs at zero. Android Studio is the official IDE provided by Google , and SQLite is an embedded database that requires no separate server process or licensing

- Internal Skill Development: Instead of hiring consultants, the team will use the project as a training opportunity to improve mobile app development expertise

Schedule Risks

Risk Identification

- Strict Deadline: The 12-week deadline creates pressure for rapid development, potentially limiting time for thorough testing and refinement

- Waterfall Inflexibility: Using a linear model like Waterfall could be problematic if requirements change, causing delays

Management Strategy

- Adopt Agile Methodology: Use Agile to allow for iterative development and incremental delivery. This ensures core functionalities are built rapidly and allows the team to adapt to changes without missing the final deadline

- Efficient Collaboration: Implement Git for version control. This allows multiple developers to work concurrently (branching and merging) without conflicts, significantly speeding up the workflow

Product Quality Risks

Risk Identification

- UI/UX Issues: Designing a user-friendly interface for diverse student needs requires UX expertise, and a lack thereof may result in a poor user experience

- Bugs and Errors: Rapid development increases the risk of bugs in the final product

Management Strategy

- Visual Design Tools: Use Android Studio's Visual Layout Editor to drag and drop UI components. This simplifies the design process and ensures the interface is responsive across different devices

- Continuous Feedback: The Agile model emphasizes constant collaboration and user feedback, allowing the team to detect issues early and progressively enhance the app quality throughout the 12 weeks

- Code Integrity: Use Git to track changes and maintain codebase integrity, ensuring that if a new feature breaks the app, the team can easily revert to a previous stable version

3. (M1) Analyse a business-related problem using appropriate methods to produce a well-structured software design document.

3.1. Problem Analysis using UML Diagrams

To address the requirements of the "CampusExpense Manager" app, which aims to provide students with an easy-to-use solution for managing expenses, the following UML diagrams analyze the system structure and interactions

A. Use Case Diagram Analysi

Rationale: The system must focus on the "Students" as the primary users. The diagram illustrates the interaction between the Student actor and the core system functionalities defined in the requirements

Primary Actor

- Student the main user who tracks expenses and manages budgets

Core Use Cases

- Register/Login: Secure authentication to protect user accounts
- Manage Expenses (CRUD): Create, Read, Update, and Delete expense entries
- Set/Adjust Budget: Define monthly limits and receive alerts
- View Reports: Visualize expense summaries and trends
- Manage Recurring Expenses: Automatically add recurring costs like rent or subscriptions
- Offline Access: Store and process data locally without internet

B. Use Case Specifications

Detailed specifications for the two most critical functions based on the System Requirements

Specification 1: Add Expense Entry

Focus: CRUD operations and data validation

Table 1: Add Expense Entry

Item	Description
Use Case Name	Add Expense Entry
Actor	Student
Description	The user inputs a new expense transaction into the system with detailed categorization
Preconditions	The user is logged in

Main Flow	1. User selects "Add Expense" 2. System displays input form (Amount, Category, Date, Note) 3. User enters details and taps "Save" 4. System validates data (e.g., Amount > 0) 5. System saves data to the local SQLite database 6. System checks current total against the defined budget. If the limit is exceeded, a notification is triggered 7. System confirms success and updates the balance
Alternative Flow	If the device is offline, data is stored locally and synchronized later

Specification 2: Set Monthly Budget

Focus: Budget setting and dynamic adjustments

Table 2: Set Monthly Budget

Item	Description
Use Case Name	Set Monthly Budget
Actor	Student
Description	The user sets a spending limit for a specific category or the entire month
Preconditions	The user is logged in
Main Flow	1. User navigates to "Budget Settings" 2. User selects a category (e.g., Food) or "Total Budget" 3. User inputs the limit amount 4. System saves the configuration to the database 5. System recalculates the remaining balance and updates the visual progress bar
Alternative Flow	If the device is offline, data is stored locally and synchronized later

C. Class Diagram

The class structure reflects the business entities required to fulfill the objectives of expense tracking and budgeting

Key Classes

User: Manages authentication and profile data

- *Attributes:* userID, username, passwordHash

- *Methods:* register(), login()

Expense**:** Represents individual spending entries

- *Attributes:* expenseID, amount, date, note, categoryID

- *Methods:* createEntry(), editEntry(), deleteEntry()

Budget**:** Handles spending limits

- *Attributes:* budgetID, limitAmount, startDate, endDate

- *Methods:* setLimit(), checkThreshold()

Category**:** Categorizes expenses (e.g., Food, Transport)

- *Attributes:* categoryID, name, icon

D. Entity Relationship Diagram (ERD)

This data model is designed for SQLite, selected for its efficiency in handling local storage for offline functionality

Entity: Users

- user_id (PK, Integer), email (Text), password_hash (Text)

Entity: Categories

- category_id (PK, Integer), name (Text), type (Text)

Entity: Expenses

- expense_id (PK, Integer), user_id (FK), category_id (FK), amount (Real), date (Text), note (Text)

Entity: Budgets

- budget_id (PK, Integer), user_id (FK), category_id (FK), amount_limit (Real), month (Integer)

3.2. Test Plan Construction

Given the constraints of a small, inexperienced team and a strict 12-week timeline, the test plan prioritizes core functionality and early bug detection

Project Name: CampusExpense Manager Testing Tools: Android Studio (JUnit), Android Emulator

Test Objectives

Verify that secure user registration and offline functionality work as required

Ensure the application is stable and responsive across different Android device configurations

Validate that budget alerts trigger correctly when thresholds are met.

Roles and Responsibilities

Junior Developers: Responsible for writing Unit Tests for their own code logic (e.g., calculation algorithms) using the Agile iterative approach.

Testers (Peer Review): Responsible for Integration Testing and executing manual tests on the Android Emulator

Environment Requirements

Hardware: Development laptops

Software

- IDE: Android Studio (Official IDE)

- Database: SQLite (Embedded)

- Simulator: Android Emulator configured for API levels 29, 30, and 31

Test Strategy

Unit Testing

- *Scope:* Test individual classes like BudgetManager to ensure calculations are accurate

- *Tool:* JUnit (integrated in Android Studio)

Integration Testing

- *Scope:* Verify data flow between the User Interface and the SQLite database. For example, creating an expense must decrease the remaining budget displayed on the dashboard

UI/Compatibility Testing

- *Scope:* Use the Android Emulator to test the responsive layout designed with the Visual Layout Editor

Risk Management in Testing

Risk: Junior developers may miss complex edge cases

- *Mitigation:* Focus testing on the "Happy Path" (standard usage) first to ensure the MVC(Model-View-Controller) works, then address edge cases

Risk: Inconsistent user experience across devices

- *Mitigation:* Mandatory testing on at least three different screen sizes via the Emulator

Testing Schedule Aligned with the 12-week Agile timeline

Weeks 4-6: Unit testing for Registration and Expense CRUD modules

Weeks 7-9: Integration testing for Budgeting and Reporting features

Weeks 10-11: System testing and UI refinement

Week 12: Final review and bug fixing before submission

CHAPTER 2: LO2 RESEARCH DESIGN AND DEVELOPMENT TOOLS AND METHODOLOGIES FOR THE CREATION OF A BUSINESS APPLICATION

4. (P3) Research the use of software development tools and techniques for the development of a proposed application.

4.1. SDLC models

Table 3: Software Development Life Cycle (SDLC) Models

Model	Description	Pros	Cons
Waterfall	A linear, sequential approach where each phase (Requirements, Design, Implementation) must be completed before the next begins.	Simple and easy to manage. Clear milestones and deliverables. Good for small projects with fixed requirements.	Inflexible: Difficult to accommodate changes once development starts. Late Testing: Bugs are often found only at the end. High risk if requirements are misunderstood1.
Agile (Scrum)	An iterative approach that delivers software in small, incremental cycles called "Sprints."	Flexible: Adapts easily to changing requirements. Early Delivery: Users get a working MVC quickly. Continuous feedback loop improves quality.	Requires active user involvement. Can lead to "scope creep" without strict management. Requires a disciplined team to manage sprints effectively.
V-Model	An extension of Waterfall where each development phase has a corresponding testing phase (e.g., Design Integration Test).	Emphasizes testing and quality assurance at every stage. Clear verification process.	Rigid like Waterfall; difficult to handle dynamic changes. If requirements change, both development and testing documentation must be updated.

Software Development Life Cycle (SDLC) Models

Technologies for Comparison:

- Waterfall Model
- Agile Methodology (Scrum)

- V-Model

Selected Methodology: Agile Methodology (Scrum)

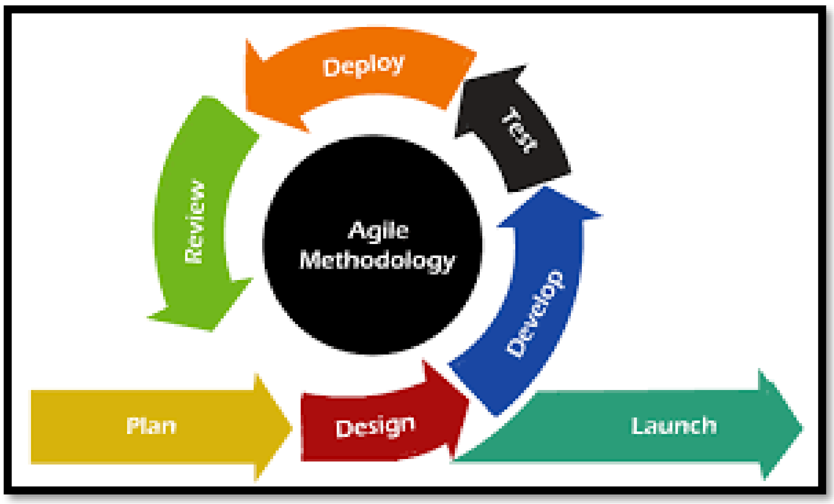


Figure 3: Agile Model

Definition: Agile is an iterative and incremental approach to project management and software development. Unlike linear models, Agile breaks the product into smaller pieces called "sprints" (typically 2-4 weeks long). It emphasizes flexibility, continuous collaboration between cross-functional teams, and rapid delivery of functional software. Scrum is a specific framework within Agile that utilizes fixed-length iterations and specific roles (such as Scrum Master and Product Owner) to structure the workflow.

4.2. Development tools and methodology selected

Integrated Development Environment (IDE)

Table 4: Integrated Development Environment (IDE)

Tool	Description	Pros	Cons
Android Studio	The official IDE for Android development, built by Google on IntelliJ IDEA.	It offers native support from Google, ensuring compatibility with the latest Android features. The visual layout editor allows for drag-and-drop UI design , and it includes powerful tools like a built-in emulator and Gradle build system[10].	The software is heavy and consumes significant RAM and system resources. It also tends to have a slower startup time compared to more lightweight code editors.
Visual Studio Code	A lightweight, open-source code editor developed by Microsoft with	It is extremely lightweight, fast, and responsive. It is also versatile, supporting many languages via extensions, and boasts a large community marketplace.	It lacks seamless integration for Android development, requiring manual setup for SDKs and emulators. Debugging Android apps is

	extensive plugin support.		generally less efficient here than in Android Studio.
Eclipse	An older, open-source IDE that was previously the standard for Android development.	It is well-known by legacy developers and has a massive ecosystem of existing plugins.	It is largely considered outdated and is no longer the official standard for Android. It suffers from slower build speeds and lacks the modern features found in Android Studio.

Technologies for Comparison:

- Android Studio
- Visual Studio Code
- Eclipse

Selected Tool: Android Studio

Definition: Android Studio is the official Integrated Development Environment (IDE) for Google's Android operating system, built on JetBrains' IntelliJ IDEA software. It provides a unified environment for building apps for all Android devices. Key features include a Gradle-based build system, a fast and feature-rich emulator, code templates, and GitHub integration. It also offers a visual layout editor that allows developers to create UI by dragging and dropping components.

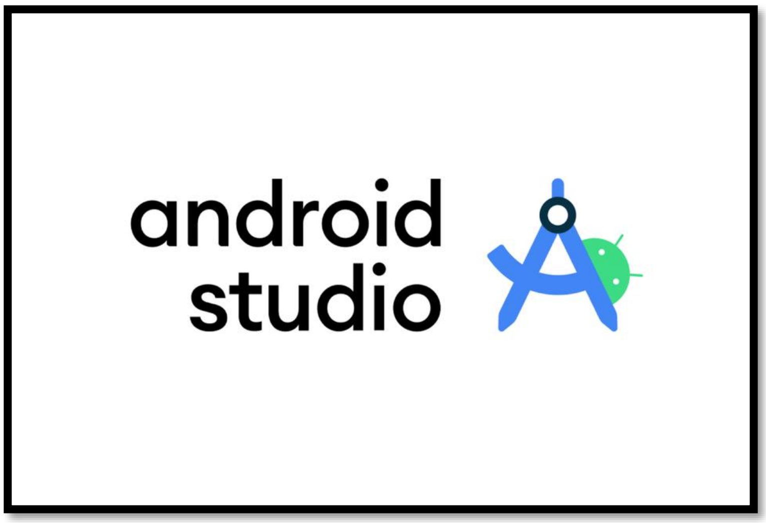


Figure 4: Android Studio

Database Management System (DBMS)

Technologies for Comparison:

- SQLite
- Firebase Realtime Database
- Realm

Selected Tool: SQLite

Definition: SQLite is a C-language library that implements a small, fast, self-contained, high-reliability, full-featured, SQL database engine. Unlike client-server database management systems, the SQLite engine has no standalone processes; it reads and writes directly to ordinary disk files. In the context of Android, it is an embedded database that comes built-in with the OS, allowing apps to store structured data locally on the user's device without requiring a network connection.

Table 5: Programming Language

Language	Description	Pros	Cons
Java	A mature, class-based, object-oriented language that has been the standard for Android for years.	The team is already familiar with it through the BTEC curriculum, eliminating the learning curve. It has a massive library of documentation and community support and is highly stable.	Java code can be verbose, requiring more lines of code to accomplish tasks compared to modern languages like Kotlin. Its evolution is also slower than newer alternatives.
Kotlin	A modern, statically typed language that runs on the JVM and is now Google's preferred language for Android.	It is concise, allowing developers to write less code to achieve the same functionality. It also features Null Safety, which drastically reduces common app crashes like NullPointerExceptions.	The primary downside is the learning curve. Adopting it would require the team to learn new syntax, which risks delays given the strict 12-week timeline.
Dart (Flutter)	A client-optimized language used with the Flutter framework for	It allows for cross-platform development, meaning one codebase works for both iOS and Android. Features like Hot Reload	It requires learning a completely new language (Dart), which is a high risk for a short project. It is also not

Language	Description	Pros	Cons
	cross-platform apps.	allow for instant viewing of code changes.	"Native," meaning it may lack direct access to some specific device hardware features.

Technologies for Comparison:

- Java
- Kotlin
- Dart (Flutter)

Selected Language: Java

Figure 5: Java

Definition: Java is a high-level, class-based, object-oriented programming language designed to have as few implementation dependencies as possible. It is a mature technology widely used for building enterprise-scale applications and Android mobile applications. Java applications are typically compiled to bytecode that can run on any Java Virtual Machine (JVM) regardless of the underlying computer architecture.

Table 6: Database Management System (DBMS)

Tool	Description	Pros	Cons
SQLite	A self-contained, serverless SQL database engine embedded directly into the app.	It is offline capable, storing data locally without needing an internet connection. It requires zero configuration or server setup and is free to use.	It has limited concurrency, allowing only one writer at a time. It is also not suitable for real-time synchronization across multiple devices.

Firebase	A cloud-hosted NoSQL database platform provided by Google.	It offers real-time synchronization, meaning data updates instantly across all devices. It is also easy to scale as the user base grows.	It is primarily designed for online use and relies on internet connectivity. It can become expensive if limits are exceeded, and storing data on servers abroad may raise compliance issues.
Realm	An object-oriented mobile database that is an alternative to SQLite.	It is generally faster than SQLite for complex queries and is easy to use with object-oriented code.	It increases the application size slightly and requires learning a specific database interaction method that differs from standard SQL.

Technologies for Comparison:

- SQLite
- Firebase Realtime Database
- Realm

Selected Tool: SQLite

Figure 6: SQLite

Definition: SQLite is a C-language library that implements a small, fast, self-contained, high-reliability, full-featured, SQL database engine. Unlike client-server database management systems, the SQLite engine has no standalone processes; it reads and writes directly to ordinary disk files. In the context of Android, it is an embedded database that comes built-in with the OS, allowing apps to store structured data locally on the user's device without requiring a network connection.

Version Control System

Table 7: Version Control System (VCS)

Tool	Description	Pros	Cons

Git	A distributed version control system that tracks changes in source code.	It is distributed, meaning every developer has a full copy of the history[19]. It offers excellent support for branching and parallel development [20] and is the industry standard.	It has a steep learning curve for command-line usage and resolving merge conflicts can be complex for beginners.
SVN (Subversion)	A centralized version control system.	It uses a single central repository, which can be simpler for beginners to understand. It handles binary files efficiently.	It has a single point of failure; if the central server goes down, no one can commit code. It is also slower for branching and merging compared to Git.
Manual Backup	Manually copying files to USBs or cloud storage like Google Drive.	It requires no special tools to learn and is simple for single-person projects.	It carries a high risk of version conflicts where one developer overwrites another's code[21]. There is no history tracking or ability to easily undo specific changes.

Technologies for Comparison:

- Git (via GitHub)
- Apache Subversion (SVN)
- Mercurial

Selected Tool: Git (via GitHub)

Definition: Git is a distributed version control system designed to handle everything from small to very large projects with speed and efficiency. It allows multiple developers to work on the same codebase simultaneously by tracking changes in source code during software development. GitHub is a cloud-based hosting service for Git repositories, providing a web-based graphical interface and access control/collaboration features such as bug tracking, feature requests, and task management.

5. (M2) Justify the software development tools and development methodology selected.

The selection of tools and methodologies for the "CampusExpense Manager" project was driven by the specific constraints of BudgetWise Solutions: a strict 12-week timeline, a team of junior developers, a limited budget, and a critical functional requirement for offline capability

5.1. Software Development Methodology: Agile (Scrum)

Selected: Agile Methodology

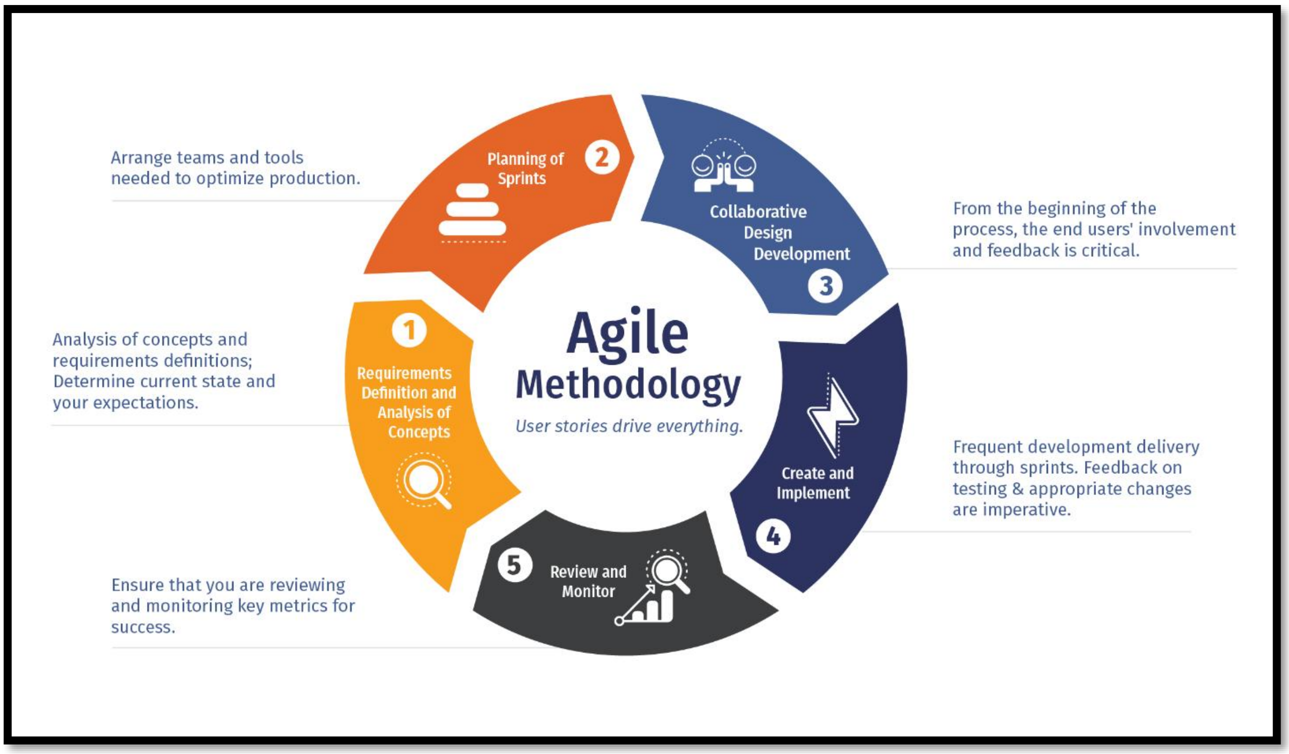

Arrange teams and tools needed to optimize production.
Planning of Sprints
2
Collaborative Design Development
3
From the beginning of the process, the end users' involvement and feedback is critical.
Analysis of concepts and requirements definitions; Determine current state and your expectations.
1
Requirements Definition and Analysis of Concepts
Agile Methodology
User stories drive everything.
Create and Implement
4
Frequent development delivery through sprints. Feedback on testing & appropriate changes are imperative.
5
Review and Monitor
Ensure that you are reviewing and monitoring key metrics for success.

Figure 7: Agile Model

The Constraint : The project operates under a "strict 12-week timeline" with a team of "junior developers" who risk "technical debt". The team cannot afford the "Waterfall" risk of finding critical bugs only in the final week.

The Research : Comparative research showed that Waterfall is "Inflexible" and prone to "Late Testing". In contrast, Agile uses "sprints" to ensure "Early Delivery" and a "Continuous feedback loop".

The Justification: We selected Agile (Scrum) to mitigate schedule risk. By breaking the project into 2-week sprints, the team could test the "Expense Tracking" feature in Week 4 rather than waiting until Week 12. This iterative approach allowed us to "descale the scope" (e.g., dropping the monetization feature) if the junior team struggled, ensuring the delivery of a working MVC by the deadline.

5.2. Integrated Development Environment (IDE): Android Studio

Selected: Android Studio

Alternative Considered: Eclipse or Visual Studio Code

Justification: While Visual Studio Code is lightweight, it lacks the deep integration required for native Android development.

Why Android Studio?

Native Support: It is the official IDE provided by Google, offering the most stable environment for the Android SDK.

Figure 8: Android Studio

The Constraint : The "junior developers" lack experience with complex code structures , yet the system requires "maintainability and extensibility". Without a clear structure, there is a high risk of creating "spaghetti code" that is impossible to debug within the short timeline.

The Justification: We selected MVC to solve the "Junior Developer" and "Collaboration" constraints. MVC enforces a strict "separation of concerns", which allowed our team to work in parallel without conflict:

- Model (Data): One developer focused on SQLite and the DatabaseHelper class to manage raw data.
- View (UI): Another developer used Android Studio's Layout Editor to build XML interfaces.
- Controller (Logic): A third developer wrote the Java logic (e.g., ChartHelper) to process data. This structure was essential for the team to work simultaneously on different files ("collaborative development") as facilitated by Git.

5.3. Programming Language: Java

Selected: Java Alternative

Considered: Kotlin or Flutter (Cross-platform)

Justification: Although Flutter allows for iOS and Android development simultaneously, it requires learning a new language (Dart), which poses a high risk for a team with a strict 12-week deadline.

Why Java?

Figure 9: Java

The Constraint: The team has "Limited Development Experience" and cannot afford the learning curve of a new language like Dart/Flutter given the 12-week limit.

The Research: Java is the "core language taught in the BTEC curriculum," eliminating the learning curve. Android Studio offers a "Visual Layout Editor" that allows for "drag-and-drop" UI design, unlike lightweight editors like VS Code.

The Justification:

- For the Controller (Java): We chose Java because the team was already familiar with it, allowing us to focus on implementing the complex MVC Controller logic (e.g., ChartHelper) immediately without learning new syntax.
- For the View (Android Studio): We selected Android Studio specifically for its visual tools. The "Visual Layout Editor" allowed junior developers to construct the View layer (XML layouts) visually, drastically reducing UI errors and development time compared to writing code from scratch.

5.4. Database Management System: SQLite

Selected: SQLite Alternative Considered: Firebase Realtime Database

Justification: Firebase is a powerful cloud-based database, but it relies primarily on internet connectivity.

Why SQLite?

Figure 10: SQLite

The Constraint: A "fundamental constraint" is that the app "must function effectively without an internet connection" due to students utilizing it in basements or lecture halls.

The Research: Research confirmed that Firebase "relies on internet connectivity" and can be "expensive". SQLite is "self-contained," "serverless," and "embedded directly into the app".

The Justification:

SQLite is the only valid choice for the Model layer because it directly meets the "Offline Functionality" requirement. It stores data locally on the device, ensuring reliability where cloud solutions would fail. It also fits the "limited budget" as it is free and open-source.

5.5. Version Control: Git & GitHub

Selected: Git Alternative Considered: Manual File Sharing (Google Drive/USB)

Justification: Manual sharing leads to "version conflicts" where one developer overwrites another's code.

Why Git?

Figure 11: GIT & GitHub

The Constraint: With multiple junior developers working on the same timeline, there is a high risk of "version conflicts" where one developer accidentally overwrites another's code. Additionally, inexperienced developers are prone to breaking working code, requiring a "safety net".

The Research: Research into Manual Backup (USB/Drive) showed it carries a "high risk of version conflicts". In contrast, Git is "distributed," allowing for "branching and parallel development".

The Justification (Connecting to MVC): Git is mandatory because we are using MVC. Since MVC splits a single feature into multiple files (e.g., Expense.java (Model), activity_add_expense.xml (View), AddExpenseActivity.java (Controller)), multiple developers must work on these different files simultaneously.

- Parallel Work: Git allows one developer to modify the Model while another modifies the View without file lockouts.
- Safety Net: If a junior developer breaks the app logic, Git allows us to "revert" to a previous working version instantly, protecting the 12-week deadline.

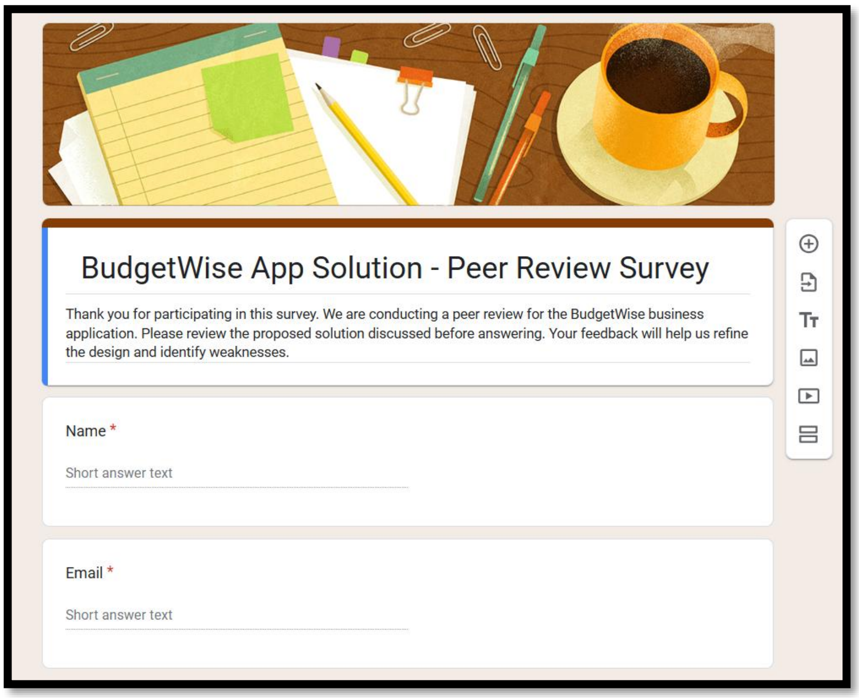
Table 8: Summary of Selected Development Tools and Methodologies

Component	Selected Tool/Pattern	Justification based on Constraints (P1) & Research (P3)
Methodology	Agile (Scrum)	Mitigates the strict 12-week timeline risk by allowing iterative testing and scope adjustment.
Architecture	MVC	Structures the code for junior developers, allowing parallel work on Data (Model), UI (View), and Logic (Controller).
Version Control	Git & GitHub	Essential for MVC collaboration, preventing code overwrites and providing a "revert" safety net for junior staff.
Model	SQLite	The only database that satisfies the mandatory Offline Functionality requirement without server costs.
Controller	Java	Chosen to avoid the learning curve of new languages, mitigating the risk of delays.
View	Android Studio	Provides visual tools that simplify UI creation for inexperienced developers.

CHAPTER 3: LO3 PLAN AND PRODUCE A FUNCTIONAL BUSINESS APPLICATION WITH SUPPORT DOCUMENTATION

6. (P4) Conduct a peer review of the problem definition statement, proposed solution and development strategy, documenting any feedback given

6.1. Survey Question



BudgetWise App Solution - Peer Review Survey

Thank you for participating in this survey. We are conducting a peer review for the BudgetWise business application. Please review the proposed solution discussed before answering. Your feedback will help us refine the design and identify weaknesses.

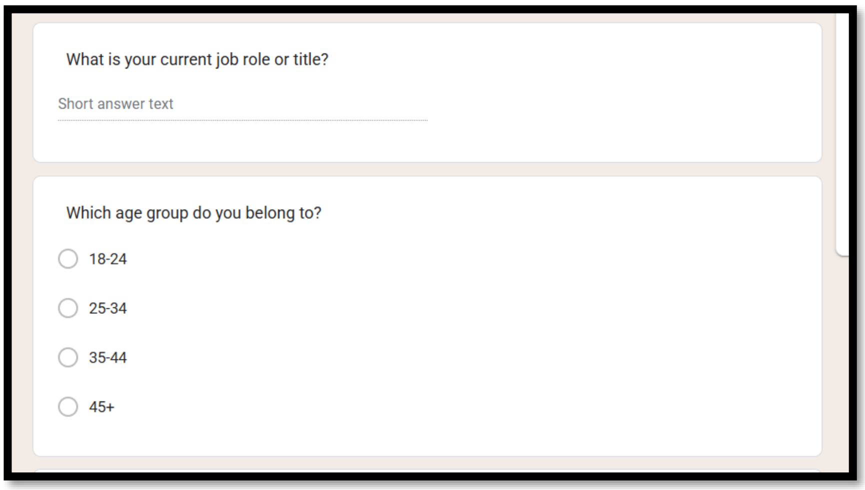
Name *

Short answer text

Email *

Short answer text

Figure 12: Survey Question 1



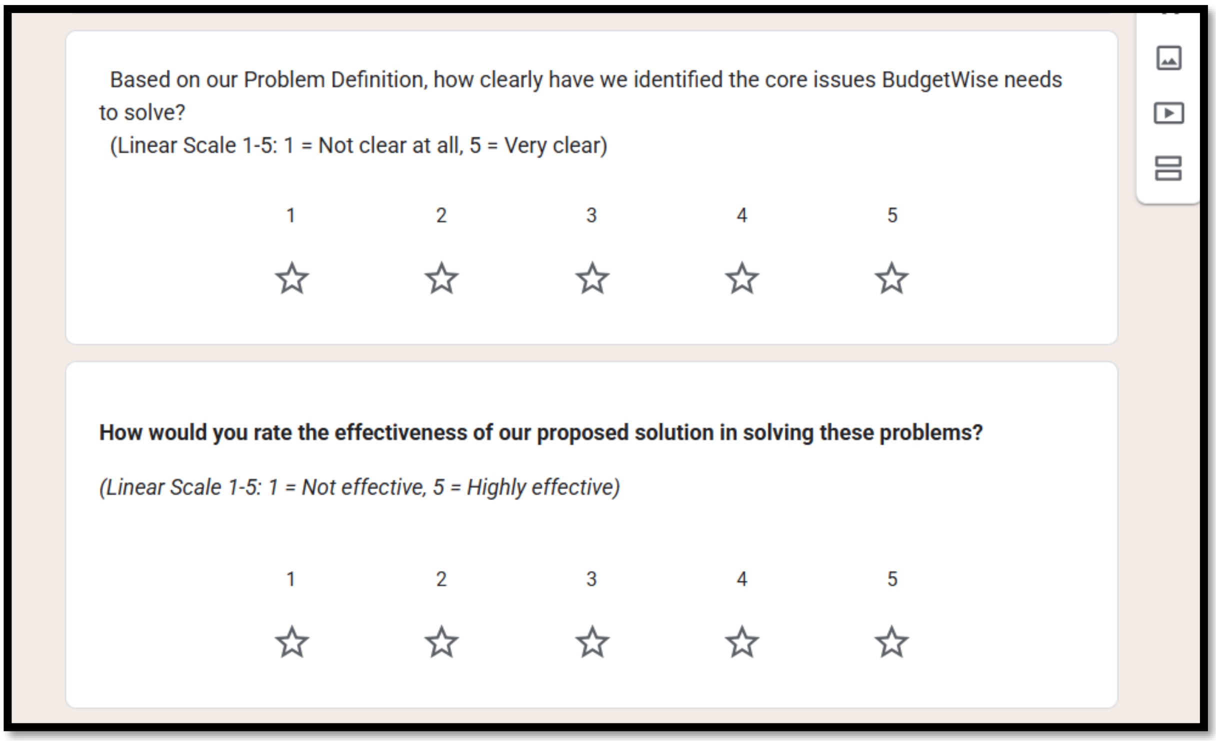
What is your current job role or title?

Short answer text

Which age group do you belong to?

- 18-24
- 25-34
- 35-44
- 45+

Figure 13: Survey Question



Based on our Problem Definition, how clearly have we identified the core issues BudgetWise needs to solve?
(Linear Scale 1-5: 1 = Not clear at all, 5 = Very clear)

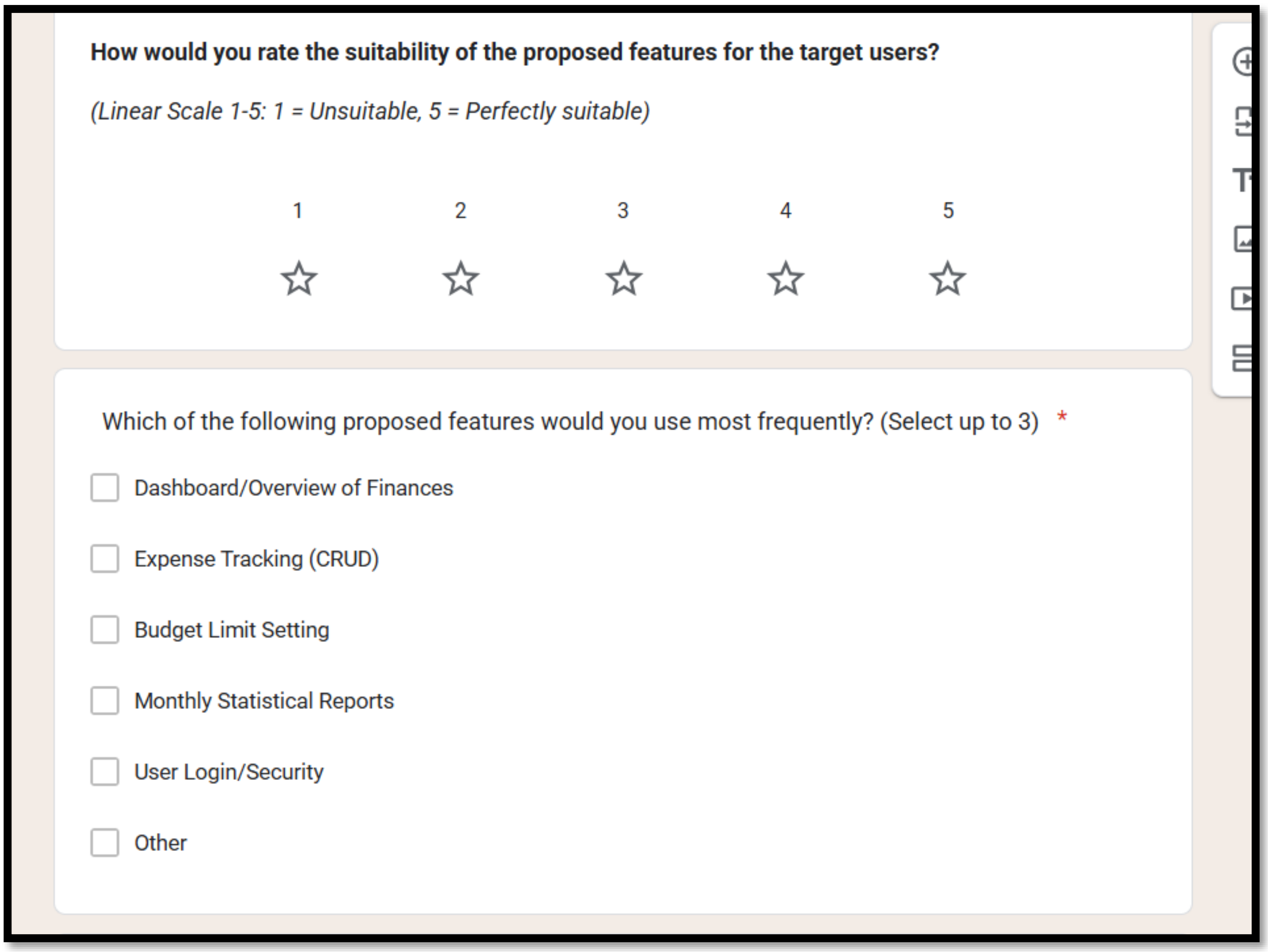
1	2	3	4	5
☆	☆	☆	☆	☆

How would you rate the effectiveness of our proposed solution in solving these problems?

(Linear Scale 1-5: 1 = Not effective, 5 = Highly effective)

1	2	3	4	5
☆	☆	☆	☆	☆

Figure 14: Survey Question 3


How would you rate the suitability of the proposed features for the target users?

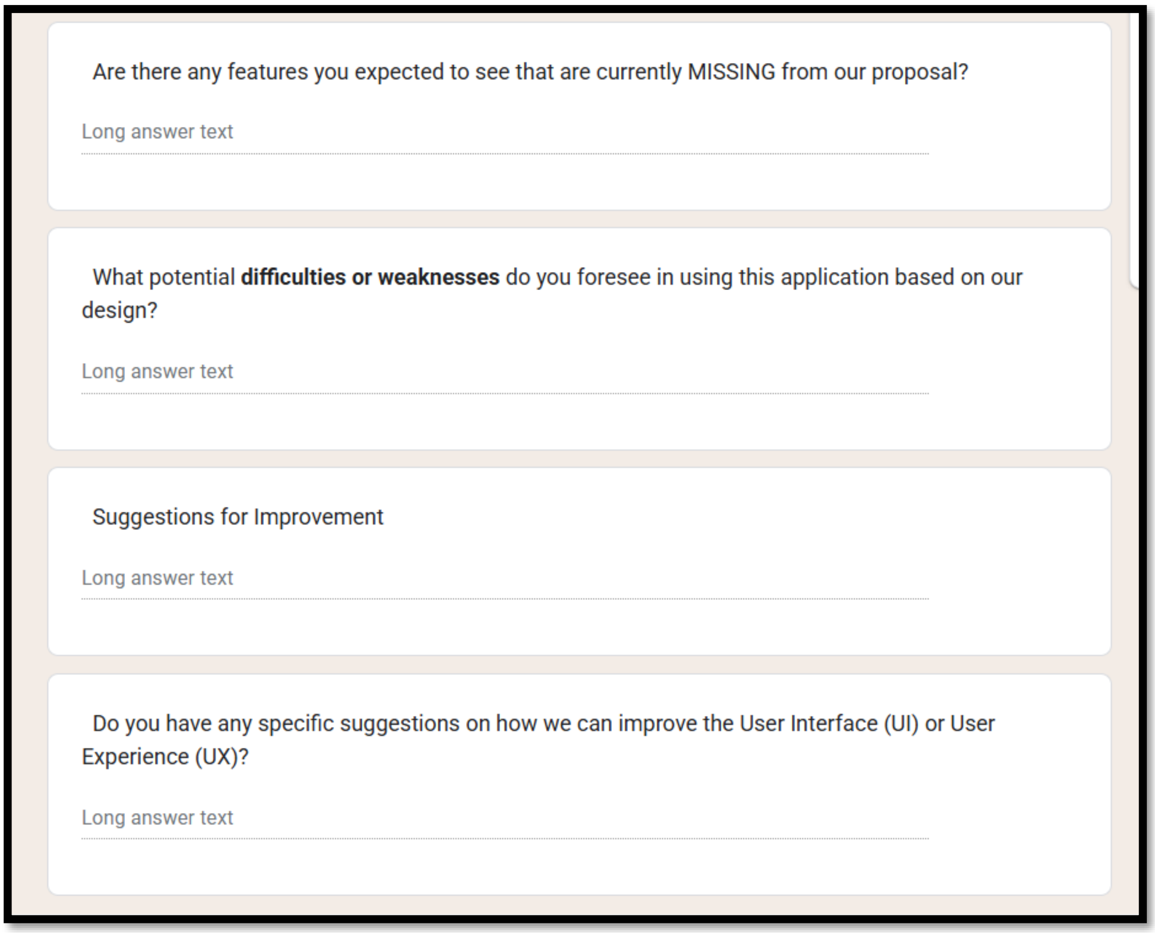
(Linear Scale 1-5: 1 = Unsuitable, 5 = Perfectly suitable)

1	2	3	4	5
☆	☆	☆	☆	☆

Which of the following proposed features would you use most frequently? (Select up to 3) *

- ☐ Dashboard/Overview of Finances
- ☐ Expense Tracking (CRUD)
- ☐ Budget Limit Setting
- ☐ Monthly Statistical Reports
- ☐ User Login/Security
- ☐ Other

Figure 15: Survey Question 4


Are there any features you expected to see that are currently MISSING from our proposal?

Long answer text

What potential **difficulties or weaknesses** do you foresee in using this application based on our design?

Long answer text

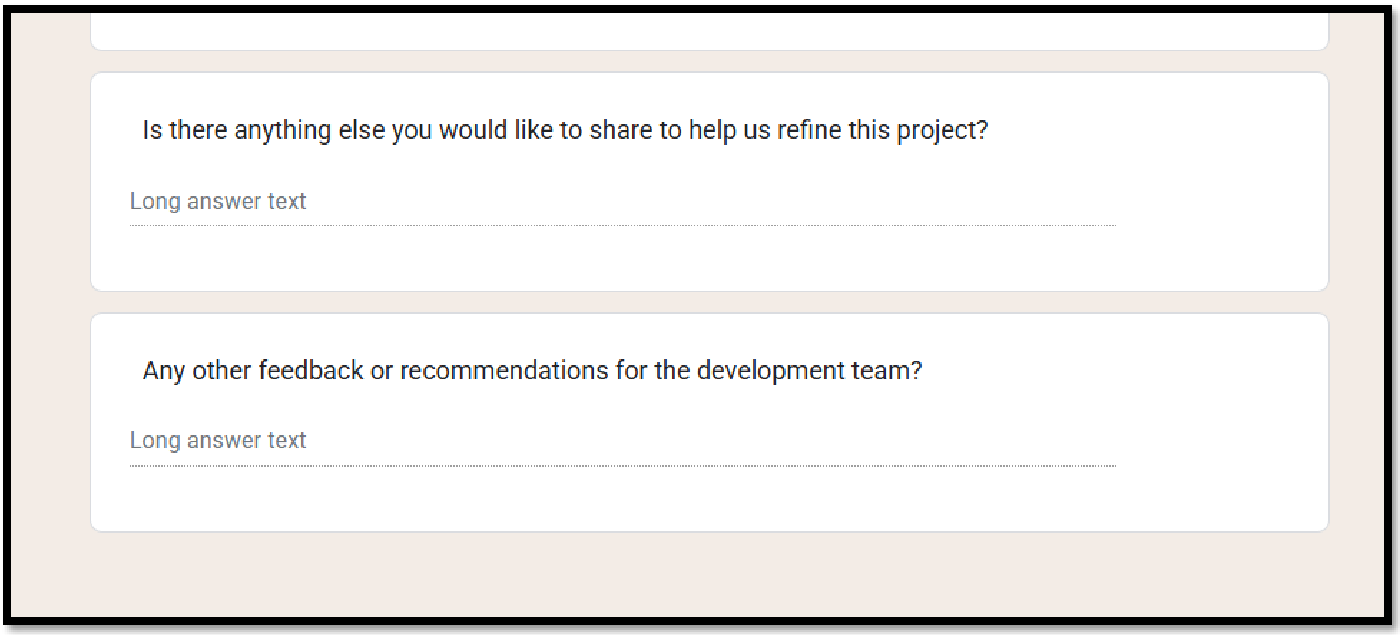
Suggestions for Improvement

Long answer text

Do you have any specific suggestions on how we can improve the User Interface (UI) or User Experience (UX)?

Long answer text

Figure 16: Survey Question 5


Is there anything else you would like to share to help us refine this project?

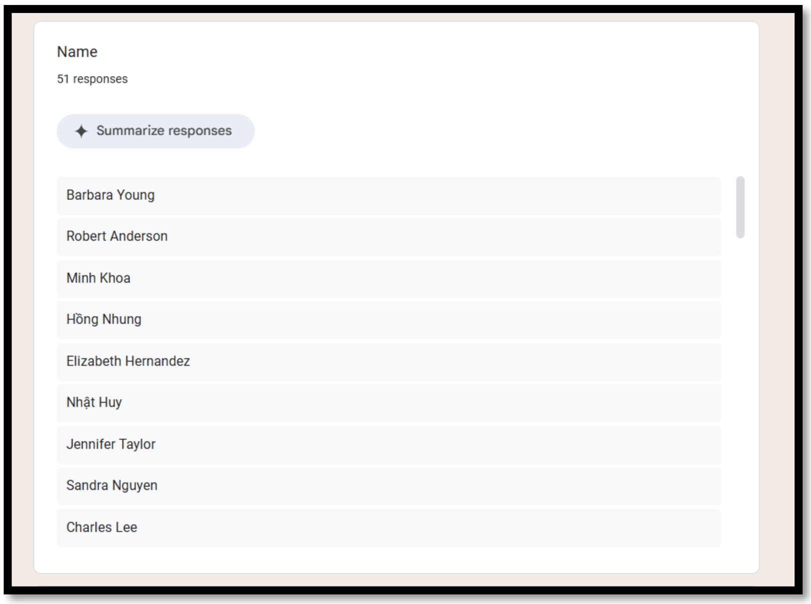
Long answer text

Any other feedback or recommendations for the development team?

Long answer text

Figure 17: Survey Question 6

6.2. Survey Answer


Name

51 responses

Summarize responses

Barbara Young

Robert Anderson

Minh Khoa

Hồng Nhung

Elizabeth Hernandez

Nhật Huy

Jennifer Taylor

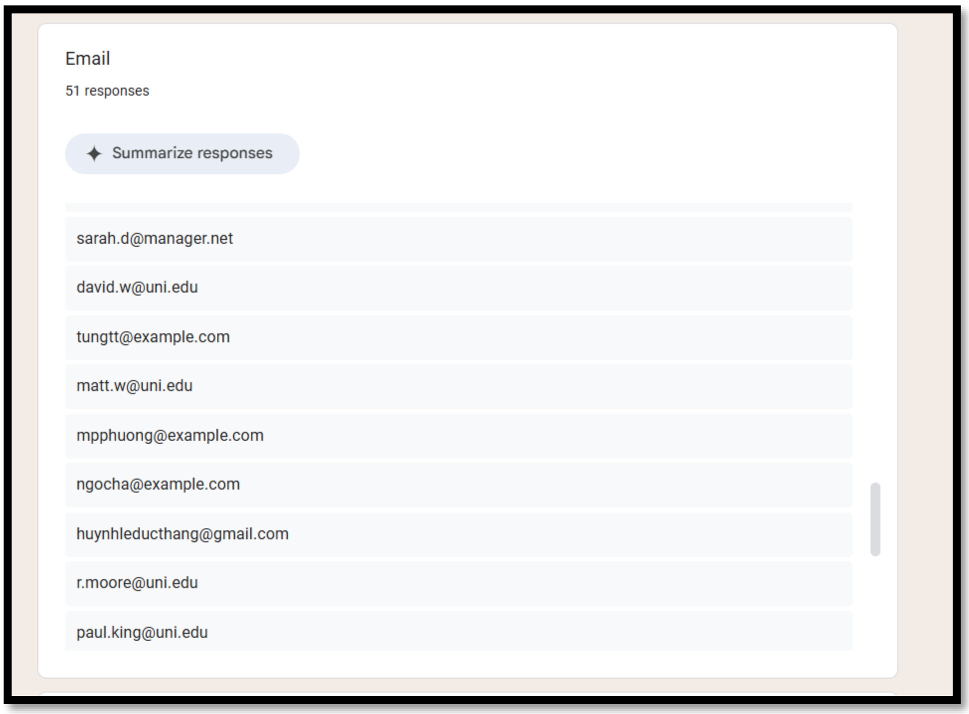
Sandra Nguyen

Charles Lee

Figure 18: Survey Answer 1

I have received 51 responses in total. This section lists the names of the people who filled out my survey.

This also the same with the email answer


Email

51 responses

Summarize responses

sarah.d@manager.net

david.w@uni.edu

tungtt@example.com

matt.w@uni.edu

mpphuong@example.com

ngocha@example.com

huynhleducthang@gmail.com

r.moore@uni.edu

paul.king@uni.edu

Figure 19: Survey Answer 2

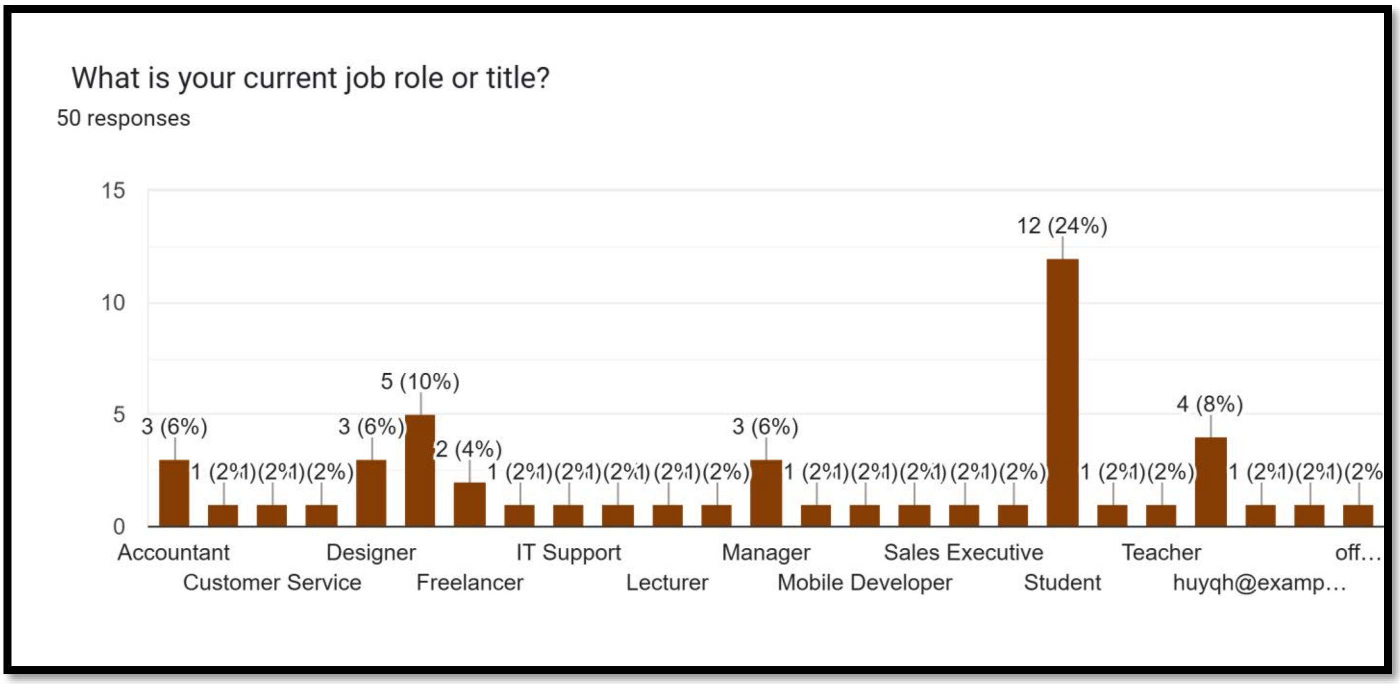


Figure 20: Survey Answer 3

This is a demographic chart from my Google Forms survey results. It shows the job titles of the 50 people who responded.

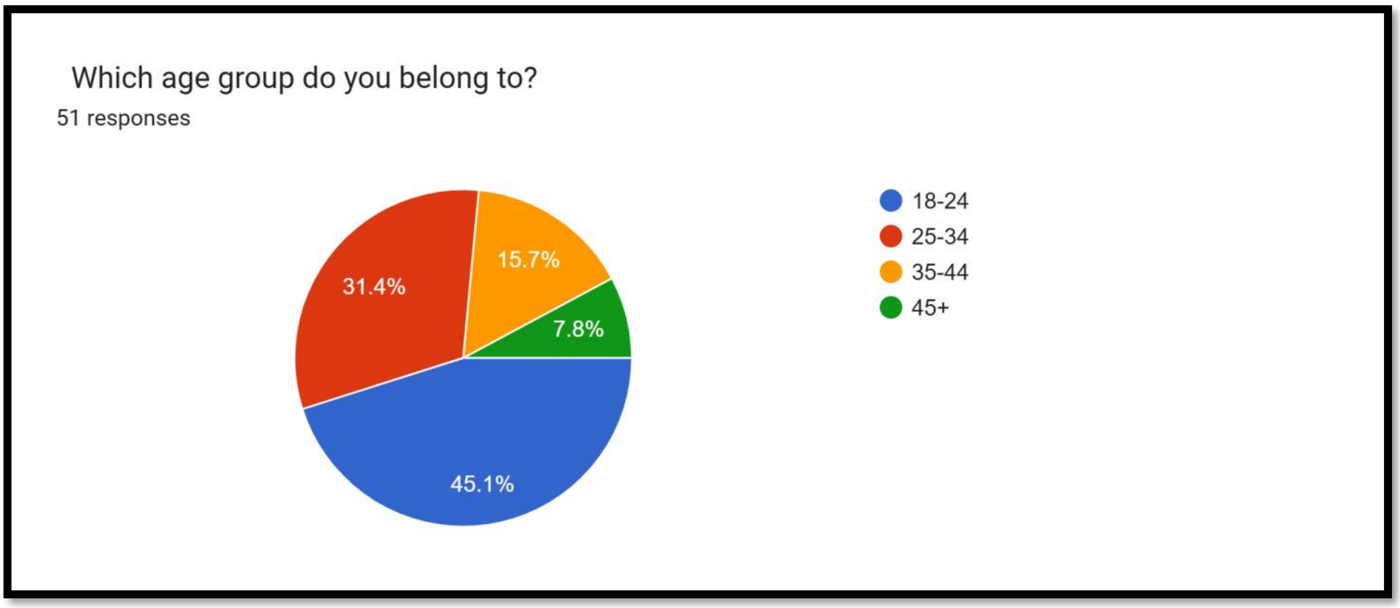


Figure 21: Survey Answer 4

The image displays the results of a survey question: "Which age group do you belong to?". It is based on a total sample size of 51 responses

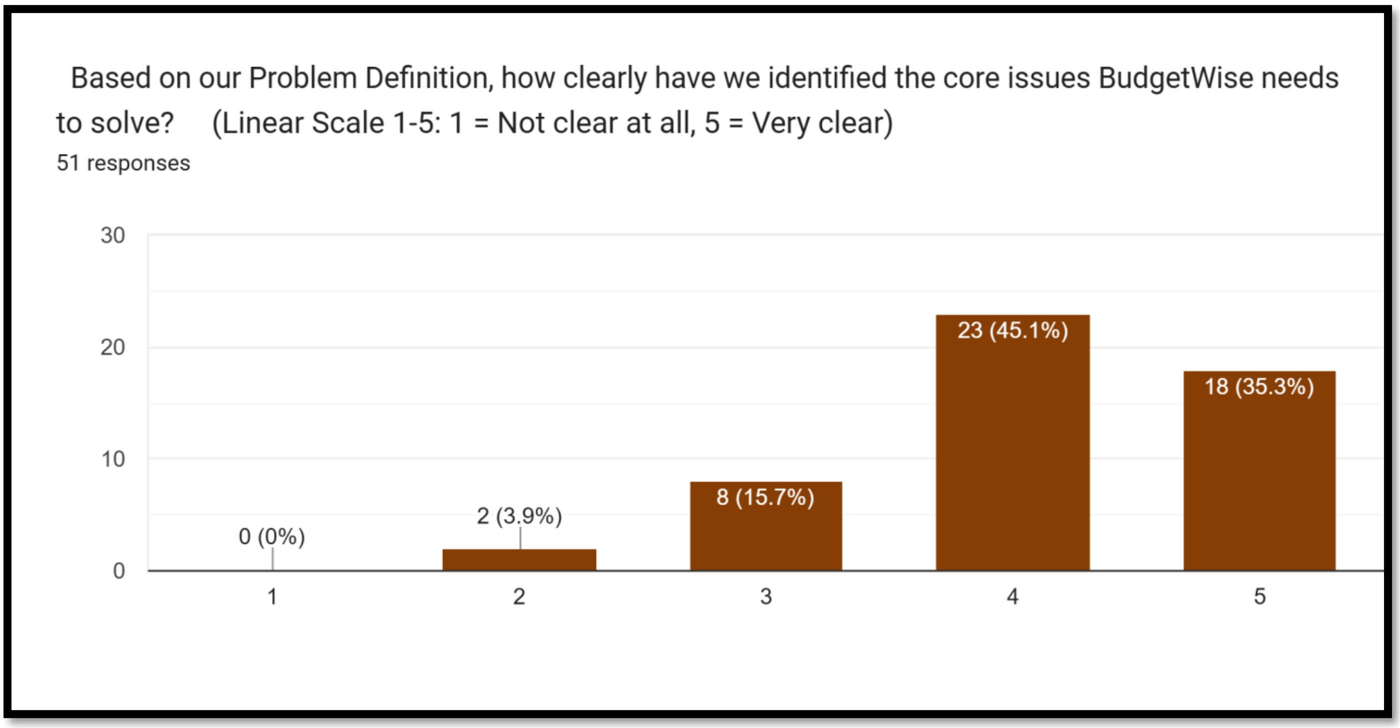


Figure 22: Survey Answer 5

The feedback is overwhelmingly positive. A combined total of 80.4% of respondents (ratings 4 and 5) feel that the core issues for BudgetWise have been identified clearly. Less than 4% of the group expressed confusion or lack of clarity.

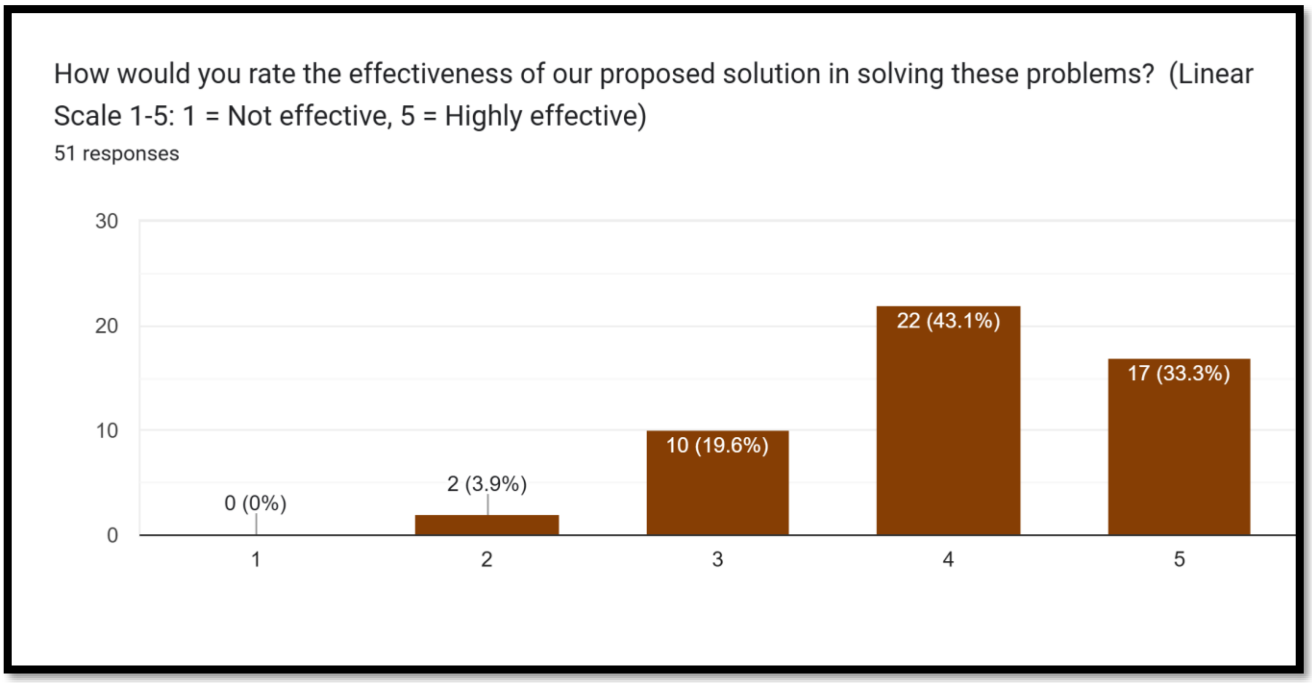


Figure 23: Survey Answer 6

The sentiment is highly positive. A combined total of 76.4% of respondents (ratings 4 and 5) believe the proposed solution is effective. Only about 4% expressed doubt, while roughly 20% remained neutral.

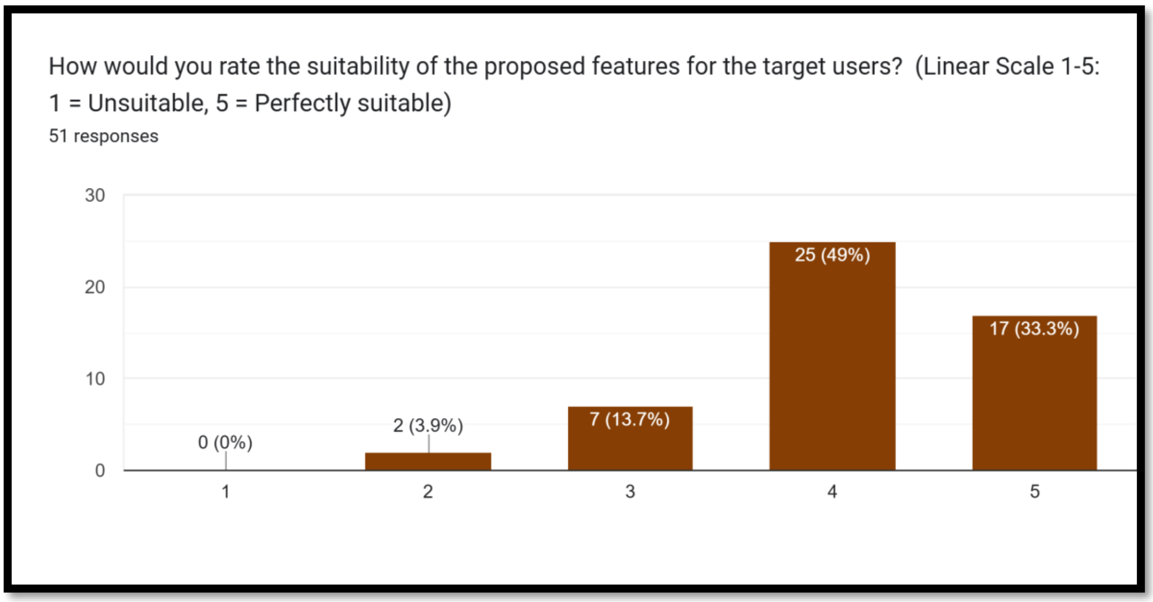


Figure 24: Survey Answer 7

This chart shows the highest positive sentiment of the three bar charts have shared so far. A combined total of 82.3% of respondents (ratings 4 and 5) believe the proposed features are highly suitable for the target users.

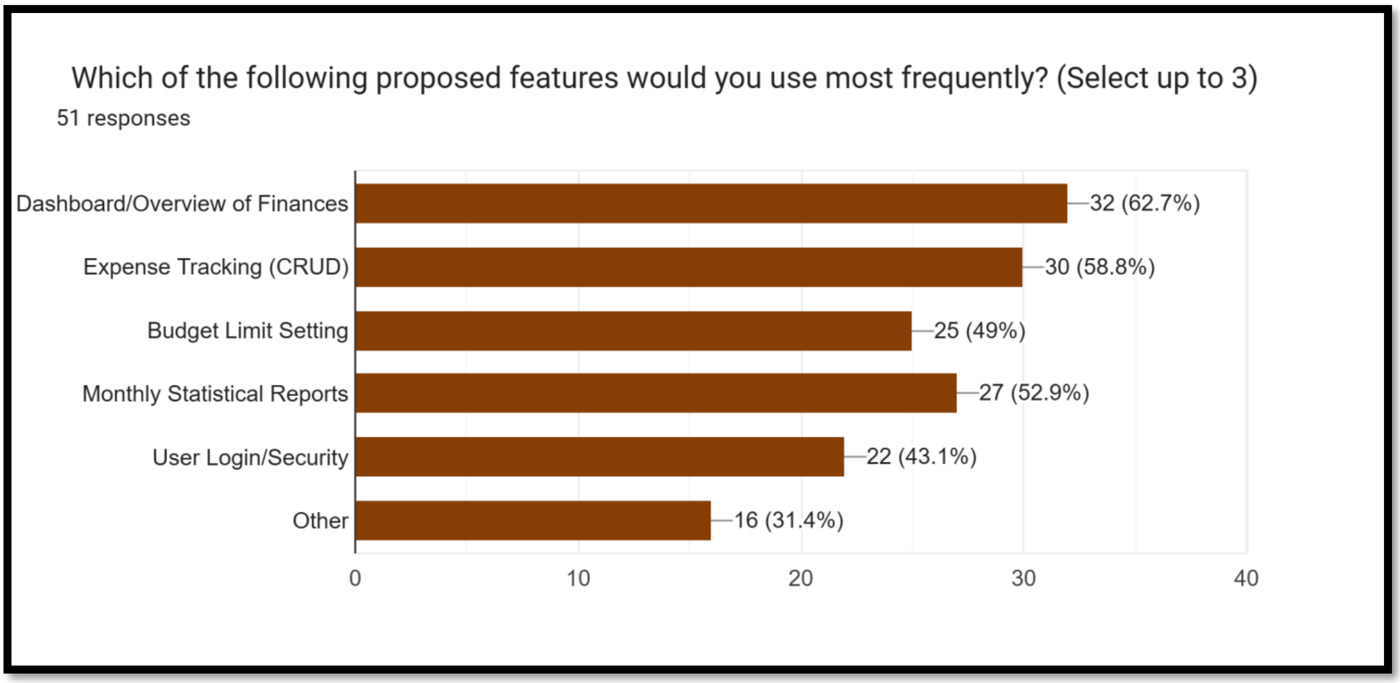
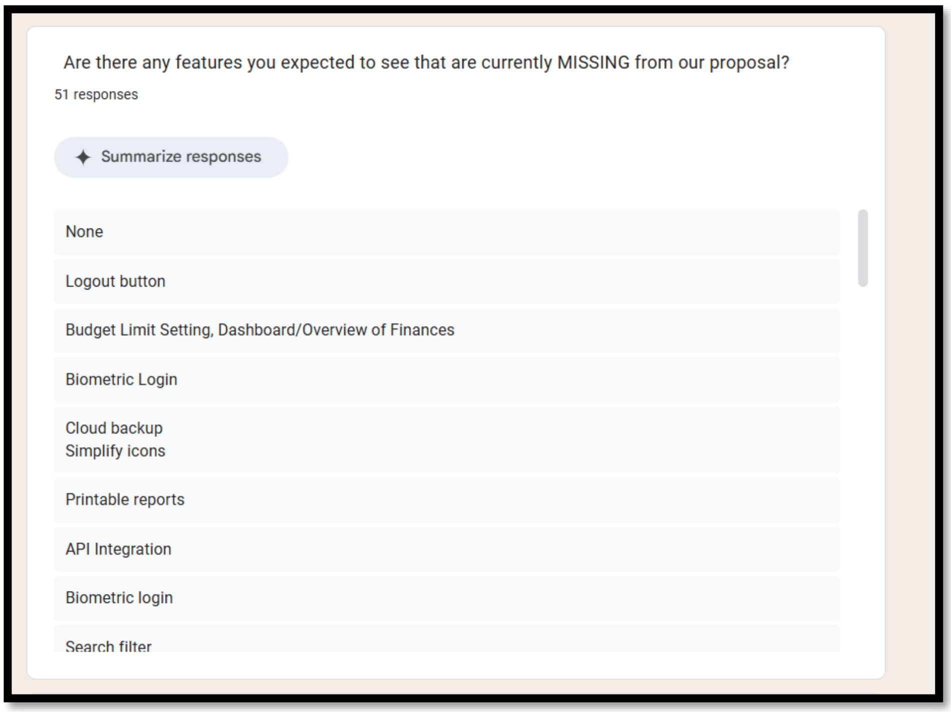


Figure 25: Survey Answer 8

The data shows a clear preference for core financial management tools. The top two features—Dashboard and Expense Tracking—are essential for daily engagement, while analytical tools like Statistical Reports are also highly valued.



Are there any features you expected to see that are currently MISSING from our proposal?

51 responses

None

Logout button

Budget Limit Setting, Dashboard/Overview of Finances

Biometric Login

Cloud backup
Simplify icons

Printable reports

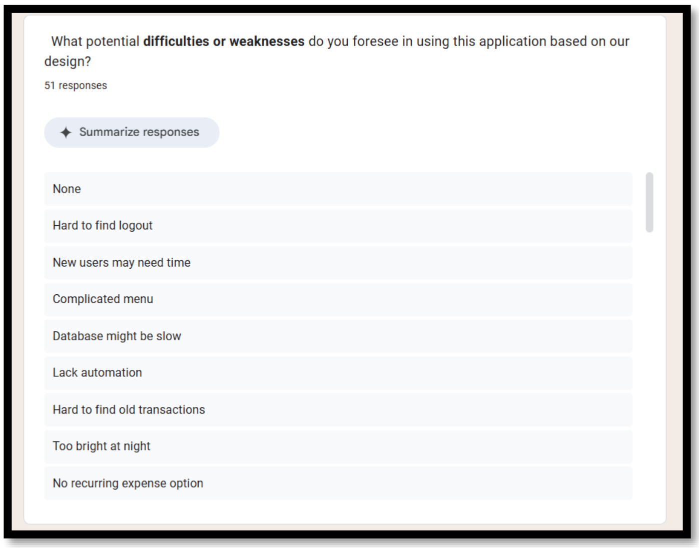
API Integration

Biometric login

Search filter

Figure 26: Survey Answer 9

The feedback indicates that while the core proposal (Dashboard, CRUD) covers the basics, users have high expectations for automation (Recurring payments, Voice input), interoperability (APIs), and modern accessibility standards (Dark mode, Biometrics).



What potential **difficulties or weaknesses** do you foresee in using this application based on our design?

51 responses

None

Hard to find logout

New users may need time

Complicated menu

Database might be slow

Lack automation

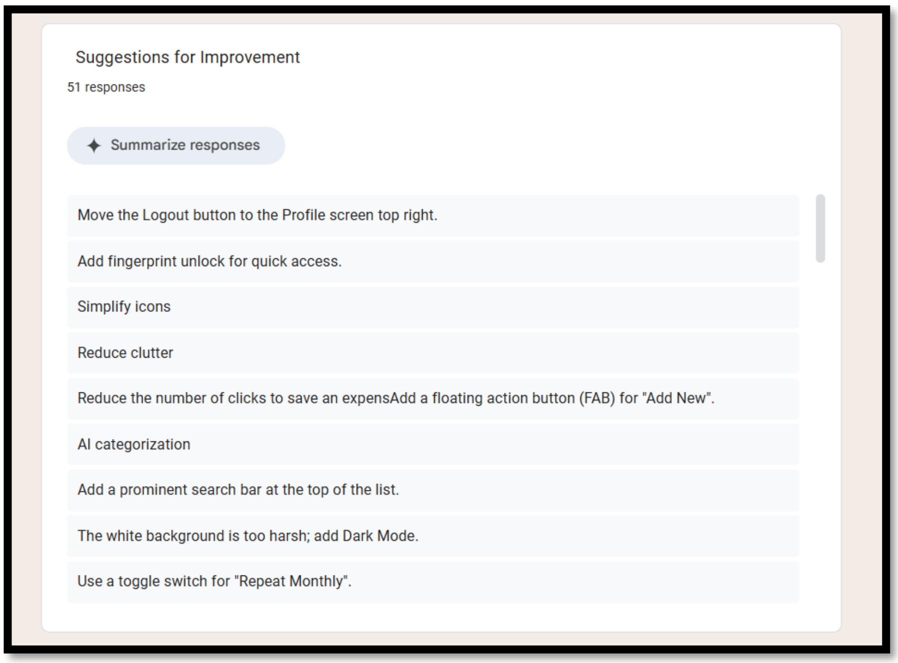
Hard to find old transactions

Too bright at night

No recurring expense option

Figure 27: Survey Answer 10

The primary fear among users is friction. They are worried the app will be slow to use (manual typing), hard to navigate (complicated menus), or technically sluggish (database speed).


Suggestions for Improvement
51 responses

Move the Logout button to the Profile screen top right.

Add fingerprint unlock for quick access.

Simplify icons

Reduce clutter

Reduce the number of clicks to save an expensAdd a floating action button (FAB) for "Add New".

AI categorization

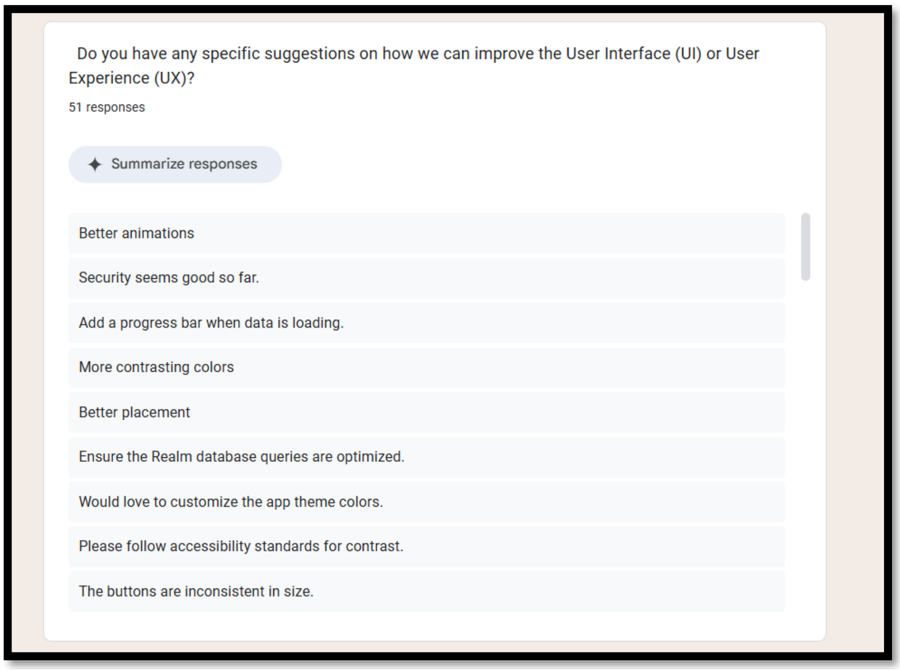
Add a prominent search bar at the top of the list.

The white background is too harsh; add Dark Mode.

Use a toggle switch for "Repeat Monthly".

Figure 28: Survey Answer 11

The users are demanding a cleaner, faster experience with fewer clicks and better visual ergonomics (less clutter, dark mode).


Do you have any specific suggestions on how we can improve the User Interface (UI) or User Experience (UX)?
51 responses

Better animations

Security seems good so far.

Add a progress bar when data is loading.

More contrasting colors

Better placement

Ensure the Realm database queries are optimized.

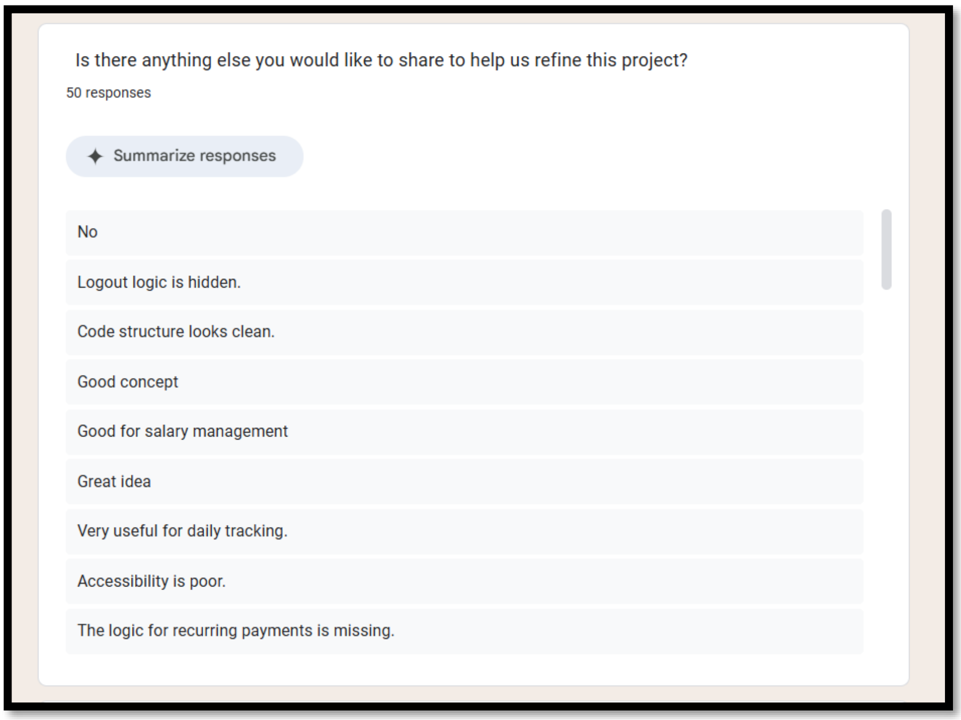
Would love to customize the app theme colors.

Please follow accessibility standards for contrast.

The buttons are inconsistent in size.

Figure 29: Survey Answer 12

The respondents are asking for a more professional and accessible polish. They want the app to not just work, but to feel consistent (button sizes), readable (contrast), and responsive (loading bars). The mention of specific technologies (Realm, Google Speech API) indicates a sophisticated user base.


Is there anything else you would like to share to help us refine this project?
50 responses

No

Logout logic is hidden.

Code structure looks clean.

Good concept

Good for salary management

Great idea

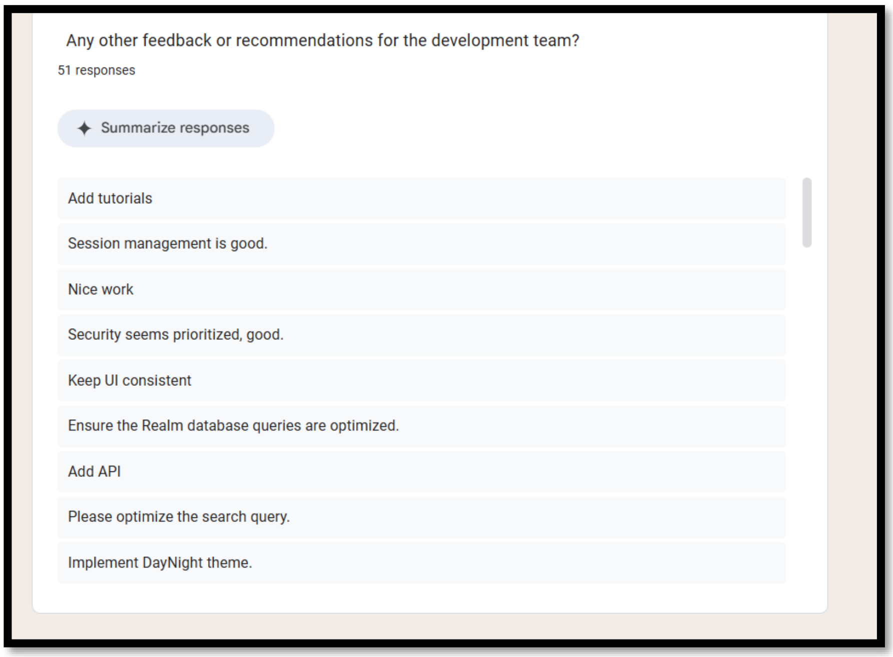
Very useful for daily tracking.

Accessibility is poor.

The logic for recurring payments is missing.

Figure 30: Survey Answer 13

General user feedback


Any other feedback or recommendations for the development team?
51 responses

Add tutorials

Session management is good.

Nice work

Security seems prioritized, good.

Keep UI consistent

Ensure the Realm database queries are optimized.

Add API

Please optimize the search query.

Implement DayNight theme.

Figure 31: Survey Answer 14

The key takeaway is to balance technical stability (fixing errors, indexing DB) with user education (FAQs, tutorials).

6.3. Personal Interview

Table 9: Personal survey 1

Persona Name	**David Wilson**
Role	University Student
Age / Tech Level	20 / High
Interview Context	Type: Semi-structured Duration: 15 mins Focus: User Interface (UI) and Speed
Goals and Needs	Speed: Needs to log costs in under 5 seconds while walking. Aesthetics: Wants a modern "Dark Mode" to reduce eye strain. Simplicity: Does not want complex charts or heavy financial terms.
Frustrations and Weaknesses	Login Delay: Found the login screen too slow for quick use. Visuals: Complained the white background is too bright/harsh at night. Boredom: Thinks the current UI looks "basic" and "outdated."
Key Quote	*"I just need to know if I can afford pizza tonight without scrolling through a boring list. Also, please add Dark Mode; my eyes hurt using this at night."*

Table 10: Personal survey 2

Persona Name	**Emily Johnson**
Role	Junior Developer
Age / Tech Level	24 / Very High
Interview Context	Type: Semi-structured Duration: 20 mins Focus: Features and Efficiency
Goals and Needs	Automation: Wants voice input or bank syncing to save time Security: Needs assurance that her data is encrypted and safe.

	Customization: Wants to create custom categories (e.g., "Tech Gadgets").
Frustrations and Weaknesses	Manual Entry: "I hate typing prices manually." Feature Gap: Frustrated by the lack of cloud sync between her phone and laptop. Rigidity: Annoyed she cannot rename default categories.
Key Quote	*"I don't have time to type 'Coffee - $5' every morning. Why can't I just say it or scan the receipt? If it's not fast, I won't use it."*

Table 11: Personal survey 3

Persona Name	Joseph Martin
Role	Department Manager
Age / Tech Level	52 / Moderate
Interview Context	Type: Semi-structured Duration: 25 mins Focus: Reporting and Management
Goals and Needs	Reporting: Needs printable reports (PDF) for physical board meetings. Oversight: Needs to see team spending totals. Readability: Needs larger text and clear buttons without confusing icons.
Frustrations and Weaknesses	Export Issues: The biggest weakness is the inability to Export to PDF. Accessibility: Found the font size too small and icons confusing. Isolation: Dislikes that he cannot share the budget view with his team.
Key Quote	*"The app works, but it's stuck on the phone. I need to print these charts to PDF for my monthly meetings. Can you make the text bigger?"*

Table 12: Personal survey 4

Persona Name	Mary Gonzalez
Role	Senior Accountant
Age / Tech Level	38 / High (specifically Excel)

Interview Context	Type: Semi-structured Duration: 20 mins Focus: Data Accuracy and Compliance
Goals and Needs	Analysis: Needs raw data to use in pivot tables. Audit: Needs to filter expenses by specific date ranges Proof: Needs to attach photos of receipts for tax purposes.
Frustrations and Weaknesses	Data Lock-in: Critical weakness is the lack of CSV/Excel Export. Navigation: Finds scrolling through history inefficient; needs a Date Filter. Compliance: Cannot use the app for work without receipt storage.
Key Quote	*"The charts are pretty, but I can't do my job without the raw data. I need to export this to Excel to double-check the tax calculations."*

6.4. Conclusion and Refinement of Project Scope

Based on the quantitative data from the survey (51 respondents) and the qualitative insights from the semi-structured interviews, the development team has refined the project scope for "BudgetWise Solutions." The feedback confirms that the core concept is solid, with 80.4% of users clearly understanding the problem and 82.3% finding the proposed features suitable.

However, to address the identified weaknesses and meet the strict 12-week timeline, the following decisions have been made regarding the final feature set:

Features to be Added (Based on Feedback)

- System-Wide Dark Mode: To address the "visual discomfort" reported by users like David and general survey respondents, we will implement a Day/Night theme toggle. This is a high-priority UI improvement to enhance accessibility.
- Search and Filter Functionality: To resolve the "inefficient history navigation" frustration, we will add a search bar and logic to filter expenses by date.
- Visual Budget Alerts: Since users prioritize "Expense Tracking" and "Dashboards", we will enhance the dashboard with color-coded progress bars (Green/Amber/Red) to provide immediate visual feedback on spending limits.

Features to be Descope (Risk Management) Despite user requests for advanced features, the following will be excluded from this MVC to mitigate the risks identified in the Risk Assessment:

- Recurring Payments & Automation: While requested by users like Emily, the logic for automated background deductions is complex and poses a high technical risk for the junior team. We will focus on manual entry stability first.

- Cloud Synchronization & Cross-Platform (iOS): To strictly adhere to the "Offline Functionality" requirement and budget constraints, the app will remain a Native Android application using SQLite. Cloud features are deferred to future updates.
- Receipt Scanning (OCR): This requires advanced API integration which exceeds the current time resources.

Final Approved Feature List for Development Based on this peer review, the development phase will focus on delivering:

- Secure Authentication (Login/Register).
- Dashboard with Real-time Charts (MPAndroidChart).
- Expense CRUD with Category Management.
- Budget Setting with Visual Alerts.
- Utilities: Calculator and Notes (Added value features).
- Settings: Profile Management and Dark Mode.

7. (M3) Interpret peer-review feedback and identify opportunities not previously considered.

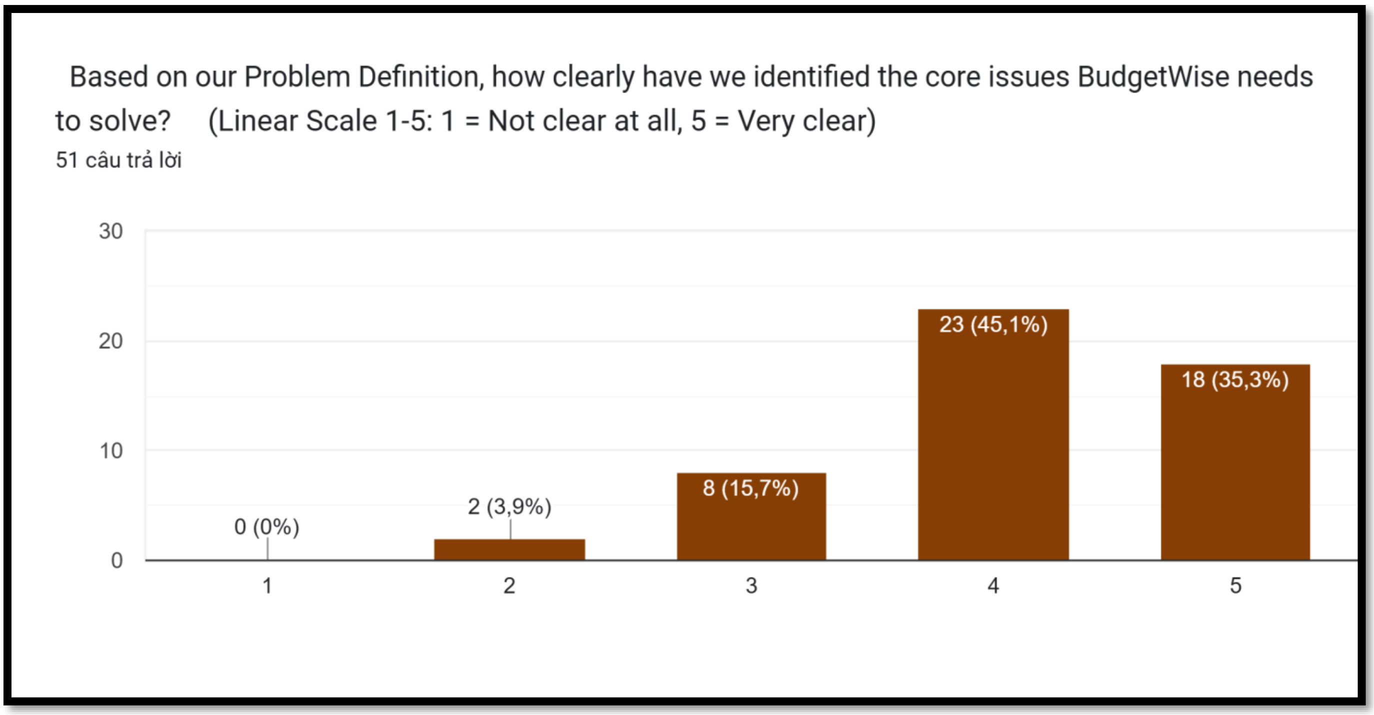


Figure 32: Survey Answer 5

Chart Title: Based on our Problem Definition, how clearly have we identified the core issues BudgetWise needs to solve?

Data Breakdown:

- Very Clear (5): 18 responses (35.3%)

- Clear (4): 23 responses (45.1%)

- Neutral (3): 8 responses (15.7%)

- Not Clear (1-2): 2 responses (3.9%)

Interpretation: The results are overwhelmingly positive. A combined total of 80.4% of respondents (selecting 4 or 5) agreed that the development team clearly identified the core issues. This indicates that the "Problem Definition" statement was well-written and relatable to the target audience.

Conclusion: The project has a solid foundation. The vast majority of stakeholders and users understand *why* this app is being built. The risk of solving the "wrong problem" is very low.

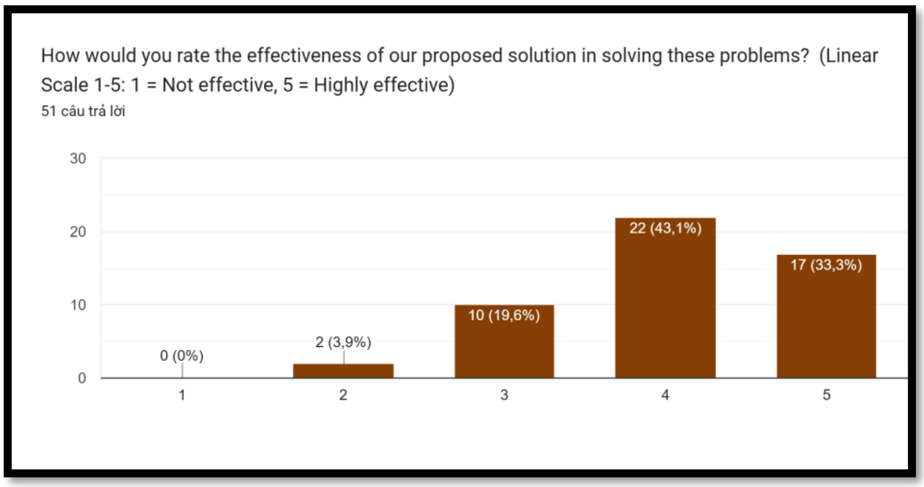


Figure 33: Survey Answer 6

Chart Title: *How would you rate the effectiveness of our proposed solution in solving these problems?*

Data Breakdown:

- Highly Effective (4-5): 76.4% combined (43.1% + 33.3%).

- Neutral (3): 19.6%.

- Ineffective (1-2): Approximately 4%.

Interpretation: While still positive, the confidence level here (76.4%) is slightly lower than the clarity of the problem (80.4%). This suggests that while users understand the problem perfectly, a segment of them (approx. 20% in the neutral zone) needs to see the actual prototype or finished product to be convinced that the *solution* will work.

Conclusion: The proposed solution is viable, but the development team needs to focus on executing the features well to convert the "Neutral" users into "Effective" voters.

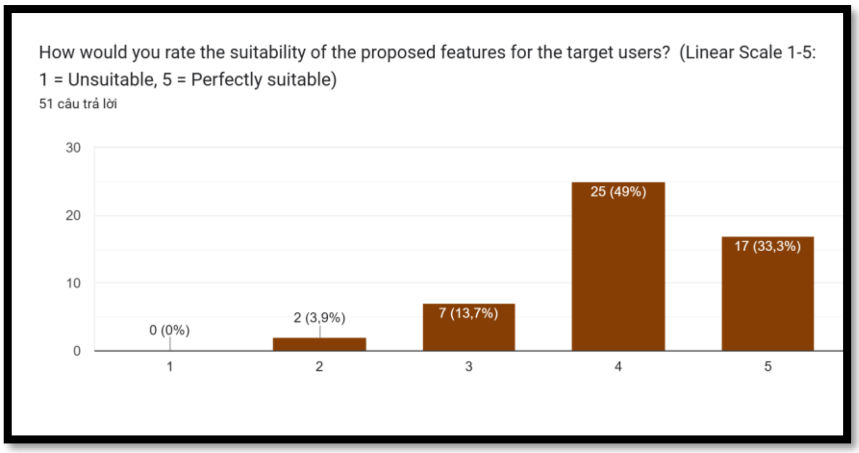


Figure 34: Survey Answer 7

Chart Title: *How would you rate the suitability of the proposed features for the target users?*

Data Breakdown:

- Suitable (4-5): 82.3% combined (49% + 33.3%).

- Neutral (3): 13.7%.

- Unsuitable (2): 3.9%.

Interpretation: This is the strongest metric in the survey. It implies that the specific features listed (Dashboard, Expense Tracking, etc.) are exactly what the users are looking for. The alignment between user needs and proposed features is excellent.

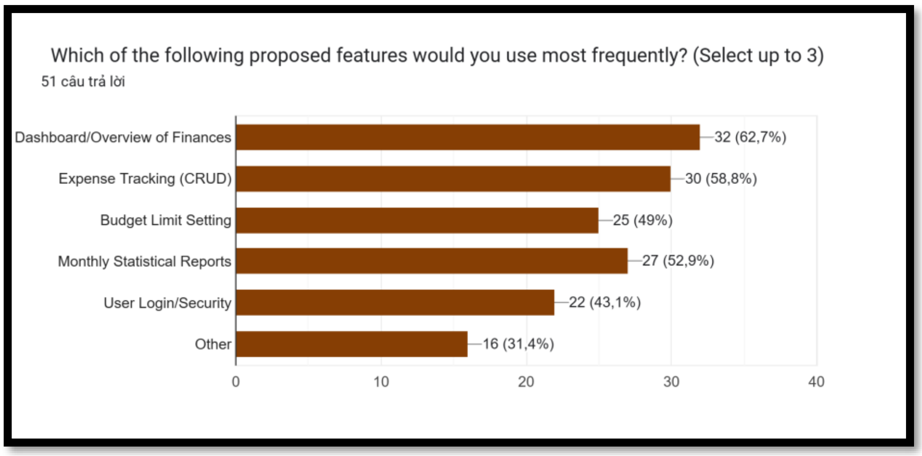


Figure 35: Survey Answer 8

Chart Title: *Which of the following proposed features would you use most frequently?*

Data Breakdown:

- Dashboard/Overview: 62.7% (32 votes).

- Expense Tracking (CRUD): 58.8% (30 votes).
- Monthly Statistical Reports: 52.9% (27 votes).
- Budget Limit Setting: 49% (25 votes).
- User Login/Security: 43.1% (22 votes).

Interpretation: Users prioritize visibility (Dashboard) and action (Tracking) over administrative tasks (Login) or restriction (Budget Limits). They want to see where their money is going immediately.

Recommendation: The development team should prioritize the UI/UX of the Dashboard and the Add Expense flow. These are the "Must-Have" features. If time runs out, complex security features or advanced budget settings are lower priority for the MVC (Model-View-Controller).

Table 13: Summary of Feedback and Improvement Opportunities

Category	Identified Weakness (Current Limitation)	Evidence (Source)	Proposed Opportunity (Improvement Solution)
User Experience (UX)	High Interaction Cost (Manual Entry): Users feel that typing every expense manually is too slow and tedious.	"I hate typing prices manually... If it's not fast, I won't use it."; "Typing on mobile is slow."	Implement Automation: Integrate Voice Input (via Google Speech API) or Optical Character Recognition (OCR) for Receipt Scanning to reduce manual typing.
User Interface (UI)	Visual Discomfort and "Basic" Design: The white background is too harsh for night use, and the design is perceived as "outdated" or "boring."	"The white background is too harsh."; "Please add Dark Mode; my eyes hurt."	Modernize UI and Dark Mode: Implement a system-wide Dark Mode toggle and refresh the UI with modern typography and consistent button sizes.
Data Management	Data Lock-in (Lack of Export): Users cannot extract their data for external use (e.g.,	"The biggest weakness is the inability to Export to PDF."; "Cannot	Data Export Features: Develop functionality to export reports as PDF (for managers) and

Category	Identified Weakness (Current Limitation)	Evidence (Source)	Proposed Opportunity (Improvement Solution)
	printing for meetings or analyzing in Excel).	do my job without... CSV/Excel Export."	CSV/Excel (for accountants/analysis).
Navigation	Inefficient History Navigation: Finding old transactions requires excessive scrolling, which is inefficient for power users.	"Hard to find old transactions."; "Finds scrolling through history inefficient; needs a Date Filter."	Advanced Search and Filtering: Add a prominent Search Bar and Date Range Filters to allow users to quickly locate specific past expenses.
Functionality	Lack of Recurring Expenses: The app does not support automatic logging for fixed monthly costs (e.g., rent, subscriptions).	"No recurring expense option."; "Logic for recurring payments is missing."	Recurring Payments Module: Add a toggle switch for "Repeat Monthly" when adding an expense to automatically generate these entries.
Accessibility	Poor Accessibility: Font sizes are too small for older users, and contrast/icons are confusing.	"Accessibility is poor."; "Found the font size too small and icons confusing."	Accessibility Settings: Implement dynamic text sizing (scaling) and ensure color contrast meets WCAG standards. Add labels to icons to reduce confusion.
Connectivity	Offline Limitation: The app is stuck on a single device with no ability to sync data between phone and laptop.	"Frustrated by the lack of cloud sync."; "Consider implementing cloud sync soon."	Cloud Synchronization: Plan for a future update to integrate Cloud Sync (e.g., Firebase) to allow multi-device access and data backup.

The most critical opportunities to address immediately are Dark Mode, Export Options, and Search Filters, as these address specific user frustrations regarding usability and professional utility

8. (M4) Develop a functional business application based on a specific software design document, with supportive evidence of using the preferred tools, techniques and methodologies

8.1. Feature Selection

To demonstrate the correct application of the selected development tools (Android Studio, Java) and techniques (Third-party Library Integration), I have selected the "Expense Analysis & Visualization" feature for detailed analysis

Rationale: This feature fulfills "Requirement 4: Expense Overview and Visualization". It requires a complex architecture that retrieves financial data from the SQLite database, processes it based on time filters (Daily/Weekly/Monthly), and renders it visually using the MPAndroidChart library

8.2. Architecture Explanation

As defined in the technical specifications, the BudgetWise Solutions application implements the MVC (Model-View-Controller) architectural pattern to ensure code maintainability and separation of concerns

- Model: Responsible for data management and business logic. In this feature, the DatabaseHelper class acts as the Model, handling all raw SQL queries to retrieve expense totals from the SQLite database

- View: Responsible for the user interface. This includes the XML layout (activity_analysis.xml) and UI components such as BarChart (for visualization), TextView (for time tabs), and the BottomNavigationView

- Controller: Acts as the intermediary. It handles user inputs (e.g., clicking the "Weekly" tab), requests data from the Model, processes that data, and updates the View. In the code, ChartActivity.java and the utility class ChartHelper.java function as the Controller

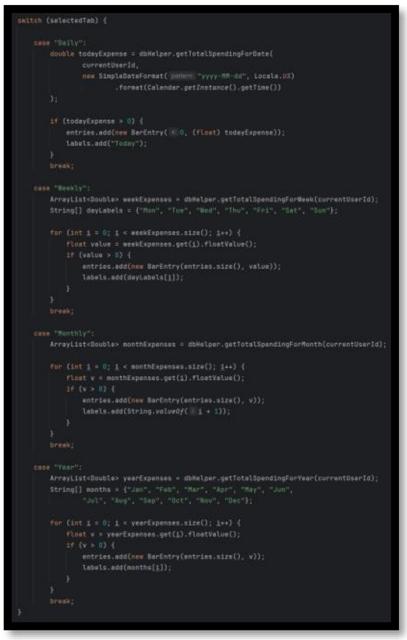
9.3. Code Evidence

We will now analyze the following core code to illustrate how the MVC architecture is implemented for this feature

Model Interaction (Data Retrieval)

This code is located in ChartHelper.java. It demonstrates the Controller requesting specific financial data from the Model (DatabaseHelper) based on the user's selection

Code Snippet 1: Requesting Data from the Model



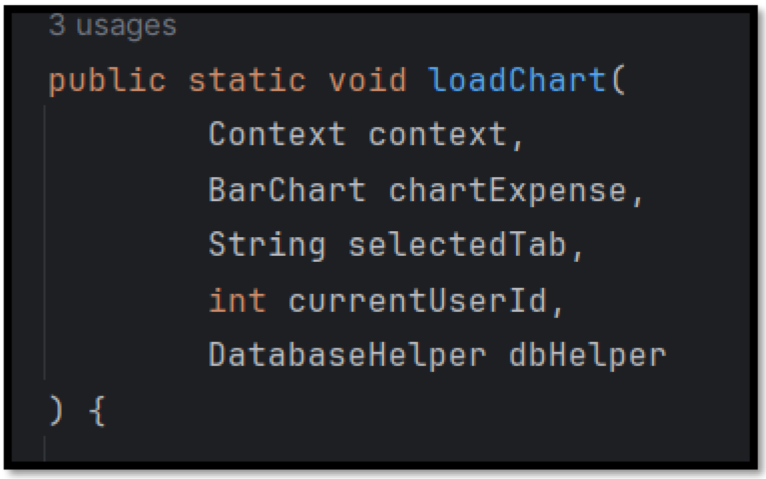
```java
switch (selectedTab) {

    case "Daily":
        double todayExpense = dbHelper.getTotalSpendingForDate(
                currentUserId,
                new SimpleDateFormat( pattern: "yyyy-MM-dd", Locale.US)
                        .format(Calendar.getInstance().getTime())
        );

        if (todayExpense > 0) {
            entries.add(new BarEntry( x: 0, (float) todayExpense));
            labels.add("Today");
        }
        break;

    case "Weekly":
        ArrayList<Double> weekExpenses = dbHelper.getTotalSpendingForWeek(currentUserId);
        String[] dayLabels = {"Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"};

        for (int i = 0; i < weekExpenses.size(); i++) {
            float value = weekExpenses.get(i).floatValue();
            if (value > 0) {
                entries.add(new BarEntry(entries.size(), value));
                labels.add(dayLabels[i]);
            }
        }
        break;

    case "Monthly":
        ArrayList<Double> monthExpenses = dbHelper.getTotalSpendingForMonth(currentUserId);

        for (int i = 0; i < monthExpenses.size(); i++) {
            float v = monthExpenses.get(i).floatValue();
            if (v > 0) {
                entries.add(new BarEntry(entries.size(), v));
                labels.add(String.valueOf( i: i + 1));
            }
        }
        break;

    case "Year":
        ArrayList<Double> yearExpenses = dbHelper.getTotalSpendingForYear(currentUserId);
        String[] months = {"Jan", "Feb", "Mar", "Apr", "May", "Jun",
                "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"};

        for (int i = 0; i < yearExpenses.size(); i++) {
            float v = yearExpenses.get(i).floatValue();
            if (v > 0) {
                entries.add(new BarEntry(entries.size(), v));
                labels.add(months[i]);
            }
        }
        break;
}
```

Figure 36: Requesting Data from the Model

This proves the separation of data. ChartHelper does not calculate the SQL queries itself; it delegates data retrieval to the dbHelper (Model)

B. Controller Logic (Data Processing & Visual Logic)

To adhere to "Clean Code" principles and reduce the complexity of the Activity, the logic for formatting chart data was separated into the ChartHelper class



```java
3 usages
public static void loadChart(
        Context context,
        BarChart chartExpense,
        String selectedTab,
        int currentUserId,
        DatabaseHelper dbHelper
) {
```

Figure 37: loadChart

Code Snippet 2: Processing Logic for Visualization

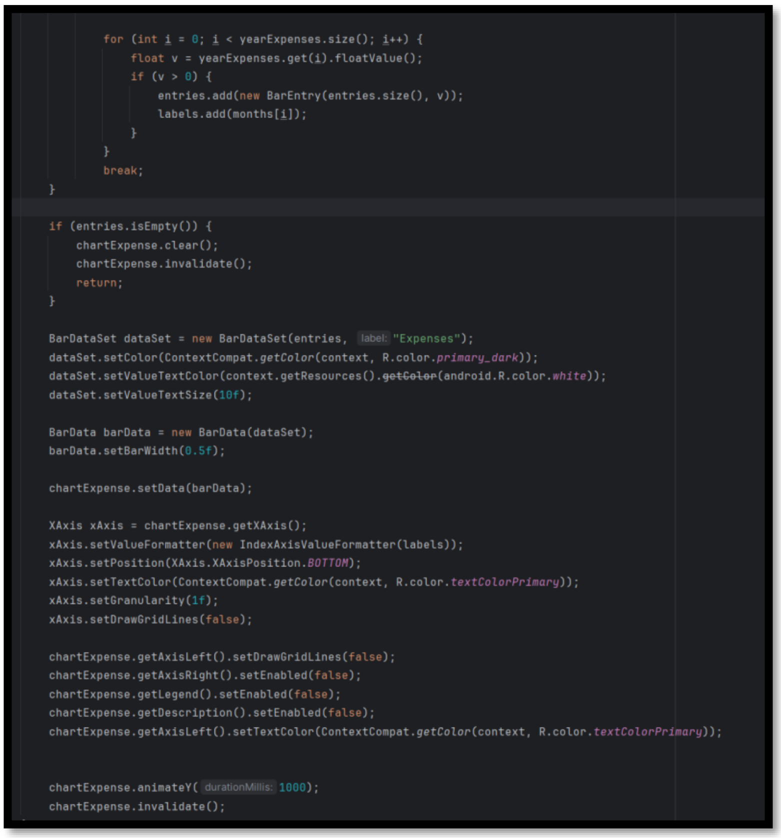

```java
            for (int i = 0; i < yearExpenses.size(); i++) {
                float v = yearExpenses.get(i).floatValue();
                if (v > 0) {
                    entries.add(new BarEntry(entries.size(), v));
                    labels.add(months[i]);
                }
            }
            break;
    }

    if (entries.isEmpty()) {
        chartExpense.clear();
        chartExpense.invalidate();
        return;
    }

    BarDataSet dataSet = new BarDataSet(entries,  label: "Expenses");
    dataSet.setColor(ContextCompat.getColor(context, R.color.primary_dark));
    dataSet.setValueTextColor(context.getResources().~~getColor~~(android.R.color.white));
    dataSet.setValueTextSize(10f);

    BarData barData = new BarData(dataSet);
    barData.setBarWidth(0.5f);

    chartExpense.setData(barData);

    XAxis xAxis = chartExpense.getXAxis();
    xAxis.setValueFormatter(new IndexAxisValueFormatter(labels));
    xAxis.setPosition(XAxis.XAxisPosition.BOTTOM);
    xAxis.setTextColor(ContextCompat.getColor(context, R.color.textColorPrimary));
    xAxis.setGranularity(1f);
    xAxis.setDrawGridLines(false);

    chartExpense.getAxisLeft().setDrawGridLines(false);
    chartExpense.getAxisRight().setEnabled(false);
    chartExpense.getLegend().setEnabled(false);
    chartExpense.getDescription().setEnabled(false);
    chartExpense.getAxisLeft().setTextColor(ContextCompat.getColor(context, R.color.textColorPrimary));


    chartExpense.animateY( durationMillis: 1000);
    chartExpense.invalidate();
```

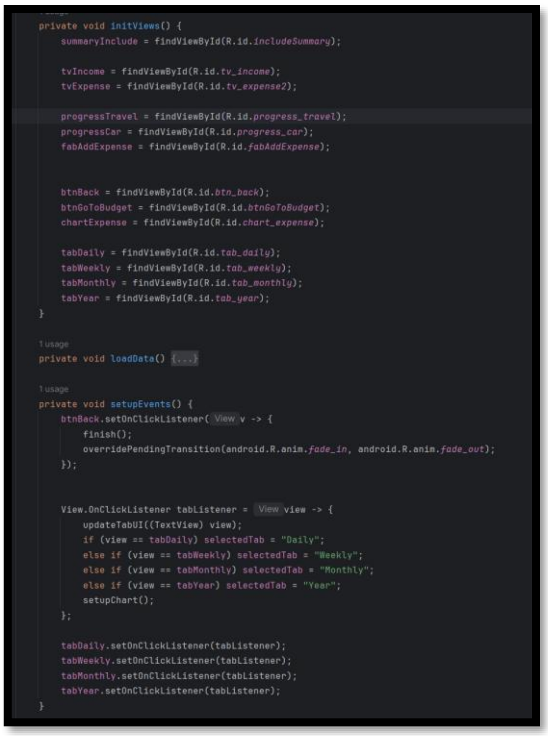
Figure 38: Processing Logic for Visualization

This code demonstrates the technical skill of integrating the external MPAndroidChart library. It processes raw lists of numbers into visual BarEntry objects that the View can render

C. View Initialization & Event Handling

This code is located in ChartActivity.java. It initializes the UI components and listens for user interactions

Code Snippet 3: View Initialization and Event Listening



```
private void initViews() {
    summaryInclude = findViewById(R.id.includeSummary);

    tvIncome = findViewById(R.id.tv_income);
    tvExpense = findViewById(R.id.tv_expense2);

    progressTravel = findViewById(R.id.progress_travel);
    progressCar = findViewById(R.id.progress_car);
    fabAddExpense = findViewById(R.id.fabAddExpense);


    btnBack = findViewById(R.id.btn_back);
    btnGoToBudget = findViewById(R.id.btnGoToBudget);
    chartExpense = findViewById(R.id.chart_expense);

    tabDaily = findViewById(R.id.tab_daily);
    tabWeekly = findViewById(R.id.tab_weekly);
    tabMonthly = findViewById(R.id.tab_monthly);
    tabYear = findViewById(R.id.tab_year);
}

1 usage
private void loadData() {...}

1 usage
private void setupEvents() {
    btnBack.setOnClickListener( View v -> {
        finish();
        overridePendingTransition(android.R.anim.fade_in, android.R.anim.fade_out);
    });


    View.OnClickListener tabListener =  View view -> {
        updateTabUI((TextView) view);
        if (view == tabDaily) selectedTab = "Daily";
        else if (view == tabWeekly) selectedTab = "Weekly";
        else if (view == tabMonthly) selectedTab = "Monthly";
        else if (view == tabYear) selectedTab = "Year";
        setupChart();
    };

    tabDaily.setOnClickListener(tabListener);
    tabWeekly.setOnClickListener(tabListener);
    tabMonthly.setOnClickListener(tabListener);
    tabYear.setOnClickListener(tabListener);
}
```

Figure 39: View Initialization and Event Listening

9.4. Functional Evidence

The following screenshot demonstrates the feature running in the Android Emulator (Pixel 9, API 36)

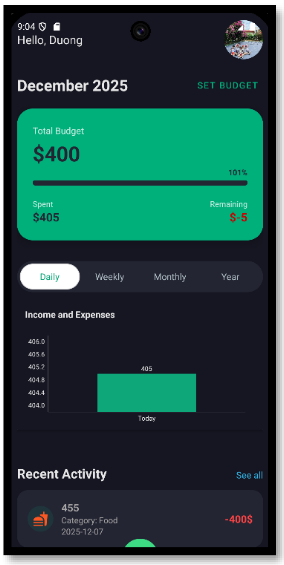


Figure 40: Android Emulator (Pixel 9, API 36)

Description of Evidence

- Interface: The tabs (Daily, Weekly, Monthly) are rendered correctly using TextViews as defined in initViews

- Visualization: The BarChart successfully renders the spending data with the animation defined in ChartHelper

- Data Accuracy: The "Spent" and "Remaining" figures match the calculations performed by the dbHelper class

9.5. Critical Review & Justification

The implementation of this feature provides strong technical evidence for the following

- Correct Architecture Implementation: The code clearly separates the logic (ChartHelper), the data (DatabaseHelper), and the UI (ChartActivity). This adheres to the MVC pattern proposed in the design phase, ensuring the code is organized and scalable

- Advanced Tool Usage: The successful integration of MPAndroidChart proves the ability to research and utilize third-party Android libraries to solve complex visualization problems, rather than relying solely on basic Android components

- Adherence to Requirements: The dynamic switching between timeframes (Daily/Weekly/Monthly) directly addresses the user requirement for detailed financial trend analysis

9. (P5) Develop a functional business application with support documentation based on a specified business problem.

9.1. WireFrame Of The Android Project

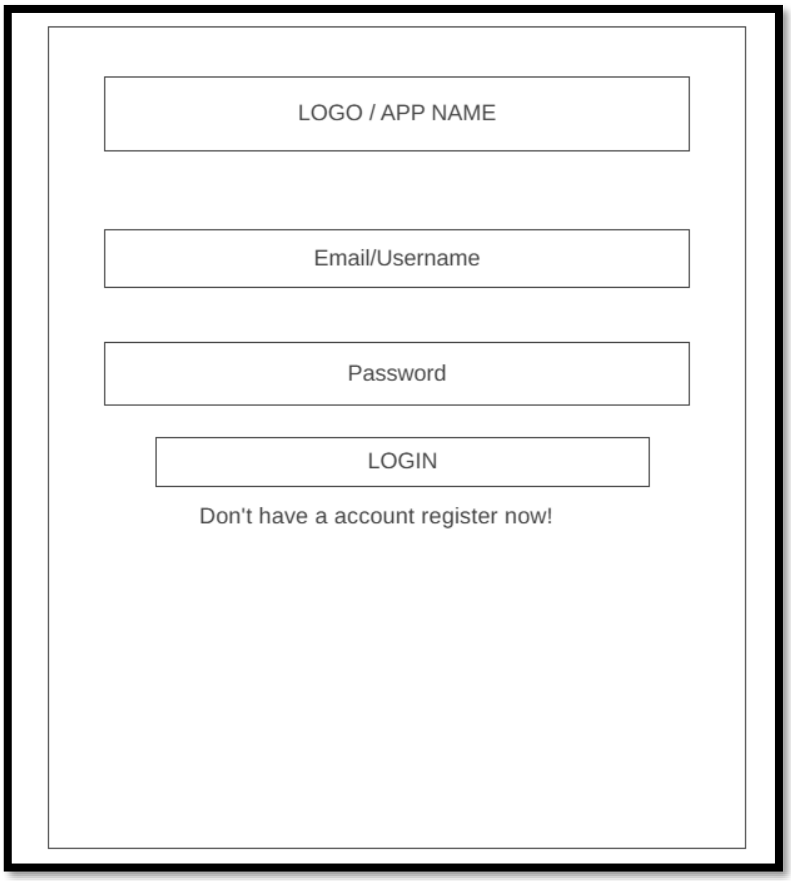


Figure 41: Login Screen

Login Screen (activity_login)

Purpose: This is the entry point for users to access their accounts securely.

Key Elements:

- **Curved Header:** A distinct visual element housing the "Expense Manager" branding, setting the app's identity right away.
- **Toggle Tabs:** A clear switch between "Sign In" and "Sign Up," allowing users to quickly choose their path without navigating to a separate screen.
- **Input Fields:** Standard fields for email/username and password, with an eye icon for password visibility.
- **Actions:** "Remember me" checkbox for convenience and a "Forgot Password?" link for account recovery.

- **Social Login:** Options to sign in with Facebook or Google, offering a faster alternative to traditional registration.

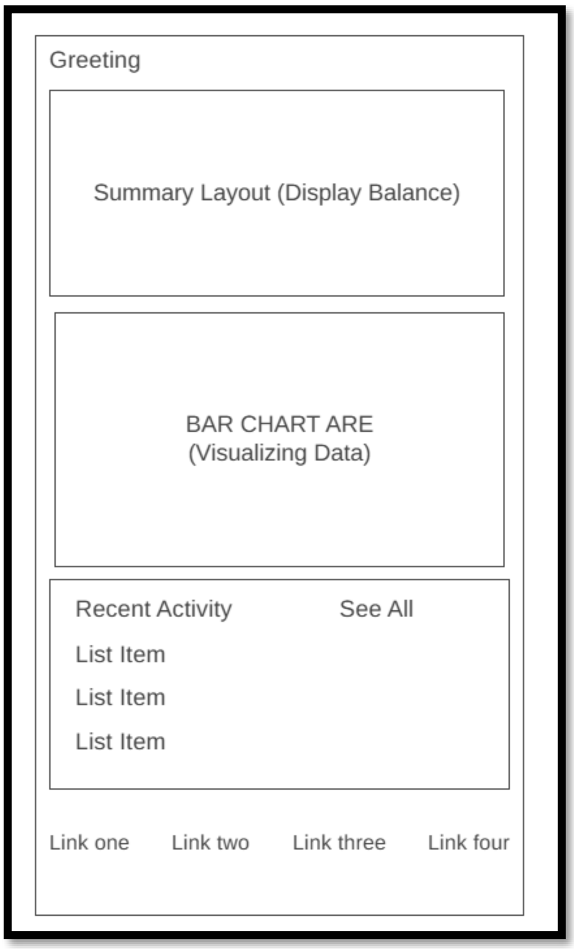


Figure 42: Activity Dashboard

Dashboard (activity_dashboard)

- Purpose: The central hub where users get an immediate overview of their financial health.
- Key Elements:
  - Header: Displays a personalized greeting and profile picture, making the app feel personal.
  - Summary Cards: Two prominent cards showing total "Income" and "Expense" at a glance, likely with percentage changes to indicate trends.
  - Time Filters: Tabs (Daily, Weekly, Monthly, Yearly) allow users to adjust the scope of the data displayed in the chart and summary.
  - Bar Chart: A visual representation of income vs. expenses over the selected time period, making it easy to spot spending patterns.

- Recent Activity: A condensed list of the most recent transactions, giving users a quick check on their latest spending.
- Bottom Navigation: Persistent access to other core areas: Home, Analysis, Budget, and Profile.
- Floating Action Button (FAB): A primary call-to-action button, centrally placed for quickly adding new expenses.

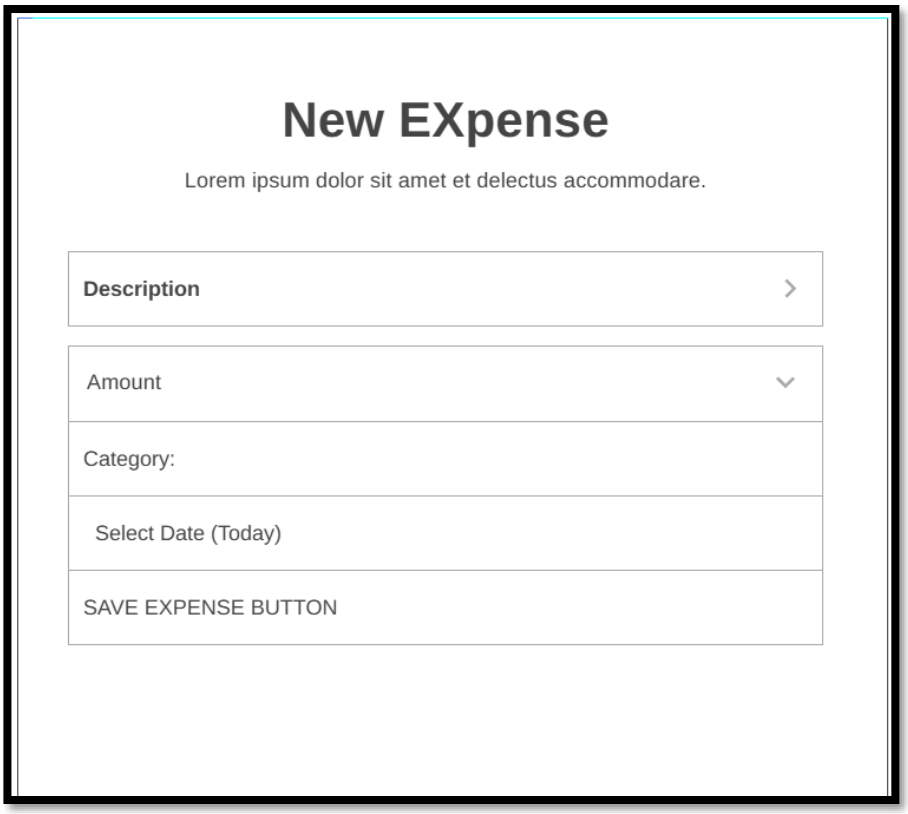


Figure 43: Add Expense

Purpose: The specific screen for logging a new transaction.

Key Elements:

- Large Amount Display: The input for the amount is prominent, emphasizing the most critical piece of data.
- Description: A text field to label the expense (e.g., "Lunch," "Taxi").
- Category Selector: A dropdown or spinner to categorize the expense (e.g., Food, Transport), which is crucial for later analysis.
- Date Picker: Allows users to backdate expenses or confirm the current date.

- Attachments: Placeholders for adding receipts (camera) or notes, adding rich detail to the transaction record.

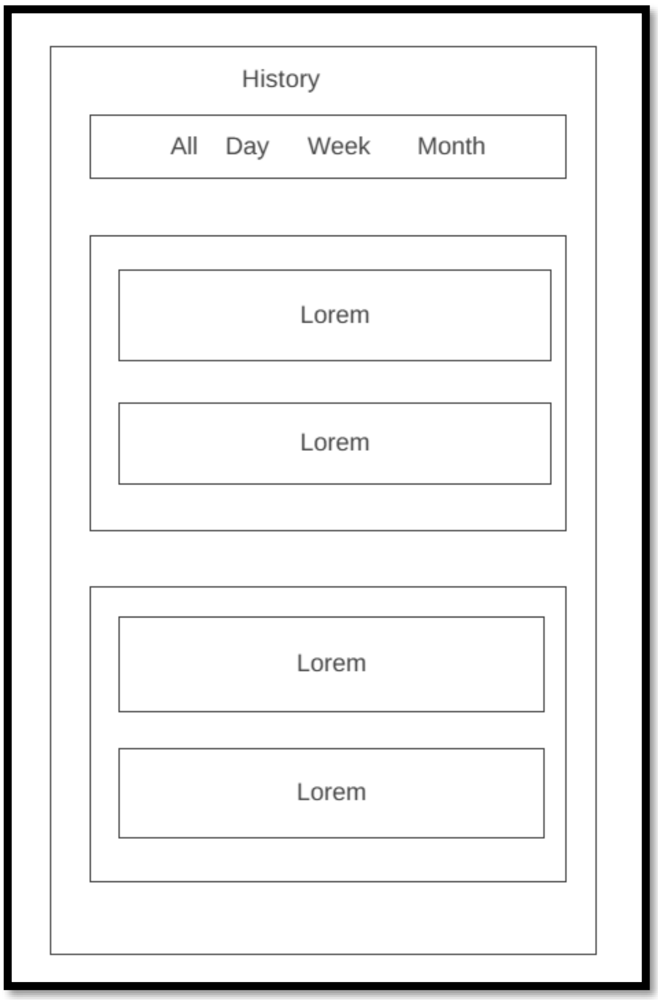
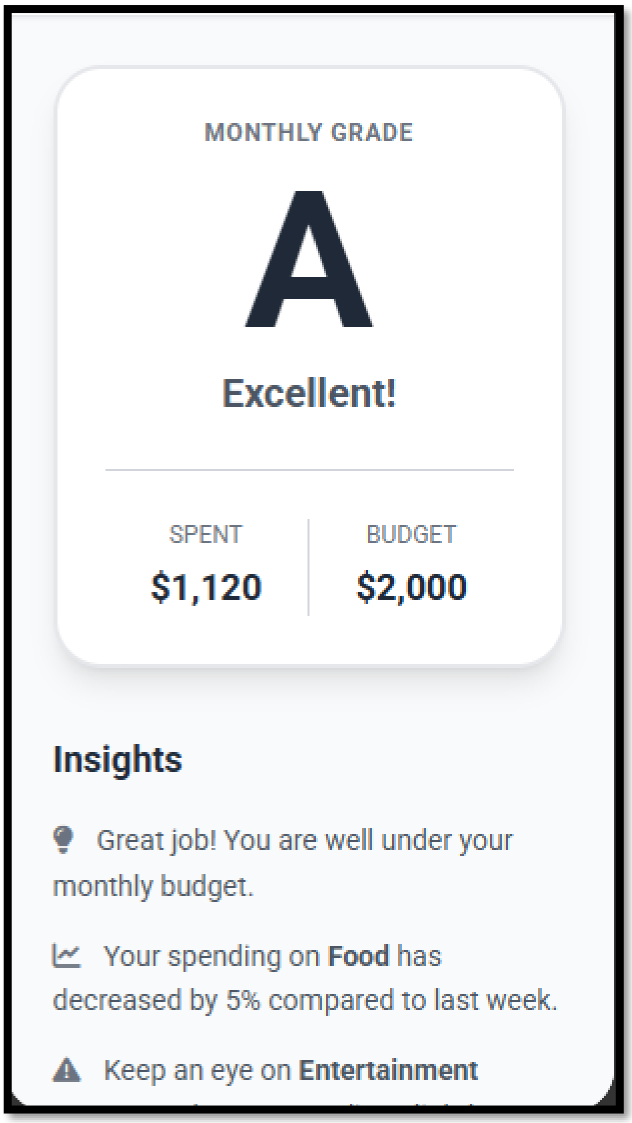


Figure 44: History


MONTHLY GRADE

A

Excellent!

SPENT $1,120

BUDGET $2,000

Insights

Great job! You are well under your monthly budget.

Your spending on **Food** has decreased by 5% compared to last week.

Keep an eye on **Entertainment**

Figure 45: Spending Analyze

Purpose: To provide deeper insights and a "grade" on the user's financial habits.

Key Elements:

- Report Card: A gamified element showing a letter grade (e.g., "A") based on spending performance, making financial management feel more engaging.
- Spent vs. Budget: A direct comparison of actual spending against the set budget.
- Insights Section: Text-based feedback offering specific advice (e.g., "Spending on Food has decreased"), which is more actionable than raw numbers.
- Progress Bar: A visual indicator of how much of the budget has been consumed, helping users pace their spending for the rest of the month.

9.2. Application Overview

Application Name: BudgetWise Solutions (Mobile App)

Purpose: The BudgetWise Solutions application is a comprehensive personal finance management tool designed to help users track their daily spending, manage monthly budgets, and analyze financial habits. Beyond finance, it integrates utility features like a task manager (Notes) and a calculator to serve as a central productivity hub for students and young professionals.

Link to my project: https://github.com/Phinnf/ASM_Android_Code

Target Audience

- University Students: Who need to manage limited funds across categories like Tuition, Rent, and Food.

- Young Professionals: Who require a simple, offline tool to track personal cash flow.

7.2. Technical Specifications

The application was built using the following technologies and architecture:

Programming Language: Java (Native Android Development).

Development Environment (IDE): Android Studio Ladybug/Koala.

Database: SQLite. The app uses a custom DatabaseHelper class to manage relational tables for Users, Expenses, Budgets, and Notes.

Architecture Pattern: MVC (Model-View-Controller).

- *Model:* DatabaseHelper.java and classes (e.g., Note.java, HistoryItem.java).

- *View:* XML Layouts (Material Design components) and Activities.

- *Controller:* Activity classes (e.g., DashboardActivity, BudgetActivity) that handle logic and user input.

Key Libraries and Tools:

- MPAndroidChart: Used in ChartHelper.java to render dynamic Bar Charts for financial analysis.

-Rhino (Mozilla): Used in CalculatorActivity.java to handle complex mathematical string evaluations.

-Android Material Design: For UI components like BottomNavigationView, FloatingActionButton, and Cards.

s7.3. User Guide and Features

User Authentication (Login and Register)

The app is secured with a user authentication system.

- Registration: New users can create an account by providing their Name, Email, and a strong Password (requires uppercase, special characters, etc.).

- Login: Users sign in with their email and password.

- Auto-Login: The "Remember Me" checkbox saves the user's session, allowing them to bypass the login screen on subsequent visits.

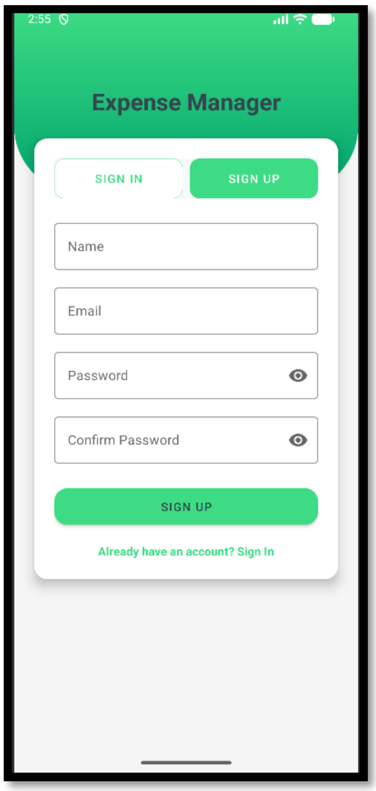


Figure 46: User Registration Screen

How to Register a New Account**:**

- Launch the BudgetWise Solutions application.

- On the Login screen, click the "Sign Up" tab.

- Enter your Full Name.

- Enter a valid Email Address.

- Enter a Password (Note: Password must be at least 8 characters and include an uppercase letter, a number, and a special character).

- Re-enter the password in the Confirm Password field.

- Tap the "Sign Up" button. If successful, you will be redirected to the Login screen.

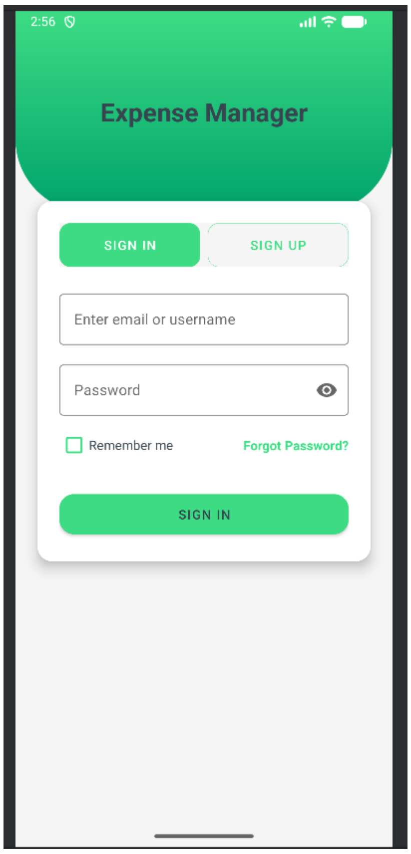


Figure 47: User Registration Screen

- Enter your registered Email and Password.

- (Optional) Check the "Remember Me" box to stay logged in for future sessions.

- Tap "Sign In". You will be directed to the Dashboard.

The Dashboard (Home)

The Dashboard serves as the central hub. It displays:

- Financial Summary: Shows the total budget, total spending, and remaining balance for the current month.

- Quick Actions: A "See All" button to view transaction history and a FloatingActionButton (+) to add new expenses.

- Visual Chart: A bar chart showing spending trends (Daily, Weekly, Monthly).

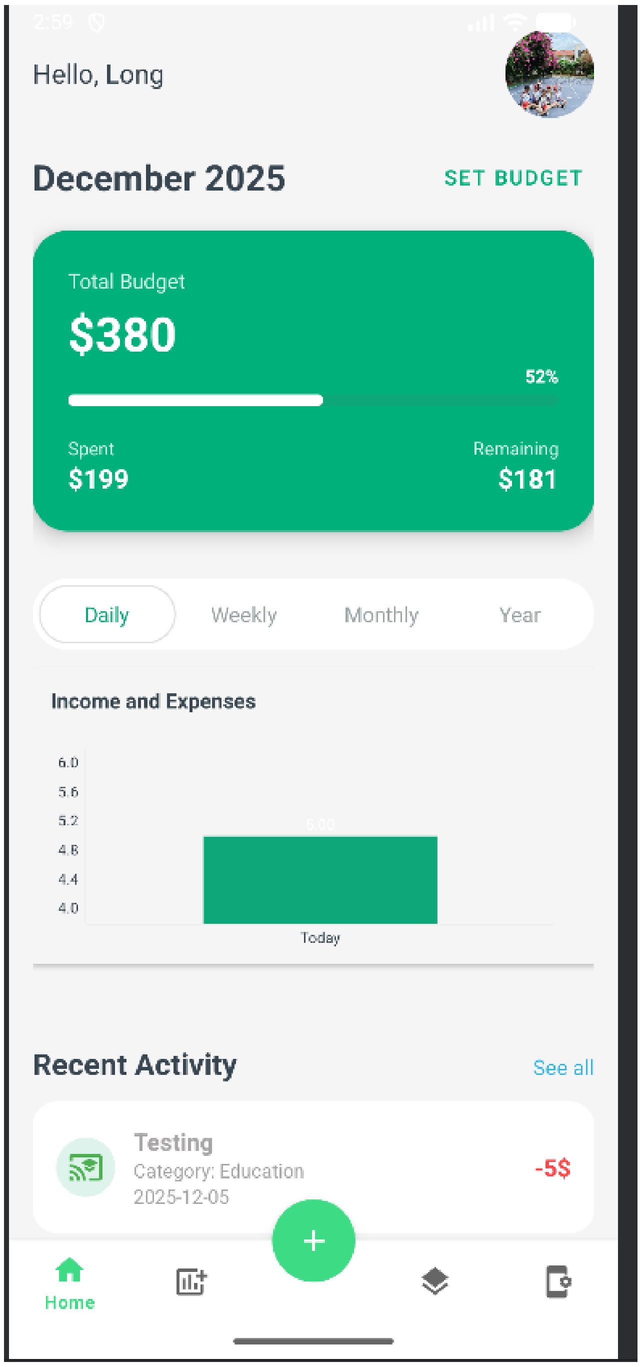


Figure 48: Dashboard Interface

Adding an Expense

To record a new transaction:

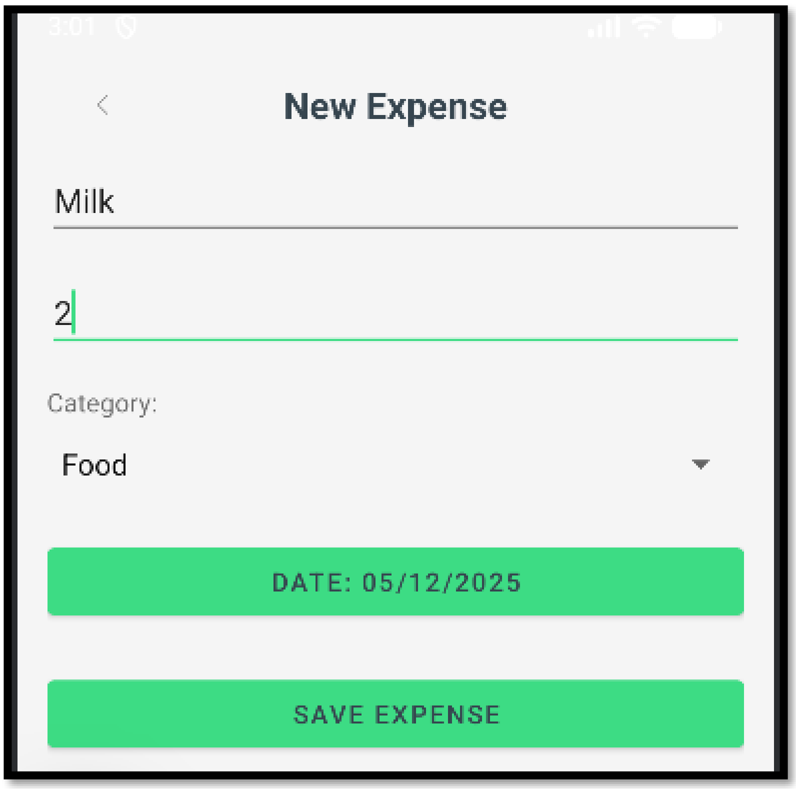
- Click the (+) button on the Dashboard or Layers screen.

- Enter the Description and Amount.

- Select a Category (Food, Transport, Rent, etc.) from the dropdown.

- Choose the Date (defaults to today).

- Click "Save Expense".



New Expense

Milk

2

Category:

Food

Figure 49: Add Expense Screen

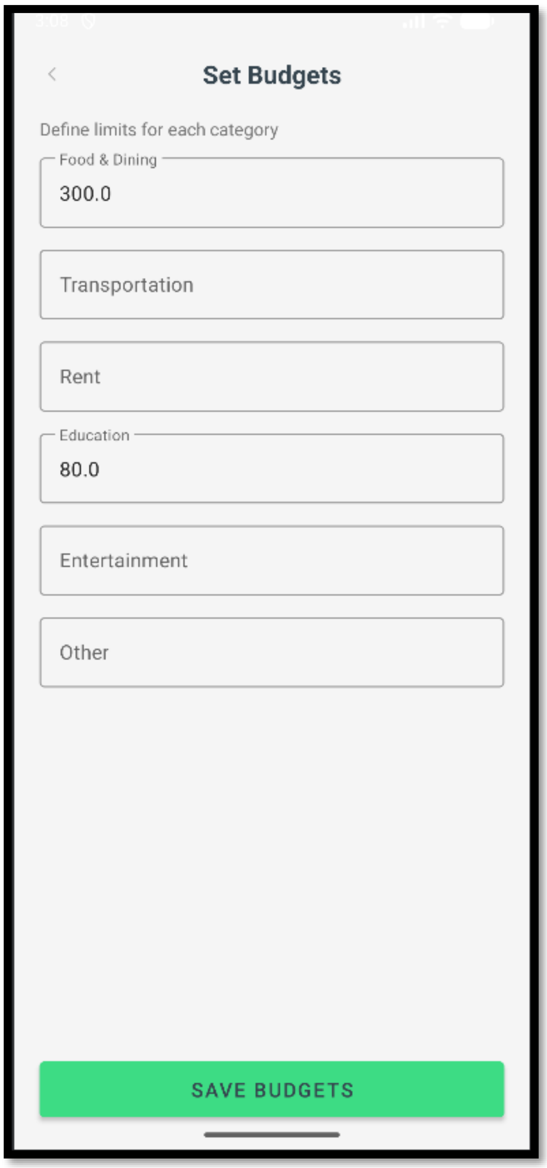
Budget Management

Users can set spending limits for specific categories to avoid overspending.

- Navigate to the Layers tab and select "Track Budget" (or use the button on the Dashboard).

- Enter the monthly limit for categories such as Food, Transportation, or Entertainment.

- Click "Save Budgets" to store the limits in the SQLite database.



Set Budgets

Define limits for each category

Food & Dining: 300.0

Transportation

Rent

Education: 80.0

Entertainment

Other

Figure 50: Budget Management

Financial Analysis and Alerts

The app provides two ways to analyze financial health:

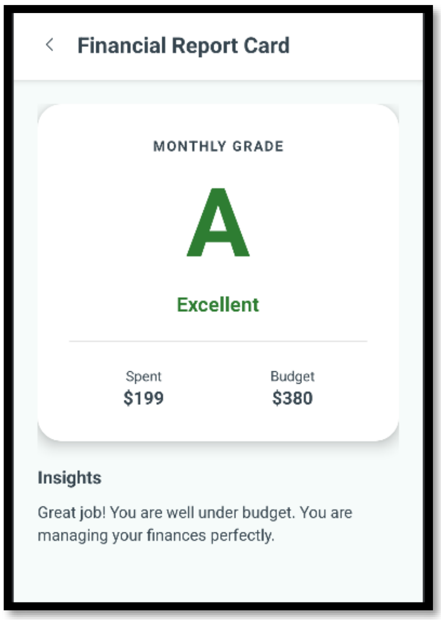
Category Analytics: Visualizes spending vs. budget using progress bars. The bars change color based on usage:

- Green: Safe (<80%).

- Amber: Caution (80% - 99%).

- Red: Over Budget (100%+).

Spending Grade: The "Spending Health" feature gives the user a grade (A, B, C, F) based on their total budget adherence.


Financial Report Card

MONTHLY GRADE

A

Excellent

Spent
$199

Budget
$380

Insights

Great job! You are well under budget. You are managing your finances perfectly.

Figure 51: Financial Report Card

Utilities (Notes and Calculator)

To support productivity, the app includes:

- Notes: A To-Do list feature where users can add tasks, mark them as complete (strikethrough effect), or delete them.

- Calculator**:** A built-in calculator for quick math operations without leaving the app.

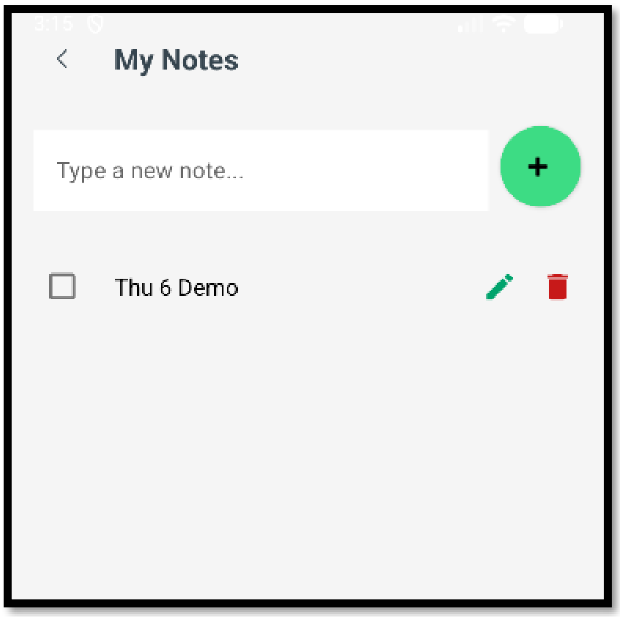


Figure 52: Notes and Calculator

- Go to the Layers tab and tap "Notes".

- Type a new task in the text box and tap "Add".

- To Complete: Tap the checkbox next to a note. The text will turn gray and have a strikethrough line.

- To Delete: Tap the trash icon next to the note.

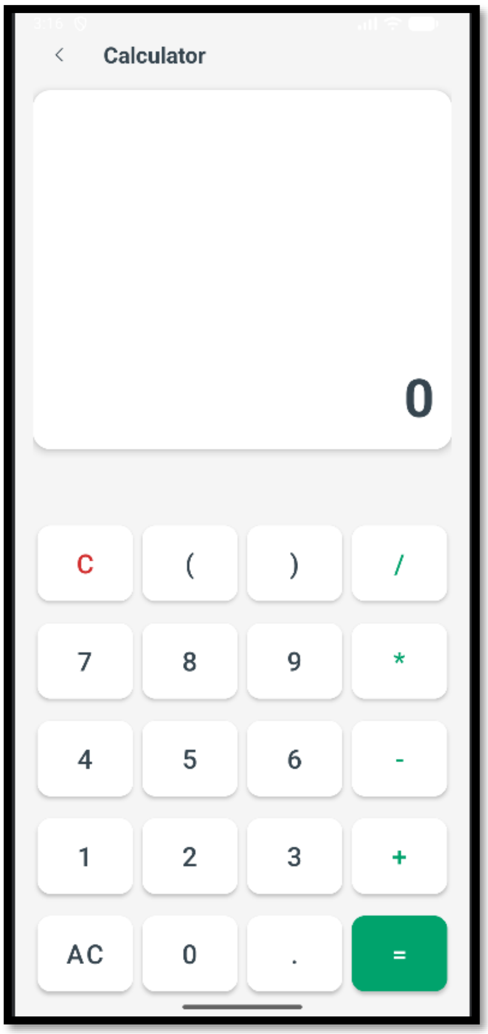


Figure 53: In-App Calculator Utility

- Go to the Layers tab and tap "Calculator".

- Use the number pad and operation buttons (+, -, *, /) to perform calculations.

- Tap "=" to see the result.

Settings and Profile

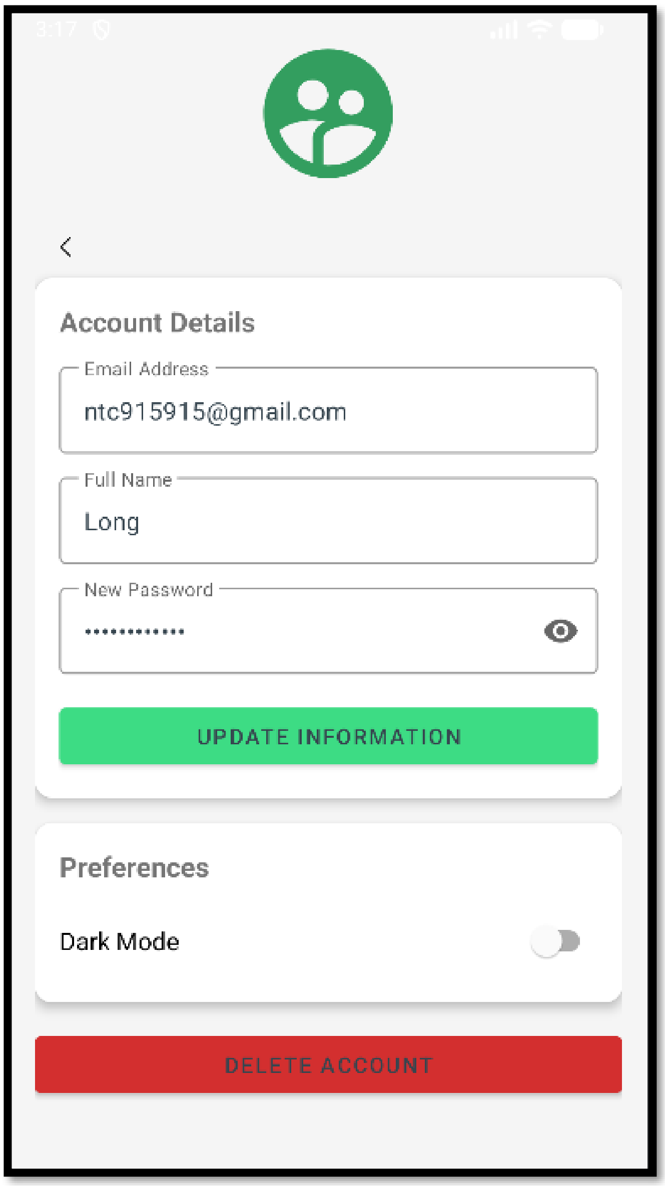
Users can manage their account via the Settings tab:

- Edit Profile: Update Full Name and Password.

- Dark Mode: Toggle the application theme between Light and Dark mode.

- Delete Account: Permanently remove the user account and all associated data from the database.

- Logout: Securely sign out and return to the Login screen.


Account Details

Email Address: ntc915915@gmail.com

Full Name: Long

New Password: ••••••••••••

Figure 54: Settings and Profile

Tap the Settings icon on the Bottom Navigation Bar.

Edit Profile: Tap "Edit Profile" to change your displayed Name or Password.

Dark Mode: Toggle the "Dark Mode" switch to change the app's appearance to a dark theme for better night viewing.

Logout: Tap "Logout" to end your session securely.

Delete Account: (Caution) Tap the Delete button and confirm the alert dialog to permanently erase your user data and expenses.

CHAPTER 4: LO4 EVALUATE THE PERFORMANCE OF A BUSINESS APPLICATION AGAINST ITS SOFTWARE DESIGN DOCUMENT AND INITIAL REQUIREMENTS.

9. (P6) Review the performance of the business application against the problem definition statement and initial requirements

9.4. Review against Functional Requirements

This section compares the final application functionality against the specific system requirements defined in Chapter 1

Requirement 1: Secure User Registration and Authentication

Initial Requirement: The system must provide secure user registration and authentication mechanisms (username/password) to protect user accounts

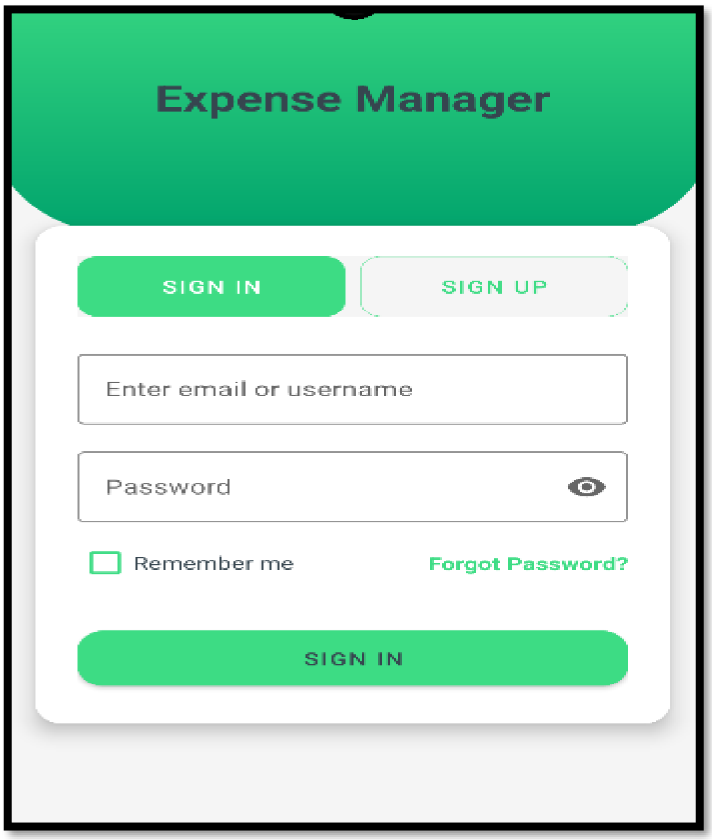


Figure 55: Secure User Registration and Authentication

Evaluation: Completed**.** The application successfully implements a registration and login system. Users can create accounts with validation (e.g., password length) and secure access. The "Remember Me" feature was added to enhance user convenience

Requirement 2: Expense Tracking (CRUD)

Initial Requirement: Users can add, edit, and categorize expenses. Each entry must include description, date, amount, and category

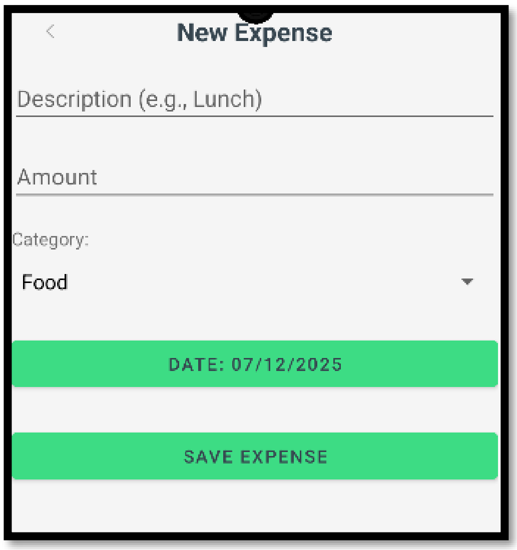
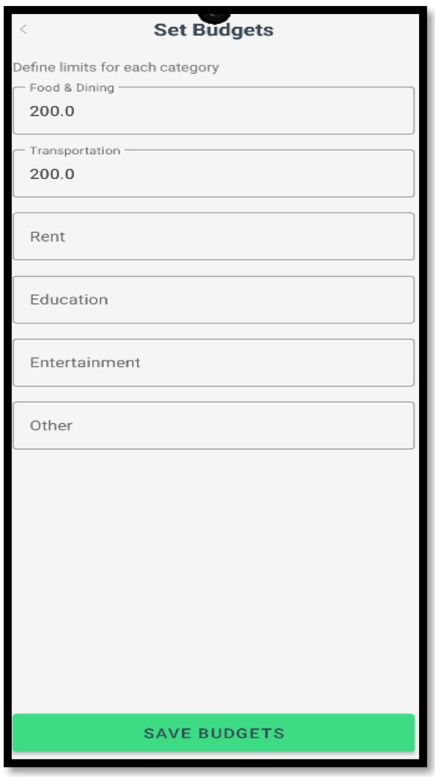


Figure 56: Expense Tracking (CRUD)

Evaluation: Completed**.** The core functionality works as expected. The "Add Expense" form captures all required fields (Amount, Category, Date, Note) and saves them to the SQLite database

Requirement 3: Budget Setting

Initial Requirement: Users can set and adjust monthly budgets for various expense categories (e.g., food, entertainment)



Set Budgets

Define limits for each category

Food & Dining	200.0
Transportation	200.0
Rent	
Education	
Entertainment	
Other	

Figure 57: Budget Setting

Evaluation: Completed. The app allows users to define specific limits for different categories. These budgets are saved and used to calculate the remaining balance shown on the dashboard

Requirement 4: Expense Overview and Visualization

Initial Requirement: The app should provide a summary of monthly expenses, remaining budget, and view expense trends over time

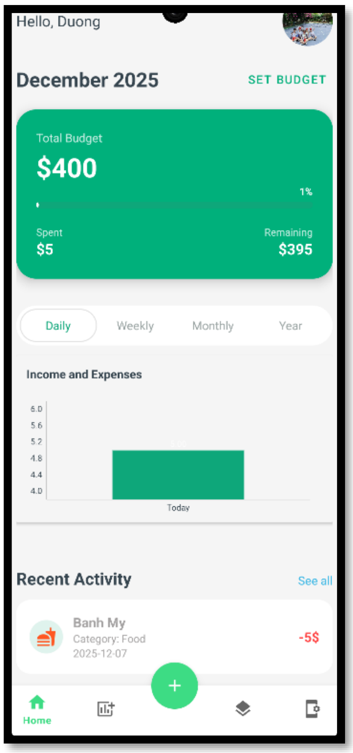


Figure 58: Expense Overview and Visualization

Evaluation: Completed. The Dashboard serves as a central hub displaying "Total Budget", "Spent", and "Remaining" in real-time. The MPAndroidChart library is successfully integrated to visualize spending trends (Daily/Weekly/Monthly)

Requirement 5: Recurring Expenses

Initial Requirement: The system must handle recurring expenses (e.g., monthly rent) by automatically adding them to the budget

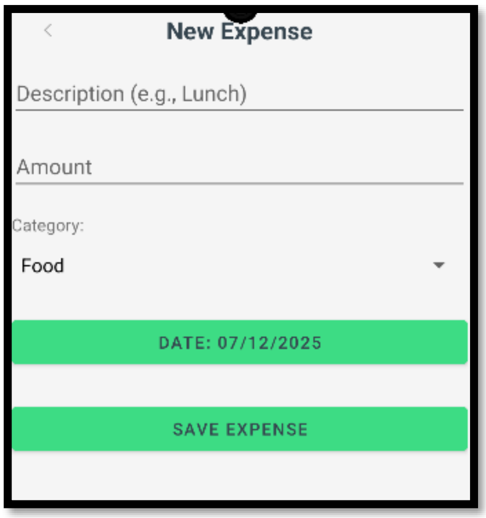


Figure 59: Recurring Expenses

Evaluation: Not Completed. The current "Add Expense" screen allows for single entries only. The logic for automated recurring payments was identified as a technical risk and was omitted due to the strict 12-week timeline constraints mentioned in the Risk Management plan

Requirement 6: Budget Alerts (Notifications)

Initial Requirement: The app should send reminders or notifications when users approach or exceed their budget limits

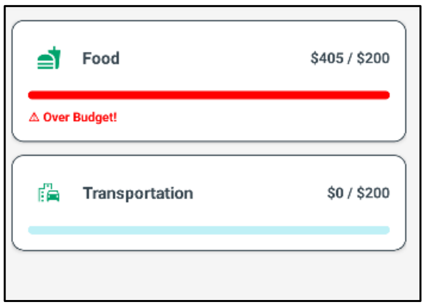


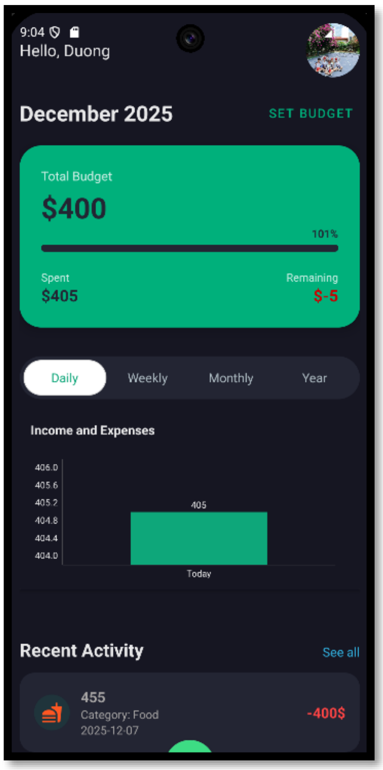
Figure 60: Budget Alerts

Evaluation: Modified / Partially Completed. Instead of intrusive push notifications, the team implemented "Visual Alerts". The category progress bars change color (Green to Amber to Red) to visually warn users when they exceed 80% or 100% of their budget

9.5. Review against Non-Functional Requirements

Requirement 7: Offline Functionality

Initial Requirement: The app must store and process data locally to ensure offline functionality.


9:04

Hello, Duong

December 2025

SET BUDGET

Total Budget

$400

101%

Spent $405

Remaining $-5

Daily | Weekly | Monthly | Year

Income and Expenses

Recent Activity

See all

455
Category: Food
2025-12-07
-400$

Figure 61: Offline Functionality

Evaluation: Completed. The application is built using SQLite as an embedded database. This ensures all features (CRUD, Budgeting) work perfectly without an internet connection

Requirement 8: Platform Compatibility

Initial Requirement: The app should be developed for both Android and iOS platforms

Evaluation: Not Completed / Scope Changed. Due to limited resources and the team's expertise being focused on Java/Android, the project scope was adjusted to "Native Android Application" only. iOS development is deferred to a future phase

Requirement 9: Feedback and Support

Initial Requirement: Include a feedback form within the app for users to report issues

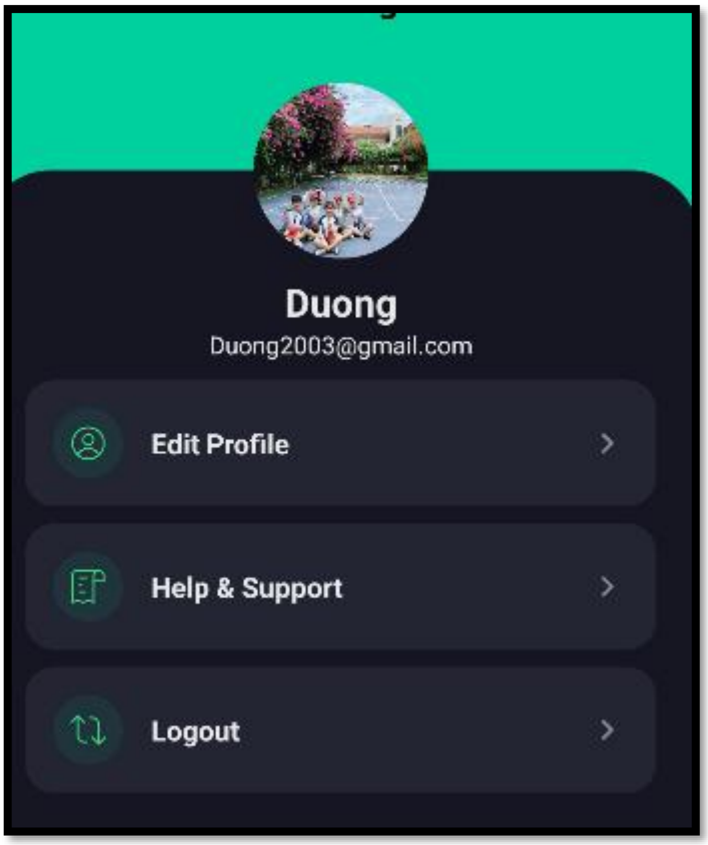


Figure 62: Feedback and Support

Evaluation: Partially Completed (UI Only). The "Help and Support" option is visible in the Settings/Profile menu. However, the backend integration for sending actual feedback reports is not yet fully operational in this prototype

Requirement 10: Performance and User Interface

Initial Requirement: The app should be responsive and user-friendly

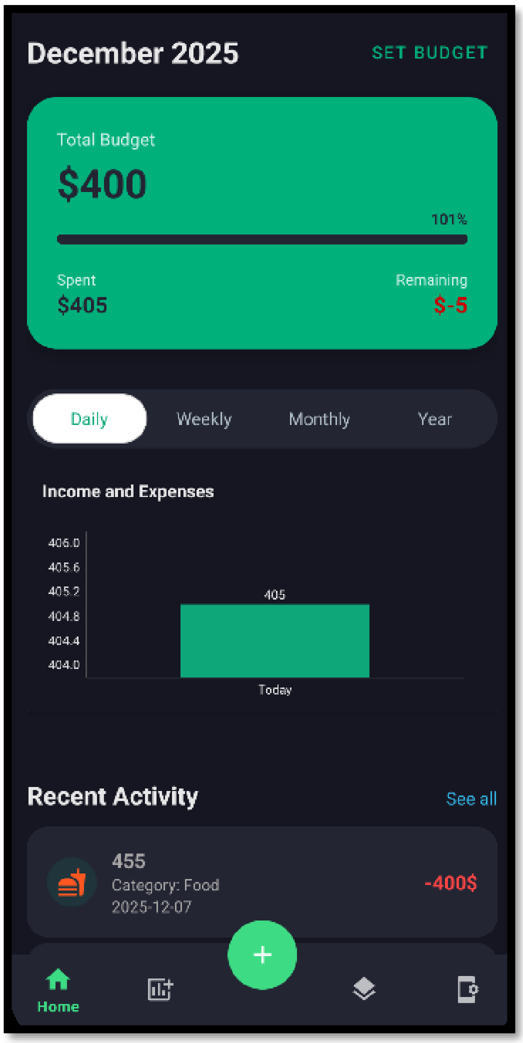


Figure 63: Performance and User Interface

Evaluation: Completed. The app uses standard Material Design components for a clean interface. Navigation between activities (Dashboard -> Add Expense -> History) is smooth and responsive on the tested emulators

9.6. Summary of Requirement Completion Table

Table 14: Summary of Requirement Completion Table

Requirement ID	Requirement Description	Completion Status	Notes / Justification
FR-01	User Registration and Authentication	Completed	Secure login and session management ("Remember Me") implemented

FR-02	Expense Tracking (CRUD)	Completed	Full Add/Edit/Delete functionality with categorization
FR-03	Budget Setting	Completed	Users can set limits for individual categories
FR-04	Expense Overview and Reports	Completed	Dashboard charts and Financial Report Card functional
FR-05	Recurring Expenses	Not completed	Omitted due to time constraints and backend complexity
FR-06	Budget Alerts	Modified	Implemented as Visual Alerts (Color-coded bars) instead of Push Notifications
NFR-01	Offline Functionality	Completed	Fully functional using SQLite local database
NFR-02	Cross-Platform (Android and iOS)	Not completed	Scope reduced to Android Native only due to resource limitations
NFR-03	Feedback Mechanism	Partially completed	UI button ("Help and Support") exists, but full backend logic is pending
NFR-04	Data Security	Partially completed	Authentication is secure; full database encryption is pending
Extra	Utilities (Notes and Calculator)	Added Value	Added features (To-Do List, Calculator) to enhance user productivity beyond initial scope

10. (M5) Critically review the design, development and testing stages of the application development process including risks.

The "BudgetWise Solutions" (CampusExpense Manager) application development project was executed within a strict fixed timeline of 12 weeks, with the primary goal of providing a robust offline expense management solution for university students. Although the final product achieved MVC (Model-View-Controller) status with stable core functionalities, the execution process revealed significant challenges regarding scope management and technical capability. This critical review analyzes the four distinct stages of the project, highlighting the specific risks encountered and the strategic solutions employed to mitigate them

Stage 1: Investigation

This phase focused on identifying user needs and defining the problem statement

- Activities: The team conducted a survey with 51 respondents and performed semi-structured interviews with 4 distinct user personas. The data showed that 80.4% of users clearly understood the core problem, but they had high expectations for automation and modern UI

- Risk identified: Scope Creep. Based on the initial survey, users requested advanced features such as "Recurring Payments," "Cloud Synchronization," and cross-platform support (iOS and Android). Given the team's profile (junior developers) and the short deadline, attempting to satisfy all these requirements posed a high risk of project failure due to resource overextension

- Mitigation Strategy: Prioritization (MoSCoW Method). The team immediately applied the MoSCoW prioritization technique. We decided to cut the scope significantly by dropping cross-platform development to focus exclusively on Native Android. Furthermore, we selected SQLite for local offline storage to eliminate the complexity of building and maintaining a Cloud Sync backend, pushing cloud features to "Could Have" (future scope)

Stage 2: Design

This stage involved defining the system architecture and selecting development tools

- Activities: The team constructed UML diagrams (Use Case, Class, ERD) and selected the Agile methodology over Waterfall to allow for flexibility. Android Studio and the MPAndroidChart library were chosen as the primary tools

- Risk identified: Technical Complexity and Security Implementation. During the ERD design for SQLite, the team identified a security risk regarding data privacy. Implementing full "Encryption at Rest" for the database required advanced knowledge of the Android Keystore system, which the team lacked. Additionally, the initial UI design did not account for "Dark Mode," a key requirement identified later in user feedback

- Mitigation Strategy: Simplified Security Model. To mitigate the technical risk while maintaining basic security, the team adopted a Password Hashing mechanism for user authentication instead of attempting full database encryption, which could have led to critical bugs. We also leveraged established third-party libraries (e.g., MPAndroidChart) to handle complex data visualization, reducing the likelihood of coding errors compared to building custom charts from scratch

Stage 3: Development

This was the execution phase where design was translated into code, revealing the most significant "Technical Debt."

- Activities: The team developed the Authentication, Expense CRUD, Budget Setting, and Reporting Dashboard modules

- Risk identified: Skill Gap and Time Constraints leading to Feature Drop. During implementation, the team struggled significantly with the logic for Recurring Expenses. Implementing a background service to automatically deduct funds monthly without draining the battery or crashing the app proved to be beyond the team's current technical capability within the remaining timeframe. This is the primary reason why requirement FR-05 was marked as "Not Completed."

- Mitigation Strategy: Strategic Descoping. Rather than delivering a buggy feature, the team made the critical decision to completely descope the Recurring Expenses feature for this release to ensure the Stability of the core functions. To compensate for this loss, the team developed additional "Utility" features such as Notes and a Calculator, which were technically feasible and added immediate value to the user experience

Stage 4: Testing

The final stage focused on verifying product quality

- Activities: Functional Testing and UI Testing were conducted using the Android Emulator (Pixel 9, API 36)

- Risk identified: User Experience (UX) Limitations. During testing, it was observed that "Push Notifications" for budget alerts were inconsistent across different Android versions due to OS battery optimization settings. This posed a risk that users might miss critical budget warnings

- Mitigation Strategy: Alternative Implementation (Visual Alerts). The team pivoted from system-level notifications to "Visual Alerts" within the application. We implemented color-coded Progress Bars (Green/Amber/Red) on the Dashboard. While less proactive than push notifications, this solution ensured consistency across all devices and guaranteed that users would visually perceive their financial status whenever they opened the app

In retrospect, the project successfully delivered a functional application that solves the core problem of offline expense tracking for students. However, the process provided a critical lesson in Technical Risk Management: there was a significant gap between the theoretical design (ambitions for iOS, Cloud Sync, Recurring Payments) and practical execution capabilities. The decision to adopt the Agile methodology was crucial, as it allowed the team to adaptively cut scope (descoping) and pivot to alternative solutions (Visual Alerts, Utilities) to meet the deadline. The final product, while lacking some initial "Nice-to-have" features, is a stable and usable MVC

CONCLUSION

In conclusion, the development of "BudgetWise Solutions" has successfully met the primary learning outcomes of Unit 22. The project delivered a functional, offline-capable mobile application that addresses the specific financial management needs of university students.

While the final product deviated from the initial scope by omitting Android & iOS support and recurring payments, these decisions were the result of effective risk management and critical prioritization. The application stands as a stable Model-View-Controller (MVC) with a robust MVC architecture, secure authentication, and valuable utility features like the integrated calculator.

Moving forward, the clear feedback from the peer review provides a roadmap for future iterations, specifically the implementation of Dark Mode and data export capabilities, transitioning the app from a student tool to a professional financial aid. The project demonstrates that even with limited resources and experience, a structured Agile approach can produce a viable business application.

EVALUATION

The evaluation of "BudgetWise Solutions" involves a critical review of the application's performance against the initial Software Design Document and an analysis of the development process itself.

Review against Functional Requirements The project successfully delivered the core "Must-Have" features defined in the problem statement.

Successes: The application successfully implements secure user authentication, CRUD (Create, Read, Update, Delete) capabilities for expenses, and real-time dashboard visualizations using the MPAndroidChart library. The mandatory requirement for "Offline Functionality" was fully met through the implementation of an embedded SQLite database, ensuring students can use the app regardless of connectivity.

Deviations: Two significant functional requirements were not fully met due to technical constraints. The "Recurring Expenses" feature was descoped because the logic for automated background deductions was too complex for the team's experience level within the 12-week timeline. Additionally, "Budget Alerts" were modified from system-level push notifications to in-app visual alerts (color-coded progress bars) to avoid OS battery optimization issues.

Review of the Development Process & Risk Management The adoption of the Agile methodology was the decisive factor in the project's completion. It allowed the team to manage the risk of "Limited Team Experience" by breaking the project into manageable sprints.

Scope Management: Initial ambitions for a cross-platform (iOS and Android) application with cloud synchronization were identified as high-risk scope creep. Using the MoSCoW prioritization method, the team strategically reduced the scope to a native Android application with local storage, ensuring a stable MVCcould be delivered on time.

Technical Execution: The use of the MVC architecture proved effective, as evidenced by the clear separation of code between the DatabaseHelper (Model), ChartHelper (Controller), and Activities (View). This structure improved code maintainability and debugging.

User Feedback Analysis Peer review data indicated that 80.4% of users clearly understood the problem definition. However, user feedback highlighted necessary future improvements, specifically the demand for "Dark Mode" to reduce eye strain and "Data Export" features (PDF/Excel) for professional use.

REFERENCES

Android Developers (2024). *Save data using SQLite*. [Online] Available at: https://developer.android.com/training/data-storage/sqlite [Accessed 05 Sep. 2025].

Android Developers (2024). *Meet Android Studio*. [Online] Available at: https://developer.android.com/studio/intro [Accessed 02 Sep. 2025].

Atlassian (2024). *What is Agile? | Atlassian*. [Online] Available at: https://www.atlassian.com/agile [Accessed 04 Sep. 2025].

Bloch, J. (2018). *Effective Java*. 3rd ed. Boston: Addison-Wesley.

Chacon, S. and Straub, B. (2014). *Pro Git*. 2nd ed. Apress.

Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, Mass.: Addison-Wesley.

Google Developers (2024). *Material Design Guidelines*. [Online] Available at: https://m3.material.io/ [Accessed 08 Sep. 2025].

Oracle (2024). *Java Platform, Standard Edition Documentation*. [Online] Available at: https://docs.oracle.com/en/java/ [Accessed 03 Sep. 2025].

Schwaber, K. and Sutherland, J. (2020). *The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game*. [Online] Available at: https://scrumguides.org/scrum-guide.html [Accessed 04 Sep. 2025].

SQLite (2025). *About SQLite*. [Online] Available at: https://www.sqlite.org/about.html [Accessed 05 Sep. 2025].

Sommerville, I. (2016). *Software Engineering*. 10th ed. Harlow: Pearson Education.

Trello (2025). *Trello Guide: How to use Trello*. [Online] Available at: https://trello.com/guide [Accessed 06 Sep. 2025].