System Verification and Validation Plan for Plutos

Team #10, Plutos
Payton Chan
Eric Chen
Fondson Lu
Jason Tan
Angela Wang

March 10, 2025

Revision History

Table 1: Revision History

Date	Version	Notes
11/01/2024	0.1	Add: 3.1, 3.2, 3.3, 3.6
11/02/2024	0.2	Update Rev0 draft
11/03/2024	0.3	Add 4.1 for Rev0
...	...	...

Contents

1 Symbols, Abbreviations, and Acronyms

Refer to Section 1.3 of the Software Requirements Specification (SRS) document for the list of symbols, abbreviations, and acronyms.

In addition, the following abbreviations are used in this document:

Table 2: Symbols, Abbreviations, and Acronyms

symbol	**description**
V&V	Verification and Validation
UI	User Interface
OCR	Optical Character Recognition
SQL	Structured Query Language
GDPR	General Data Protection Regulation

This document will provide a detailed plan for the Verification and Validation (V&V) of Plutos. Below is an overview of what will be covered in this document:

- 2 General Information: This section will provide a brief overview of the software being tested, its objectives, challenge level, and extras.
- 3 Plan: This section will outline the V&V team roles, the delegation of roles to team members, and the V&V plan.
- 4 System Tests: This section will detail the system tests for functional and nonfunctional requirements, and traceability between test cases and requirements.
- 5 Unit Test Description: This section will provide an overview of the unit testing scope, unit tests for functional and nonfunctional requirements, and traceability between unit tests and modules.

2 General Information

This section will outline the summary of the V&V document, the objectives of the process, the challenge level and extras of the project, and the relevant external documentation that will be referenced in this document.

2.1 Summary

This V&V document outlines the testing approach for Plutos to ensure the application meets the functional and non-functional requirements outlined in the SRS document. The verification process will confirm the functionality of components such as receipt scanning, item recognition, and categorization function as intended, while validation efforts will focus on ensuring usability, performance, and security standards are met for an optimal user experience.

2.2 Objectives

This subsection will provide an overview of the V&V objectives that are in-scope and out-of-scope for the Plutos application.

2.2.1 In-Scope V&V Objectives

The following are the V&V objectives, outlining what is intended to be verified or validated as part of the V&V process:

- **Accurate Data Extraction and Categorization**: Ensure that the machine learning (ML) model accurately extracts key information from receipts and correctly categorizes it by expense type. This is critical to providing users with precise budgeting insights. The accuracy requirements are denoted in the Software Requirements Specification (SRS).
- **Seamless User Experience**: Prioritize usability by designing an intuitive and responsive interface that simplifies the process of scanning and categorizing receipts. The app should be easy to navigate, allowing users to view spending summaries effortlessly.
- **Real-Time Budgeting Feedback** Provide timely updates on spending habits to make users aware of their financial status. This feature will empower users to make informed spending decisions based on real-time data.

- **Security and Privacy** Securely handle all user data, with particular attention to sensitive financial information. This objective includes encrypting stored data and ensuring compliance with relevant data protection regulations.

2.2.2 Out-of-Scope V&V Objectives

The following are the V&V objectives that are out-of-scope due to time or resource constraints:

- **In-Depth Usability Testing with All User Types** While basic usability will be validated, comprehensive testing with diverse user demographics and accessibility concerns is out of scope due to resource limitations. Feedback gathered post-launch may help inform future usability improvements.
- **Verification of External Libraries** The application will use external libraries for image processing and data handling, assuming these libraries have been rigorously tested by their developers. We will prioritize testing our AI model and app functionality rather than the underlying libraries to focus resources on core functionality.
- **Accuracy for Poor-Quality Receipts and Non-Common Items**: Currently, the receipt scanner focuses on high-quality receipts containing commonly purchased items for students. While expanding the application to support lower-quality receipts and less common items is planned for future phases, this enhancement is out of scope for now due to the complexity of training the ML model during our timeline.

2.3 Challenge Level and Extras

The challenge level for the project is **general**. The extras that will be included are:

- **Requirements elicitation**: Surveys and interviews have been conducted as part of developing our Software Requriements Specfication (SRS). See Requirements Elicitation Report for more details.
- **Usability testing**: As part of our validation plan, we will conduct usability testing in which users will interact with the Plutos application and evaluate their experience.

2.4 Relevant Documentation

The following documents are relevant to the V&V of Plutos:

- **Problem Statement and Goals document**: This document outlines the problem statement and goals of the Plutos application, which will considered to ensure the V&V process aligns with the project's purpose and objectives.
- **Software Requirements Specification (SRS) document**: This document outlines the functional and non-functional requirements of the Plutos application, serving as the basis for all testing activities.
- **Hazard Analysis document**: This document outlines potential hazards and risks associated with the Plutos application, which will be considered during V&V to ensure effective risk mitigation.
- **Module Interface Specification (MIS) document**: This document provides a detailed design of the Plutos application, which will be used to inform the development of unit tests for each module. *Note: This document is yet to be completed as of the time of writing this initial V&V plan.*
- **Module Guide (MG) document**: This document provides a detailed description of the software architecture and modules of Plutos, which will guide the development of unit tests for each module. *Note: This document is yet to be completed as of the time of writing this initial V&V plan.*

3 Plan

This section introduce the team members involved in the V&V process and describe the V&V plan for the following components:

- The SRS
- The design
- The V&V
- The implementation

Following this, there will be a subssections detailing the automated testing and verification tools, as well as the software validation plan.

3.1 Verification and Validation Team

There will be a total of 7 distinct roles when it comes to our verification and validation team. These roles will be split amongst team members and members may share the same secondary roles to facilitate a broader range of tests without overwhelming one person. This division allows each team member to specialize in a certain aspect of our app when it comes to verification and validation. Attached below is a list of roles available in our V&V group along with the descriptions of what each role entails.

V&V Roles

- Test Case Designer
  - Develops detailed test cases for functional requirements, focusing on different aspects such as image quality, data parsing accuracy, and categorization correctness.
  - Works closely with the other developers to understand key components of the app and to ensure that test cases cover all scenarios, including edge cases.
- Quality Assurance Engineer
  - Ensures overall quality of the app by performing functional, integration, and system tests.
  - Coordinates end-to-end testing for receipt scanning and expense categorization workflows.
  - Collaborates with the Test Case Designer to verify that all test cases have been executed and outcomes meet expected results.
- Ai Model Validator
  - Focuses on testing the AI model specifically, evaluating its accuracy in parsing text and categorizing expenses.
  - Conducts model performance assessments under various conditions, such as diverse receipt layouts, lighting conditions, and languages if applicable.
  - Continuously monitors for model drift and helps recalibrate the model if accuracy degrades over time.

- Usability Tester
  - Evaluates the app's user interface and user experience for intuitiveness and ease of use.
  - Conducts usability testing sessions with other students or users in the target demographic (e.g., university students).
  - Provides feedback on the app's functionality and ensures that user feedback is incorporated into iterative improvements.
- Data Integrity Specialist
  - Ensures the accuracy and security of the data collected, especially sensitive information such as financial data on receipts.
  - Verifies that the app correctly anonymizes data if required and that categorization matches expected outputs based on provided datasets.
  - Works closely with the AI Model Validator to confirm data handling complies with privacy regulations and standards.
- Automation Engineer
  - Develops automated testing scripts to streamline regression testing and ensure the app functions consistently across different updates.
  - Automates repetitive tests, especially those related to scanning, categorization, and UI testing, to save time during development sprints.
  - Coordinates with the Test Case Designer to convert key test cases into automated tests for efficient reuse.
- Performance Tester
  - Measures and optimizes app's performance, focusing on response time, load time, and battery usage (important for a mobile app).
  - Conducts stress tests to determine the app's behavior under heavy usage (e.g., scanning multiple receipts in a short period).
  - Collaborates with developers to troubleshoot and resolve performance bottlenecks.

The following section delegates the roles above to each team member. Note that each team member has a secondary role which may be repeated amongst other members. The secondary role is to provide additional assistance when it comes to verification and validation and also serves to bring additional confidence that the system is working as it should.

Delegation of V&V Roles

Eric Chen

- Role(s)
    1. Test Case Designer
    2. Quality Assurance Engineer
- Responsibilities
    1. Creates comprehensive test cases for functional requirements.
    2. Executes tests to ensure all functionalities are covered.

Angela Wang

- Role(s)
    1. AI Model Validator
    2. Data Integrity Specialist
- Responsibilities
    1. Tests the AI model's accuracy in parsing receipts.
    2. Ensures data handling complies with security and privacy standards.

Fondson Lu

- Role(s)
    1. Usability Tester
    2. Quality Assurance Engineer
- Responsibilities
    1. Conducts usability tests and collects user feedback.
    2. Supports quality assurance by testing overall workflows and integration.

Jason Tan

- Role(s)
    1. Automation Engineer
    2. Performance Tester
- Responsibilities
    1. Develops automated test scripts for repetitive testing.
    2. Measures app performance under different conditions to optimize speed and efficiency.

Payton Chan

- Role(s)
    1. Quality Assurance Engineer
    2. Performance Tester
- Responsibilities
    1. Coordinates end-to-end testing and verifies the app's functionality.
    2. Focuses on maintaining consistent performance across updates.

3.2 SRS Verification Plan

3.2.1 Approaches for SRS Verifiation

- Peer Review Feedback:
  - Each team member will review the SRS individually and provide ad hoc feedback. They will focus on identifying ambiguous language, incomplete requirements, and inconsistencies.
  - Each reviewer will log feedback in a shared document, categorizing issues by priority (e.g., high, medium, low).
- Primary Reviewer Session:
  - A primary reviewer from the team, ideally someone with experience in requirements verification, will perform a detailed examination. They will document findings, focusing on critical areas such as requirement clarity, feasibility, and completeness.
- External Peer Feedback:
  - If possible, solicit feedback from classmates or others familiar with requirements engineering. They can offer objective insights and identify issues that may have been overlooked internally.

3.2.2 Structured Internal Review Process

- Initial Team Meeting:
  - Hold a structured review meeting where team members present and discuss major sections of the SRS. Key points of focus should include functional requirements, assumptions, and constraints.
  - Prepare a set of questions to guide the discussion, such as:
    - "Are the requirements unambiguous and complete?"
    - "Is each requirement feasible and realistic?"
    - "Do the requirements adequately represent the user's needs?"
    - "Are there any areas that might be challenging to test or verify?"
- Task-Based Inspection:
  - Team members can take turns as "inspectors," responsible for a task-based evaluation of key areas:
    - Ensuring requirements align with project goals.
    - Verifying each requirement's testability and clarity.
    - Checking that the document is organized and easy to navigate.
- Issue Tracker:
  - Use an issue tracker or a collaborative tool to manage findings and resolutions. Each issue should include:
    - A summary of the issue.
    - Priority level (e.g., high, medium, low).
    - Suggested resolution steps.
    - A status field for tracking progress until resolved.

3.2.3 SRS Quality Checklist for Verification

Table 3: SRS Quality Checklist

Checklist Item	**Description**	**Pass/Fail**
Requirements Clarity	All requirements are clear, unambiguous, and written in active voice.	
Completeness	The SRS includes all functional and non-functional requirements, assumptions, and constraints.	
Consistency	Requirements are consistent across the document, with no contradictory statements.	
Testability	Each requirement is formulated so that it can be tested through measurable criteria.	
Feasibility	All requirements are realistic and achievable within the project's scope and resources.	
Traceability	Each requirement is traceable to a higher-level objective or user need.	
Maintainability	The document is well-organized, with clear headings and numbered requirements for easy referencing.	
Priority and Dependencies	Requirements are prioritized, and dependencies are explicitly stated where applicable.	
User-Centered Language	Requirements reflect user needs and use terminology consistent with user and stakeholder language.	
Compliance with Standards	The SRS complies with any applicable standards or best practices in software requirements engineering.	

3.3 Design Verification Plan

3.3.1 Specification Verification

Table 4: Specification Verification Checklist

Checklist Items	**Description**	**Verification Method**	**Pass /Fail**	**Comments**
Requirements Completeness	Ensure all system and user requirements are addressed in the design.	Requirements Traceability Matrix		
System Requirements Compliance	Verify that design conforms to software/system specifications.	Specification Review		
Compliance with Standards	Ensure compliance with coding standards, industry best practices, and regulatory guidelines.	Code Review, Standards Checklist		
Security Requirements	Confirm that the security design follows the requirements, especially for data privacy and user information.	Security Audit		

3.3.2 Functional Verification

Table 5: Functional Verification Checklist

Checklist Items	**Description**	**Verification Method**	**Pass /Fail**	**Comments**
Receipt Scanning Capability	Verify that receipt scanning feature works as expected with various image qualities.	Functional Testing		
Expense Categorization Accuracy	Confirm that expenses are correctly categorized by the AI model.	Model Accuracy Testing		
User Interface (UI) Responsiveness	Ensure that UI elements are responsive across devices and screen sizes.	UI Testing		
Error Handling and Recovery	Verify that the app gracefully handles errors, such as scanning failures or incorrect categorizations.	Error Simulation		
Integration with External Services	Confirm integration with any third-party services (e.g., cloud storage or payment platforms).	Integration Testing		

3.3.3 Performance Validation

Table 6: Performance Validation Checklist

Checklist Items	**Description**	**Verification Method**	**Pass /Fail**	**Comments**
System Performance Under Load	Test app performance with simultaneous receipt scans to assess system stability and responsiveness.	Load Testing		
AI Model Processing Time	Measure the average processing time for receipt categorization.	Model Timing Analysis		
Battery and Resource Consumption	Verify that app resource usage is within acceptable limits to preserve device performance and battery life.	Resource Monitoring		
Network Efficiency	Ensure the app handles network limitations (e.g., slow or intermittent connections) without data loss.	Network Simulation		
Latency for Real-Time Features	Measure latency in real-time features, ensuring a smooth user experience.	Latency Testing		

3.3.4 Document Validation

Table 7: Document Validation Checklist

Checklist Items	**Description**	**Verification Method**	**Pass /Fail**	**Comments**
User Documentation Completeness	Verify that user manuals, installation guides, and support documentation are complete and accurate.	Documentation Review		
Developer Documentation Completeness	Ensure all developer-focused documentation (e.g., API docs, setup instructions) is detailed and up to date.	Documentation Review		
Code Documentation	Confirm that all code modules and functions are documented according to guidelines.	Code Review		
Test Plans and Results	Verify the completeness of test plans and that test results meet required standards.	Test Report Review		
Change Log and Version History	Ensure that the change log and version history are comprehensive and well-documented.	Change Log Review		

3.4 Verification and Validation Plan Verification Plan

Creating a Verification and Validation (V&V) plan for a smart AI budgeting app requires a structured and systematic approach to confirm that the application operates accurately, efficiently, and reliably under real-world conditions. This app uses advanced AI to parse receipt images, extract relevant information, and categorize expenses, which presents additional challenges for the V&V process. Unlike traditional software applications, an AI-driven app requires validation not only of standard software functionalities but also of the AI model's performance in accurately interpreting various receipt formats, layouts, and data entries. This means that the V&V plan must address both classic software testing elements (such as unit and integration testing) and specific AI model evaluation techniques, like model validation and performance testing, to ensure that the app delivers consistent results.

Given the diverse functionality—spanning expense tracking, budgeting, and financial forecasting—this V&V approach must encompass both software-level testing and user-centric assessments to verify usability and accuracy from an end-user perspective. Below is a comprehensive list of recommended V&V techniques and a corresponding checklist, each tailored to ensure that the budgeting app meets its design specifications and delivers high-quality, reliable outcomes for users.

3.4.1 V&V Techniques for Plutos App

1. **Requirements Review**: Regular reviews of the Software Requirements Specification (SRS) are essential to confirm that all listed requirements align with the project goals. Stakeholders should validate that these requirements meet user needs.

2. **Code Review**: Structured code reviews help to maintain code quality and identify potential issues in the receipt parsing and expense categorization components.

3. **Unit Testing**:Unit tests should cover the core functions, especially those related to parsing and categorizing. Test cases should address a range of inputs, including expected, edge, and invalid cases.

4. **Integration Testing**: Integration testing ensures that the AI model and app components work together seamlessly, verifying the accuracy and consistency of data processing.

5. **System Testing**: System testing is necessary to evaluate the full app functionality, making sure that expense tracking, budgeting, and forecasting features meet the specified requirements.

6. **Acceptance Testing**: End-user testing allows the team to validate that the app performs well in real-world settings, especially in accurately parsing receipts of various formats.

7. **Mutation Testing**: Mutation testing can be used to check the effectiveness of the test cases and to identify any weaknesses in the testing approach.

8. **Regression Testing**: Regression testing is essential to ensure that updates or changes in the AI model or app features do not interfere with existing functionality.

9. **User Interface (UI) Testing**: UI testing helps confirm that the app interface is user-friendly, responsive, and consistent across different devices.

10. **Model Validation**: The AI model should be validated on labeled data to check its performance, especially for accuracy in categorizing expenses based on parsed information.

11. **Performance Testing**: Performance tests can be conducted to measure the app's speed and reliability, especially when processing large receipts or handling multiple users.

12. **Security Testing**: Security testing ensures that financial data is stored and transmitted securely, with protections in place for data encryption and user access.

3.4.2 V&V Plan Checklist (Peer Reviews)

- [ ] Review and validate all requirements with stakeholders.
- [ ] Conduct code reviews and log issues or recommendations.
- [ ] Develop and execute unit tests covering key parsing and categorization functions.
- [ ] Perform integration testing for model interactions with the app's front and back end.
- [ ] Complete system testing across all app functionalities (tracking, budgeting, forecasting).
- [ ] Facilitate acceptance testing sessions with end users to gather real-world feedback.
- [ ] Implement mutation testing to strengthen the test suite.
- [ ] Schedule regular regression testing, especially after updates or model retraining.
- [ ] Conduct comprehensive UI testing for design consistency across devices.
- [ ] Evaluate AI model performance, retraining if accuracy is below target.
- [ ] Measure and optimize performance for speed and reliability under heavy usage.
- [ ] Perform security testing to protect user financial data.
- [ ] Document and address all V&V findings and updates to maintain a high-quality app.

3.5 Implementation Verification Plan

3.5.1 Verification Techniques

1. **Code Walkthroughs**:
    - **Purpose**:Collaborative sessions where developers explain their code to peers.
    - **Focus Area(s)**: Core functions like receipt parsing, expense categorization, and AI integration.
    - **Goal**: Increase team familiarity with code components and detect early logical or design errors.
2. **Code Inspections**:
    - **Purpose**: Formal, structured code reviews based on a project-specific checklist.
    - **Checklist Criteria**:
        - Adherence to coding standards and proper error handling.
        - Efficient use of resources (especially for image processing and AI tasks).
        - Consistent structure, naming conventions, and formatting.
    - **Goal**: Enforce best practices, maintain code readability, and ensure alignment with functional requirements.
3. **Static Analysis**:
    - **Tools**: ESLint, Prettier
    - **Checks Performed**:
        - Syntax errors, unused code, and code smells.
        - Security vulnerabilities, especially in data handling and storage.
        - Code complexity, duplication, and performance.
        - Maintains a uniform code style so that codebase becomes more readable, navigable and manageable.
    - **Goal**: Identify issues automatically, maintain security, and ensure high code quality.

4. **Unit Testing**:

    - **Purpose**: Verify that individual functions and components operate correctly in isolation.
    - **Key Functions to Test**: Data extraction from receipts, categorization logic, and budget calculations.
    - **Coverage Goal**: Achieve ¿80% coverage on core functions.
    - **Goal**: Ensure reliability and accuracy at the component level.

5. **Integration Testing**:

    - **Purpose**: Test interactions between modules to ensure data flows as expected.
    - **Key Interactions to Verify**: Data flow between the AI model, backend, and frontend UI.
    - **Goal**: Ensure seamless integration of AI functionality with other app components.

6. **Regression Testing**:

    - **Purpose**: Prevent new updates from disrupting existing features.
    - **Scope**: Entire application with emphasis on areas impacted by recent changes or updates.
    - **Process**: Continuously updated regression suite covering core features and AI updates.
    - **Goal**: Maintain stability and functionality across iterative changes.

3.6 Automated Testing and Verification Tools

1. **Unit Testing Frameworks**

   - **Jest**: Jest is the default testing framework for React Native applications. It supports TypeScript, has a rich API for writing unit tests, and comes with built-in mocking capabilities.
   - **React Testing Library**: This library works well with Jest for testing React components, focusing on user interactions and component behavior rather than implementation details.

2. **Static Analysis Tools**

   - **ESLint**: ESLint is a powerful linter for JavaScript and TypeScript. It can be configured with TypeScript support to enforce coding standards, detect potential errors, and maintain code quality. It can also be extended with plugins for React and serves as industry best practices for JavaScript/TypeScript related-projects.
   - **Prettier**: Integrating Prettier alongside ESLint helps ensure consistent code formatting across the codebase, which helps in improving readability.

3. **Continuous Integration (CI) Tools**

   - **GitHub Actions**: This tool can be set up to run workflows that execute tests, linters, and builds automatically on every pull request or commit, ensuring that code quality is maintained consistently (i.e. Husky).
   - **Jenkins**: Jenkins is an open-source automation server that facilitates continuous integration and continuous deployment processes. It can be configured to automatically run tests, linting, and builds every time code is pushed to the repository, ensuring that code quality is maintained consistently.

4. **Test Coverage Tools**

    - **Istanbul (nyc)**: Istanbul is a code coverage tool that integrates with Jest, providing detailed reports on which parts of the codebase are covered by tests, helping to identify untested code.
    - **Codecov or Coveralls**: These services can be used to visualize and summarize code coverage metrics after each build, providing insights into overall coverage and trends over time.

5. **Profiling Tools**

    - **React Native Performance Monitor:**: This built-in tool in React Native helps track performance metrics such as frame rates and memory usage, providing insights into the app's performance.
    - **Flipper**: A platform for debugging mobile apps, Flipper offers a variety of plugins that assist in profiling and monitoring app performance.

6. **Mutation Testing**

    - **Stryker**: Stryker is a mutation testing framework for JavaScript and TypeScript. It can help assess the effectiveness of tests by introducing small changes in the code (mutations) and checking if the tests can catch them.

7. **Plans for Summarizing Code Coverage Metrics**

    - **Daily/Weekly Reports:**: The CI pipeline can be configured to generate code coverage reports with every build and aggregate these reports weekly. Tools like Codecov can be used to visualize trends over time.
    - **Review Meetings**: Regular review meetings can be conducted to discuss coverage metrics with the team, identifying areas needing improvement and setting goals for coverage percentages.
    - **Integrate Coverage Reports in PRs**: Coverage reports should be included in pull request reviews, ensuring that any code changes are evaluated in the context of their impact on overall coverage.

For additional information, please consult the development plan.

3.7 Software Validation Plan

1. **Objective**

   - The objective of this software validation plan is to ensure that the Plutos budgeting app meets all specified requirements and user needs for accurately parsing receipts, categorizing expenses, and generating reliable financial insights. The plan aims to validate the app's AI-driven capabilities, functional accuracy, security, and overall user experience.

2. **Scope**
   This validation plan will cover the following main functionalities:

   - **Receipt Parsing and Data Extraction**: Ensuring accurate interpretation of various receipt formats.
   - **Expense Categorization**: Validating that extracted data is correctly categorized.
   - **Expense Tracking and Budgeting**: Verifying that users can effectively track and budget expenses.
   - **Forecasting Capabilities**: Ensuring the app provides meaningful insights for future spending.
   - **User Interface (UI) and User Experience (UX)**: Confirming ease of use, responsiveness, and accessibility across devices.
   - **Data Security and Privacy**: Verifying secure data handling, storage, and compliance with privacy standards.

3. **Validation Techniques** The validation process will employ the following techniques to confirm that the software performs as intended and fulfills user requirements.

   (a) **Requirements Validation**

   - **Purpose**: Ensure that the system requirements, as outlined in the Software Requirements Specification (SRS), reflect user needs.
   - **Techniques**: Requirements review, stakeholder validation sessions.
   - **Criteria**: All requirements are verified as achievable, necessary, and aligned with end-user expectations.

   (b) **Functional Testing**

   - **Purpose**: Validate that the app functions correctly and meets the intended use cases.
   - **Techniques**:
     - **Unit Testing**: Test each component in isolation (e.g., receipt image processing, data extraction modules).
     - **Integration Testing**: Ensure smooth interaction between AI models and application components.
     - **System Testing**: Test the app end-to-end, covering all functional requirements.
   - **Criteria**: All functional test cases pass, with particular attention to accuracy in parsing receipts and categorizing expenses.

   (c) **Model Validation and AI Performance Testing**

   - **Purpose**: Confirm that the AI model for receipt parsing and categorization meets performance expectations.
   - **Techniques**:
     - **Model Evaluation on Labeled Data**: Test the model's accuracy on a labeled dataset of diverse receipts.
     - **Performance Benchmarking**: Measure model accuracy, precision, recall, and latency.
     - **Error Analysis**: Review misclassifications and parsing errors to refine the model.

- **Criteria**: Model meets or exceeds the required performance metrics (e.g., 80%+ accuracy in parsing and categorization).

(d) **Usability Testing**

- **Purpose**: Validate that users can easily interact with the app and achieve their goals without confusion or frustration.
- **Techniques**:
  - **User Surveys and Feedback Sessions**: Gather direct user feedback on the app's ease of use and intuitiveness.
  - **Task-Based Testing**: Have users complete key tasks (e.g., uploading a receipt, viewing categorized expenses).
- **Criteria**: Positive user feedback, with 80%+ of participants finding the app intuitive and easy to use.

(e) **Security and Privacy Testing**

- **Purpose**: Ensure that user data, especially financial information, is stored and handled securely.
- **Techniques**:
  - **Data Encryption Verification**: Confirm that data is encrypted both in transit and at rest.
  - **Access Control Testing**: Validate that only authorized users can access their data.
  - **Compliance Testing**: Ensure adherence to relevant privacy regulations (e.g., GDPR).
- **Criteria**: All data protection and compliance measures are met; no security vulnerabilities found.

(f) **Performance and Load Testing**

- **Purpose**: Verify the app's responsiveness and stability under expected usage.
- **Techniques**:
  - **Load Testing**: Simulate high user load to evaluate system performance.
  - **Stress Testing**: Test the app's behavior under extreme conditions.
- **Criteria**: App remains functional and responsive under high load, with minimal performance degradation.

(g) **Regression Testing**

- **Purpose**: Ensure that new updates or fixes do not break existing functionality.
- **Techniques**: Automated regression test suite that re-tests core functionality after each update.
- **Criteria**:All regression test cases pass, maintaining stability with no loss of functionality.

4. **Validation Schedule**

Table 8: Validation Schedule

Phase	**Duration**	**Activities**
Requirements Review	1 week	Validate SRS, confirm requirements with stakeholders
Functional Testing	2 weeks	Develop and execute unit, integration, and system test cases
Model Validation	1 week	Evaluate AI model on diverse receipts, measure performance metrics
Usability Testing	1 week	Conduct usability tests with end-users, gather feedback
Security Testing	1 week	Perform security assessments, validate data encryption and compliance
Performance Testing	3 days	Conduct load and stress tests to evaluate system performance under expected load
Regression Testing	Ongoing	Run regression tests after updates to ensure consistent functionality

5. **Deliverables**

   - **Test Case Documentation**: Detailed test cases for all functionalities, covering expected outcomes and pass/fail criteria.
   - **Test Execution Report**: Summary of test results, including success rates and any failed test cases.
   - **Defect and Issue Log**: Record of identified defects and issues, with resolution status.
   - **Usability Feedback Summary**: Report on feedback from usability testing sessions.
   - **Validation Summary Report**: Comprehensive report summarizing the validation outcomes, certifying the app's readiness for release.

6. **Success Criteria**
   The app will be considered validated if it meets the following success criteria:

   - All functional test cases passes requirements.
   - AI model achieves required accuracy in parsing and categorization.
   - Usability tests show positive user feedback with minimal issues.
   - No high-severity defects in security, performance, or compliance.
   - Regression tests show that new updates do not disrupt existing functionality.

For additional information regarding validation, please consult the Software Requirements Specification (SRS) verification section.

4 System Tests

4.1 Tests for Functional Requirements

The following test cases are divided into subsets corresponding to major functional areas in the application, as detailed in the Software Requirements Specification (SRS) document. Each subset addresses a specific functional component, including user account management, receipt scanning and processing, database management, item recognition and categorization, and financial tracking. These tests will help verify the functional requirements outlined in the SRS and will help with the creation of those tests to verify the specified functional requirements.

4.1.1 User Account Management Tests

This subsection covers tests required to verify tests revolving around user account management. The user account manager allows users to create an account, update it, and log in and out.
Functional Requirements: FR1 - FR5

1. test-UAM-1

   Description: The user should be able to create an account with specified name, email address and password

   Control: Functional

   Initial State: The app is on the account creation screen and there is no existing account with the test email

   Input: Name, valid email, and password

   Output: Account creation is successful, and the user is redirected to the login screen

   Test Case Derivation: This test ensures that the application allows users to create an account. The manager expects a valid name with no numbers or special characters, a valid email (existing email of correct email format) and a password that only contains certain characters to prevent SQL injection

How test will be performed: The test will be performed by providing instructions to a user/tester to enter valid name, email and password submissions. They will then determine if the test is successful by verifying that the actual output matches the expected output (as provided in the instructions/referenced in this document)

2. test-UAM-2

   Description: The user should be able to login with valid credentials

   Control: Functional

   Initial State: The app is on the login page and a registered account has been created with the corresponding testing credentials

   Input: Email and password

   Output: The app is on the login page and a registered account has been created with the corresponding testing credentials

   Test Case Derivation: This test ensures that only authenticated users may access the app's functionalities

   How test will be performed: The test will be performed by having a user/tester provide valid credentials to the system and verify that the app redirects them to the home page after successful login

3. test-UAM-3

   Description: The user should be able to logout of the app

   Control: Functional

   Initial State: The user is already logged into the app

   Input: Logout submission

   Output: Account logout is successful, and the user is redirected to the login page

   Test Case Derivation: This test verifies that the app terminates the user's session and redirects the user to the login page to protect user's account data

   How test will be performed: While logged in, a user/tester will click the logout button and observe that the user is redirected to the

login screen. Verify that no user-specific data is accessible after logging out

4. test-UAM-4

   Description: Users should be able to update account information after initial setup

   Control: Functional

   Initial State: User is logged in and on the account settings screen

   Input: Modify fields (e.g., name or password) and save changes

   Output: Updated account information is saved and reflected on the profile screen.

   Test Case Derivation: The test ensures that users are able to manage and update their account information after initial setup

   How test will be performed: A tester/user will modify the account fields, save, and verify that updates are displayed correctly on the profile.

5. test-UAM-5

   Description: Users should not be able to access the application without logging in.

   Control: Functional

   Initial State: The application is open and the user has not logged in.

   Input: Attempt to access the application without logging in.

   Output: The application should prompt the user to log in.

   Test Case Derivation: The test verifies that the application requires users to log in before accessing its features.

   How test will be performed: The tester will attempt to access the application without logging in and verify that the application prompts them to log in.

6. test-UAM-6

   Description: Users should be able to reset their password.

   Control: Functional

Initial State: The application is open and the user has not logged in.

Input: Use the application's password reset feature.

Output: The user receives an email with instructions to reset their password.

Test Case Derivation: The test verifies that users can reset their password.

How test will be performed: The tester will use the application's password reset feature and verify that they receive an email with instructions to reset their password.

4.1.2 Image Processing Tests

This subsection covers tests required to verify tests revolving around receipt scanning and processing. The receipt scanner allows users to upload and preview images
Functional Requirements: FR8 - FR11

1. test-IP-1

   Description: User should be able to upload an image for processing

   Control: Functional

   Initial State: The application is on the receipt upload screen.

   Input: Select and upload a receipt image from the device's file storage

   Output: The uploaded image is successfully displayed for preview

   Test Case Derivation: The test verifies that users are able to upload images from their device and preview them before receipt scanning

   How test will be performed: The user/tester will select an image from storage, upload it, and confirm the image preview correctly displayed

2. test-IP-2

   Description: Users should be able to preview an uploaded image before processing and confirm/reupload the image based on preference

Control: Functional

Initial State: Image is uploaded and displayed as a preview.

Input: Confirm or reupload the previewed image

Output: Confirmed image proceeds to processing, or retake restarts the upload process

Test Case Derivation: Ensures users can review and confirm the uploaded image, as required for accuracy before processing.

How test will be performed: The user/tester previews the image, confirms, and observes if it advances to processing. Alternatively, users can retake and verify they are taken back to the image upload process

3. test-IP-3

Description: There is a limit to the file size of the image that can be uploaded.

Control: Functional

Initial State: The application is open and the user is on the image upload screen.

Input: Attempt to upload an image that exceeds the file size limit.

Output: The application should display an error message indicating that the file size limit has been exceeded.

How test will be performed: The tester will attempt to upload an image that exceeds the file size limit and verify that the application displays an error message.

4.1.3 Manual Expense Input Tests

This subsection covers tests that verify functionality for allowing users to manually input their expenses in the case that their receipt is unable to be processed. The manual input system is responsible for outlining fields that the user must fill out for their expense(s) to be tracked.
Functional Requirements: FR12-13

1. test-MIS-1

Description: Users should be able to manually input expenses

Control: Functional

Initial State: Image captured or uploaded cannot be processed or the user wants to manually input items from their receipt from the expense tracker view

Input: Entered items, item costs, item quantities, categories and receipt date

Output: Items, quantities, costs, categories and receipt date are correctly processed and displayed

Test Case Derivation: Ensures users can manually input their receipt/expense in case their image isn't able to be processed or they want to manually input them. The test verifies that manually inputted expense data is processed correctly

How test will be performed: From the expense tracker page, the user/tester will select the "manually input expenses" option and fill out the corresponding fields for category, item, item cost and date. They will then validate that all manual entries have been processed in the finalized expenses screen and match their inputs.

4.1.4 Database Management Tests

This subsection covers tests required to verify functionality for database management. The database managers securely stores user account information and user receipt information.
Functional Requirements: FR14 - FR15

1. test-DM-1

   Description: Account information should be protected and securely stored in the database

   Control: Automatic

   Initial State: Application database is empty or has only encrypted data.

   Input: Create a new user account.

   Output: User data is stored securely in the database, encrypted.

Test Case Derivation: The test validates that user account information is protected and verifies that the application securely handles user data

How test will be performed: Check database entries post-account creation to verify data encryption and account information

2. test-DM-2

Description: Receipt data must be securely stored in database

Control: Automatic

Initial State: Application database has no receipt data for the test user.

Input: Upload a receipt image.

Output: Receipt image and extracted data are securely stored in the database.

Test Case Derivation: The test verifies receipt data is securely stored, ensuring data privacy

How test will be performed: Check database entries post-upload to confirm storage security and receipt information

4.1.5 Item Recognition and Categorization

This subsection covers tests required to verify functionality for item recognition and categorization from receipt scanning. The receipt scanner is responsible for recognizing and categorizing items from users' uploaded receipts. *Functional Requirements: FR16 - FR18*

1. test-RS-1

Description: Uploaded receipts should be processed and return a list of items with their corresponding name, category, and price.

Control: Functional

Initial State: User confirms uploaded receipt is ready to be processed

Input: Run item recognition on a sample receipt

Output: Recognized items, quantities, and prices are displayed to the user

Test Case Derivation: The test verifies the end-to-end functionality of the OCR model by identifying items on the inputted receipt

How test will be performed: Process the receipt image and verify recognized details against the original receipt

2. test-RS-2

   Description: Uploaded receipts should categorize items on them under common expense categories (i.e. groceries, entertainment, etc..)

   Control: Functional

   Initial State: Receipt has been scanned and items are recognized from the receipt

   Input: Apply categorization based on item names.

   Output: Items are correctly categorized (i.e. milk under "Groceries").

   Test Case Derivation: The test validates organized expense tracking accuracy by categorizing items under common expense categories

   How test will be performed: Run categorization on a processed receipt and check that items are correctly sorted

3. test-RS-3

   Description: Verify that users can modify receipt data.

   Control: Functional

   Initial State: Receipt has been scanned and items are recognized and categorized from the receipt

   Input: Modify item names, quantities, prices, and categories.

   Output: Manual changes are saved and reflected in the expense tracker.

   Test Case Derivation: The test verifies that users can manually modify receipt data.

   How test will be performed: The tester will modify item names, quantities, prices, and categories and verify that the changes are saved and reflected in the expense tracker.

4.1.6 Financial Tracking Tests

This subsection verifies the functionality of financial tracking features such has spending history and budget management. The financial tracker is responsible for storing users expenses/spendings, as well as allowing users to set and track budgets.
Functional Requirements: FR19 - FR21

1. test-FT-1

   Description: Users should be able view their spending history by category and trends

   Control: Functional

   Initial State: Application has recorded user's past expenses and user is on the expense tracking view

   Input: Generate spending history overview

   Output: Accurate summary of total spending by category and time period

   Test Case Derivation: The test verifies users can monitor their budgets through viewing their spending trends

   How test will be performed: The tester/user should be able to trigger spending history generation and verify summaries match expected data upon entering the expense tracker view

2. test-FT-2

   Description: Users should be able to set and track their budgets

   Control: Functional

   Initial State: User is on the budgeting window and no current budget is set for the specified category

   Input: Budget limit for a specific category

   Output: Budget limit is saved and displayed in tracking

   Test Case Derivation: This test confirms the ability for users to set and track budgets

How test will be performed: The tester/user sets a budget for a specific category, then verify it appears and updates as expected in tracking

3. test-FT-3

   Description: Verify that users get notified when they are close to exceeding their budget or when a goal has been reached.

   Control: Functional

   Initial State: User has set a budget for a specific category and is close to exceeding it. OR User is also close to reaching a budgeting goal.

   Input: User inputs an expense that would exceed the budget. OR User inputs an income that would reach a savings goal.

   Output: User receives a notification.

   Test Case Derivation: The test verifies that users are notified when they are close to exceeding their budget or when a goal has been reached.

   How test will be performed: The tester will input an expense that would exceed the budget and verify that they receive a notification. The tester will also input an income that would reach a savings goal and verify that they receive a notification.

4.2 Tests for Nonfunctional Requirements

4.2.1 Accuracy Tests

This subsection covers tests required to verify the accuracy of the system in processing various financial data.
Nonfunctional Requirements: NFR1 - NFR6

1. test-ACC-1

   Description: Verify the accuracy of the machine learning model in categorizing expenses into predefined categories.

   Type: Automatic

Initial State: Machine learning model is set to have predetermined expense categories.

Input/Condition: A CSV file with various receipt item names

Output/Result: The model will output the categories of the given receipt items and compare it against the expected CSV file. The test passes if the model has a precision of at least 80%.

How test will be performed: Testers will run the code for testing item categorization, which will output a CSV file with the assigned categories.

2. test-ACC-2

Description: Verify the accuracy of the OCR model in recognizing text from images of receipts.

Type: Manual

Initial State: The OCR model is set to process images of receipts.

Input/Condition: A set of receipt images

Output/Result: The OCR model's accuracy in recognizing text is calculated. The test passes if the accuracy is at least 90%.

How the Test Will Be Performed: Testers will run the code for testing receipt parsing, and manually evaluate the results against what is shown on the receipt image. For each row, 0.5 points are given for the name being recognizable (does not have to be exact), and 0.5 points are given for the correct price (must be exact). 0.5 points are deducted for every extra/nonsensical row parsed. The total number of points is out of the number of items on the receipt, and the final percentage is calculated accordingly.

3. test-ACC-3

Description: Verify that all monetary calculations maintain a precision of up to two decimal places.

Type: Automatic

Initial State: The application is ready for financial calculations.

Input/Condition: A set of monetary calculations (budget changes, new purchases).

Output/Result: The output should display values rounded to two decimal places.

How the Test Will Be Performed: The system will perform different calculations and verify that all results maintain precision up to two decimal places as specified.

4. test-ACC-4

   Description: Verify the accuracy of the application in summing up expenses, incomes, and savings across different periods and categories.

   Type: Manual

   Initial State: The application is set to calculate expenses, incomes, and savings.

   Input/Condition: A dataset of transactions across different periods and categories.

   Output/Result: The application's summation accuracy is assessed.

   How the Test Will Be Performed: The system will input the transaction data and verify that the calculated totals are accurate, ensuring the application meets the 99% accuracy requirement in summation.

5. test-ACC-5

   Description: Verify that data synced across multiple devices is consistent.

   Type: Manual

   Initial State: The application is up to date. User is logged in.

   Input/Condition: Financial data entered on one device.

   Output/Result: Financial data appears on all devices.

   How the Test Will Be Performed: The tester will input financial data on one device, sync, and verify that all other devices show the same data to ensure 100% consistency.

4.2.2 Performance Tests

This subsection covers tests required to assess the performance of the application under various conditions.
Nonfunctional Requirements: NFR7 - NFR9

1. test-PERF-1

   Description: The application must load user data within five seconds.

   Type: Manual

   Initial State: The system is installed and ready to run.

   Input/Condition: The application is opened and user data is loaded.

   Output/Result: User data should load within five seconds.

   How the Test Will Be Performed: The user/tester will measure the time taken to load user account data.

2. test-PERF-2

   Description: The application must respond to user actions (e.g., adding an entry, generating a report) within two seconds.

   Type: Manual

   Initial State: User information has been loaded.

   Input/Condition: The user will be asked to perform various actions of their choice on the application.

   Output/Result: All actions should respond within two seconds.

   How the Test Will Be Performed: The user/tester will measure the response time for various user actions.

3. test-PERF-3

   Description: The system must be able to handle a minimum of 10 concurrent users without significant performance degradation.

   Type: Manual

   Initial State: The application is installed and running.

Input/Condition: Ten users log into the application simultaneously and perform typical tasks.

Output/Result: The application should maintain performance levels without significant degradation.

How the Test Will Be Performed: The tester will simulate 10 concurrent users accessing the application and monitor performance metrics (e.g., load times, response times) to ensure the application performs adequately under load.

4.2.3 Usability Tests

This subsection covers tests required to evaluate the usability and user experience of the application. After each of these tests, the user will be asked to fill out a usability survey, which can be found in Appendix A.

Nonfunctional Requirements: NFR10 - NFR17

1. test-USAB-1

   Description: The application must have a clean, intuitive, and easy-to-navigate interface, allowing users to perform common tasks (e.g., adding expenses, viewing budgets) within two to three clicks, and complete tasks such as adding a new receipt or setting a budget limit within 10 seconds on average.

   Type: Manual, Usability Testing

   Initial State: The application is installed, running, and ready for user interaction.

   Input/Condition: A user with no prior experience attempts to complete key tasks (e.g., uploading a receipt, viewing expense summaries, setting a budget), including scenarios where:

   - A page fails to load due to network issues
   - A required field is left blank in a form
   - A button is disabled without clear explanation

Output/Result: Tasks should show completion and result corresponding outputs.

How the Test Will Be Performed: The user/tester will evaluate the interface by performing tasks and counting the average number of clicks and the duration taken.

2. test-USAB-2

   Description: New users should be able to complete account setup and understand core application features within five minutes, supported by an introductory tutorial and tooltips.

   Type: Manual, Usability Testing

   Initial State: The application is installed and running. The state of the application is not authenticated.

   Input/Condition: A user performs various actions, including:

   - Uploading a large receipt, causing a system processing delay
   - Entering an invalid file format for receipt upload
   - Attempting to categorize an expense while offline

   Output/Result: The user should have a new account created and understand the core features of the application.

   How the Test Will Be Performed: The user/tester will simulate a new user experience, timing the setup and assessing the clarity of tutorials, tooltips, error messages, and corrective guidance.

3. test-USAB-3

   Description: Information displayed to users must be concise and relevant, minimizing cognitive effort while ensuring that only essential details are shown on the main dashboard, with advanced options accessible through menus.

   Type: Manual, Usability Testing

   Initial State: The application is displaying its main dashboard.

Input/Condition: The user/tester reviews the main dashboard and navigates through the menus. A user triggers different error conditions, such as:

- Uploading an unsupported file format (e.g., '.txt')
- Submitting a blank required field in an expense form
- Experiencing a timeout when fetching past transactions
- Entering an invalid number format (e.g., letters in a currency field)

Output/Result: All information is displayed and menus open upon command.

How the Test Will Be Performed: The user/tester will evaluate the main dashboard and menus for clarity, relevance, and ease of access, ensuring the design aligns with the usability requirements.

4. test-USAB-4

Description: Users should be able to recognize and recover from errors with clear guidance.

Type: Manual, Usability Testing

Initial State: The application is installed and running.

Input/Condition: A user attempts to perform an action that results in an error, including but not limited to:

- Network disconnection during receipt upload
- Uploading an unsupported file format
- System timeout during data processing
- Uploading a corrupted or unreadable receipt image

Output/Result: The user should understand the error through clear messaging and be able to take corrective actions to resolve it.

How the Test Will Be Performed: The user/tester will intentionally trigger different error scenarios and assess:

- Whether the system correctly detects the error

- The clarity of the displayed error message
- The ease of taking corrective actions (e.g., retrying, re-uploading, or receiving alternative guidance)

4.2.4 Security Tests

This subsection covers tests required to ensure the security and protection of user data within the system.
Nonfunctional Requirements: NFR18 - NFR19

1. test-SEC-1

   Description: Verify that authentication mechanisms prevent unauthorized access and that sensitive data, including passwords, are securely stored.

   Type: Inspection, Manual

   Initial State: The system is fully deployed and operational.

   Input/Condition: A tester attempts various security breaches, including:

   - SQL injection attacks through login and input fields
   - Brute-force login attempts with rapid credential entry
   - Session hijacking by stealing cookies or tokens
   - Attempting to access restricted user data without authentication

   Output/Result: The system should prevent unauthorized access, detect suspicious login attempts, and lock out users after repeated failed logins. Error messages should be vague enough to avoid revealing system vulnerabilities.

   How the Test Will Be Performed:

   - The tester will attempt SQL injection in form fields and verify that queries are properly sanitized.
   - Session tokens will be analyzed to ensure secure expiration policies.
   - Authentication bypass attempts will be performed to verify proper access controls.

4.2.5 Maintainability Tests

This subsection covers tests required to verify the maintainability and compatibility of the application.
Nonfunctional Requirements: NFR20 - NFR22

1. test-MTB-1

   Description: New releases must maintain backward compatibility with previous versions and ensure system stability during updates.

   Type: Manual, Automated

   Initial State: The application is ready for a new release.

   Input/Condition: A new release is applied to the application, including:

   - Database schema changes
   - API endpoint modifications
   - Frontend framework or library updates
   - Dependency version upgrades

   Output/Result: The system must provide a confirmation message indicating a successful update, maintain functionality across versions, and log errors if any incompatibilities occur.

   How the Test Will Be Performed:

   - The tester will apply the new release and check logs for compatibility errors.
   - A rollback test will be performed to ensure users can revert to a stable version if needed.
   - User actions will be simulated post-update to verify that no regressions occurred.

2. test-MTB-2

   Description: Dependencies must be regularly updated while ensuring compatibility with the existing system.

Type: Automatic

Initial State: Dependencies are out of date.

Input/Condition: A new patch is applied to the application, including:

- Security patches for third-party libraries
- Framework or SDK version updates

Output/Result: The system updates dependencies while ensuring no breaking changes occur. If incompatibilities are detected, developers are alerted.

How the Test Will Be Performed:

- Automated dependency scanning tools will check for vulnerabilities.
- Test suites will run after updates to verify compatibility.

3. test-MTB-3

Description: The codebase must be covered by automated tests to detect regressions before deployment.

Type: Automatic

Initial State: The application is up to date.

Input/Condition: A new code change is made, triggering the CI/CD pipeline.

Output/Result: The pipeline runs automated tests, checking for:

- Unit test coverage across core components
- Integration tests ensuring API and database interactions work correctly
- Performance tests identifying bottlenecks in system response times

How the Test Will Be Performed:

- Automated tests will be executed through GitHub Actions.

- Code coverage reports will be generated, and any PR failing minimum coverage will be blocked.
- Manual review will be conducted for critical code paths.

4.2.6 Portability Tests

This subsection covers tests required to assess the portability of the application across different platforms and environments.
Nonfunctional Requirements: NFR23 - NFR 26

1. test-PORT-1

   Description: The software shall be compatible with Android and iOS mobile devices running the latest software.

   Type: Automatic

   Initial State: The application is deployed on Android and iOS devices.

   Input/Condition: Users attempt to install and run the application.

   Output/Result: The system outputs successful installation messages and the application functions correctly on both platforms.

   How the Test Will Be Performed: Automated tests will verify installation and functionality on both devices.

2. test-PORT-2

   Description: All data must be stored in platform-independent formats (e.g., JSON, CSV), and any external APIs or third-party services must support cross-platform usage.

   Type: Automatic

   Initial State: The application has data stored and is integrated with external APIs.

   Input/Condition: The user/tester requests data export and initiates API calls.

   Output/Result: The system outputs data in specified formats and returns successful responses from APIs on both platforms.

How the Test Will Be Performed: Automated scripts will verify data export formats and API integration on both Android and iOS.

3. test-PORT-3

   Description: The system must operate in various network environments, including Wi-Fi, cellular, and offline mode.

   Type: Manual

   Initial State: The application is installed and ready for testing.

   Input/Condition: The user/tester simulates different network environments.

   Output/Result: The system outputs successful operation messages for each network condition.

   How the Test Will Be Performed: The user/tester will manually switch between different network environments (Wi-Fi, cellular, offline) and observe the application's behavior, checking for consistent functionality and performance across each condition.

4.2.7 Reusability Tests

This subsection covers tests required to evaluate the reusability of components and functionalities within the application.
Nonfunctional Requirements: NFR27 - NFR 30

1. test-REUS-1

   Description: The application must be developed using reusable components (e.g., UI components, API services) and adhere to the principle of separation of concerns, ensuring that business logic, data access, and presentation layers are kept separate.

   Type: Manual

   Initial State: The application is developed and ready for testing.

   Input/Condition: The user/tester reviews the codebase and architecture.

Output/Result: The system must show evidence of reusable components and separate layers, with no code duplication.

How the Test Will Be Performed: The user/tester will conduct a manual code review to assess the architecture for reusable components, ensuring adherence to the principle of separation of concerns. They will document findings regarding code duplication and the organization of business logic, data access, and presentation layers.

2. test-REUS-2

Description: Common functionalities (e.g., receipt parsing, authentication) must be abstracted into reusable libraries, and design elements (e.g., icons, typography) must be created as reusable assets.

Type: Automatic

Initial State: The application is fully developed.

Input/Condition: The user/tester inspects libraries and design assets.

Output/Result: The system must output a report confirming that functionalities are encapsulated in reusable libraries and design assets are consistently applied across the application.

How the Test Will Be Performed: Automated tools will verify the presence of reusable libraries and assets.

4.2.8 Understandability Tests

This subsection covers tests required to assess the understandability and clarity of the application and its documentation.
Nonfunctional Requirements: NFR31 - NFR 39

1. test-UND-1

Description: To validate that the codebase is well-documented, organized, and follows consistent naming conventions and coding standards.

Type: Static, Code Review/Walkthrough

Initial State: The codebase is up to date and ready for review.

Input/Condition: The inspector reviews the code and associated documentation.

Output/Result: A review report containing a summary of the codebase's documentation, organization, and adherence to coding standards.

How the Test Will Be Performed: A team of developers will conduct a walkthrough of the codebase, using static analysis tools to check for documentation completeness and adherence to naming conventions. A designated reviewer will conduct a walkthrough of the API documentation and commit history, checking for completeness and clarity. The results will be documented in a compliance report, which will be reviewed by the development team.

2. test-UND-2

   Description: To validate that the user interface (UI) is simple to navigate and understand.

   Type: Usability Test

   Initial State: The application is up to date.

   Input/Condition: The user/tester interacts with the UI.

   Output/Result: A usability report containing feedback on the UI's clarity and organization.

   How the Test Will Be Performed: The user/tester will manually navigate the UI and fill in a usability survey, providing feedback on the clarity and organization of the interface.

3. test-UND-3

   Description: To validate that error messages are clear to the user and logs are informative to developers.

   Type: Manual

   Initial State: The application is up to date.

   Input/Condition: The user/tester intentionally triggers an error.

Output/Result: The user will provide feedback on the clarity of the error message, and the developers will assess the informativeness of the logs.

How the Test Will Be Performed: The user/tester will intentionally trigger an error and provide feedback on the clarity of the error message. The developers will review the logs to ensure they contain sufficient information to diagnose the issue.

4.2.9 Regulatory Tests

This subsection covers tests required to ensure compliance with relevant regulations and standards.
Nonfunctional Requirements: NFR40 - NFR 41

1. test-REG-1

Description: The system must comply with Canada's Privacy Act and Financial Administration Act to ensure the privacy and security of user data and the secure handling of financial information.

Type: Manual, Inspection

Initial State: The application is up to date.

Input/Condition: An inspector reviews the system's data handling policies and financial procedures.

Output/Result: The system must demonstrate adherence to both regulatory acts through documented policies and secure practices.

How the Test Will Be Performed: The inspector will examine relevant documentation, assess data handling procedures, and conduct interviews with compliance and development teams to verify compliance with both the Privacy Act and the Financial Administration Act.

4.3 Traceability Between Test Cases and Requirements

A traceability matrix between test cases and requirements can be found here.

5 Unit Test Description

Test cases will be written for each module with access routine semantics (i.e., functionality) outlined in the Module Guide (MG). One test case will be written for normal behaviour, and additional test cases will be written for any exceptional/edge cases. Similarly, test cases will be written for each endpoint in the API. Additionally, the frontend will have unit tests written for each component and screen, with test cases covering normal behaviour and edge cases.
Details for each unit test will refer directly to their respective test in the codebase.

5.0.1 Component Rendering

- **Description:** Each component is rendered in their corresponding unit test to ensure it renders without error and with all corresponding mock information correctly (i.e. the component rendered with mock expense data is expected to display the mock expense data)
- **Inputs:** The component & component props
- **Expected Output:** Component renders without error and displays corresponding information
- **Result:** Pass

5.0.2 Metrics

This section describes the unit tests for calculating and displaying user metrics.

Expense and Budget

- **Description:** Unit tests to render expense and budget metrics to validate: total expense value, total budget value, percentage of budget spent, and how much value for each budget category is left
- **Inputs:** Expense array, budget array

- **Expected Output:** Corresponding metrics are correctly calculated and displayed to users
- **Result:** Pass

Expense categories

- **Description:** Unit test to render the expense metrics graph. This test validates that the graph correctly displays the top 5 expense categories by total cost and accurately shows the percentage of each category.
- **Inputs:** Expense array
- **Expected Output:** The metrics are correctly calculated and displayed to users
- **Result:** Pass

5.0.3 Manage resources

This section describes the unit tests for managing the CRUD functions of budgets, incomes and expenses.

Manage Expenses

- **Description:** Unit tests to display user expenses and allow users to add expense
- **Inputs:** Expenses array
- **Expected Output:** Expenses correctly rendered and displayed, users are able enter expense information and then save corresponding expense information
- **Result:** Pass

Manage Budget

- **Description:** Unit tests to verify left over budget (or over budget if users have expensed more than their budget), total expenses, and budget for each expense category. Also will validate actions for deleting, editing and adding new budget.

- **Inputs:** Expenses array, budgets array
- **Expected Output:** Corresponding remaining budget displayed along with total expenses. Budget for each expense category correctly rendered
- **Result:** Pass

Manage Income

- **Description:** Unit tests to validate user income(s) and how much more users are able to budget (if applicable). Also validates actions for deleting, editing and adding new income.
- **Inputs:** Incomes array, budgets array
- **Expected Output:** Corresponding income displayed along with the remaining budget. Users are able to enter income information and save it correctly. Actions for deleting, editing, and adding new income are validated.
- **Result:** Pass

5.1 Unit Testing Scope

OCR Module: While the accuracy of the parsed text from an image using the OCR will be tested, the actual OCR model will not be unit tested, as it is a third-party service.
Authentication Module: The authentication module will be tested for correctly authenticating users and handling invalid credentials, but the actual authentication service will not be unit tested, as it is a third-party service.

5.2 Tests for Functional Requirements

This section will be filled after completion of the MIS document.

5.3 Tests for Nonfunctional Requirements

This section will be filled after completion of the MIS document.

5.4 Traceability Between Test Cases and Modules

This section will be filled after completion of the MIS document.

References

No references were used for this document.

Appendix A — Usability Survey Questions

1. **Overall Experience**
    - On a scale of 1 to 5, how easy was it to scan and upload receipts? (1 = Very Difficult, 5 = Very Easy)
    - On a scale of 1 to 5, how intuitive was the app's interface? (1 = Not Intuitive, 5 = Very Intuitive)
    - On a scale of 1 to 5, how satisfied are you with the accuracy of the receipt scanning and categorization? (1 = Not Satisfied, 5 = Very Satisfied)
2. **Receipt Scanning and AI Parsing**
    - Did you experience any difficulties capturing clear images of receipts? (Yes/No)
    - Did the app correctly categorize your spending based on the receipt items? (Yes/No)
    - If the app made mistakes in categorization, were you able to correct them easily? (1 = Very Difficult, 5 = Very Easy)
3. **Budgeting and Spending Insights**
    - Did the budgeting feature help you understand your spending patterns? (1 = Not at All, 5 = Very Much)
    - How clear was it to see the difference between your actual spending and budgeted amount? (1 = Not Clear, 5 = Very Clear)
    - How useful were the insights provided about expected vs. actual spending? (1 = Not Useful, 5 = Very Useful)
4. **User Interface and Design**
    - On a scale of 1 to 5, how visually appealing did you find the app's design? (1 = Not Appealing, 5 = Very Appealing)
    - Was the layout of information easy to understand? (1 = Very Confusing, 5 = Very Clear)
    - Were you able to navigate between features without confusion? (Yes/No)

5. **Error Handling and Support**

   - Did you encounter any errors or issues during your experience? (Yes/No)
   - If yes, how helpful were the error messages in resolving the issue? (1 = Not Helpful, 5 = Very Helpful)
   - Do you feel that any features or support options are missing? (Yes/No)

6. **Overall Satisfaction and Feedback**

   - How likely are you to use this app regularly? (1 = Not Likely, 5 = Very Likely)
   - Which feature did you find most useful? (Select all that apply: Receipt Scanning, Spending Categorization, Budget Tracking, Savings Calculation, Spending Insights)
   - Would you recommend this app to others? (Yes/No)

Appendix B — Reflection

The information in this section will be used to evaluate the team members on the graduate attribute of Lifelong Learning.

1. What went well while writing this deliverable?

   During this deliverable, the group divided the work up well through delegating certain sections to each member to complete. Each group member was able to seamlessly complete their section without many issues and conflicts. This was prevelant through the PR review process where there were minimal comments for each PR.

2. What pain points did you experience during this deliverable, and how did you resolve them?

   Some pain points that the group discussed was what we want to include in Rev0 of the VnV plan. There were some scope issues that we discussed, specifically around which functional requirements we want to include in the initial draft and include tests for. Additionlly, the team discussed the granularity of the tests that we want to include in the Rev0 VnV document. The group did not initially agree on whether to include unit tests into this document. The team brought it up, discussed it and in the end came to a conclusion of not including unit tests within this document because the components/units to test have not yet been designed in detail.

3. What knowledge and skills will the team collectively need to acquire to successfully complete the verification and validation of your project? Examples of possible knowledge and skills include dynamic testing knowledge, static testing knowledge, specific tool usage, Valgrind etc. You should look to identify at least one item for each team member.

   The team will collectively need to look into how to test machine learning models as this topic is new for all group members. We will also need to look into setting up GitHub Actions for continuous integration, so that we can implement automated testing and code coverage. A few of us have utilized GitHub Actions at previous co-ops and can utilize that knowledge to help set up GitHub Actions for the Plutos project. Specifically, this will be the breakdown per team member:

- Payton: Familiarize with continuous integration and how to set up GitHub Actions for automated testing and code coverage
- Eric: Deepen knowledge on backend testing frameworks and how to reach a high code coverage
- Fondson: Deepen knowledge on frontend testing frameworks, and ways to simulate user interactions for end-to-end testing
- Jason: Learn more about how to best conduct usability testing to evaluate user experiences, identify pain points, and improve the overall design and functionality of the project
- Angela: Familiarize with static testing tools and techniques to identify code quality issues and ensure the codebase follows best practices

4. For each of the knowledge areas and skills identified in the previous question, what are at least two approaches to acquiring the knowledge or mastering the skill? Of the identified approaches, which will each team member pursue, and why did they make this choice?

   All group members are currently taking or have taken Introduction to Machine Learning (4ML3), and Jason is currently taking Applications of Machine Learning (4AL3). Through these courses, we can learn about how to test machine learning models, which would be the most difficult part of the project to test.
   For the other knowledge areas, our team has all decided that we will use a combination of YouTube and Coursera/LinkedIn Learning. We have decided that these resources are the most straightforward to learn from and we all learn best from watching videos and putting them to practice. For GitHub Actions, we will also consult GitHub documentation as it would be the most reliable source of information for setting it up and understanding how it works.