System Verification and Validation Plan for Plutos

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

November 4, 2024

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
...	...	...

[The intention of the VnV plan is to increase confidence in the software. However, this does not mean listing every verification and validation technique that has ever been devised. The VnV plan should also be a **feasible** plan. Execution of the plan should be possible with the time and team available. If the full plan cannot be completed during the time available, it can either be modified to "fake it", or a better solution is to add a section describing what work has been completed and what work is still planned for the future. —SS]

[The VnV plan is typically started after the requirements stage, but before the design stage. This means that the sections related to unit testing cannot initially be completed. The sections will be filled in after the design stage is complete. the final version of the VnV plan should have all sections filled in. —SS]

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:

symbol	**description**
T	Test
V&V	Verification and Validation

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).
- **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

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

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

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

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

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

[You should at least point to the tests listed in this document and the unit testing plan. —SS]

[In this section you would also give any details of any plans for static verification of the implementation. Potential techniques include code walkthroughs, code inspection, static analyzers, etc. —SS]

[The final class presentation in CAS 741 could be used as a code walkthrough. There is also a possibility of using the final presentation (in CAS741) for a partial usability survey. —SS]

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

[If there is any external data that can be used for validation, you should point to it here. If there are no plans for validation, you should state that here. —SS]

[You might want to use review sessions with the stakeholder to check that the requirements document captures the right requirements. Maybe task based inspection? —SS]

[For those capstone teams with an external supervisor, the Rev 0 demo should be used as an opportunity to validate the requirements. You should plan on demonstrating your project to your supervisor shortly after the scheduled Rev 0 demo. The feedback from your supervisor will be very useful for improving your project. —SS]

[For teams without an external supervisor, user testing can serve the same purpose as a Rev 0 demo for the supervisor. —SS]

[This section might reference back to the SRS verification section. —SS]

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: Manual

   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: Manual

   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: Manual

   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: Manual

   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.

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: Manual

   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: Manual

   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

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: Manual

   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 all items, their quantities and prices should all be correctly recognized with an accuracy of TARGET_OCR_MODEL_ACCURACY

   Control: Automatic

   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 accurately

   Test Case Derivation: The test verifies the accuracy 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: Automatic

   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

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 spending their spending history by category and trends

   Control: Manual

   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: Manual

   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

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: The machine learning model must achieve at least 80% accuracy in correctly identifying and extracting key information (i.e., item names, quantities, prices, dates) from receipts.

   Type: Manual

Initial State: Machine learning model is ready for testing with a dataset of receipts prepared for evaluation.

Input/Condition: Test receipts containing various sets of key information (item names, quantities, prices, and dates).

Output/Result: Model's accuracy measured by the percentage of key information extracted correctly.

How test will be performed: The output will be compared with the initially inputted test receipts to compare information. The accuracy will be computed by determining the percentage of correctly identified and extracted information and will be determined if it meets the 80% threshold.

2. test-ACC-2

   Description: The machine learning model must categorize expenses into predefined categories (e.g., groceries, utilities, entertainment) with at least 90% precision to ensure users' financial data is correctly organized.

   Type: Manual

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

   Input/Condition: A set of receipt transactions with known categories.

   Output/Result: The model will output the categories of the given transactions, and the precision will be calculated from this.

   How test will be performed: The model will process the set of transactions and compare the assigned categories to the correct ones. Precision will be measured to see if the 90% threshold is met.

3. test-ACC-3

   Description: All monetary calculations (e.g., totals, budgets, currency conversions) must maintain a precision of up to two decimal places to ensure accurate financial reporting.

   Type: Manual

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: The OCR model must correctly recognize text from images of receipts with a minimum accuracy rate of 90%, ensuring minimal manual corrections by users.

   Type: Manual

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

   Input/Condition: A set of receipt images with known text.

   Output/Result: The OCR model's accuracy in recognizing text is calculated.

   How the Test Will Be Performed: The OCR model will process the receipt images and compare the recognized text to the actual text, ensuring an accuracy rate of at least 90%.

5. test-ACC-5

   Description: The application must guarantee 99% accuracy in summing up expenses, incomes, and savings across different periods and categories, ensuring no rounding or summation errors.

   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.

6. test-ACC-6

   Description: If the application syncs data across multiple devices, it should ensure 100% consistency of financial data across all instances.

   Type: Manual

   Initial State: The application is synced across multiple devices.

   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 and respond to user actions (e.g., adding an entry, generating a report) within two seconds to ensure a smooth user experience.

   Type: Manual

   Initial State: The application is set to load user data and respond to user actions.

   Input/Condition: A user initiates the data loading process and performs user actions simultaneously.

   Output/Result: User data should load within five seconds, and all actions should respond within two seconds.

How the Test Will Be Performed: The user/tester will measure the time taken to load user account data and the response time for various user actions during the same test session, ensuring both criteria are met.

2. test-PERF-2

   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 set up to support concurrent users.

   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.
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

   Initial State: The application is ready for user interaction.

Input/Condition: A user attempts to perform common tasks such as editing budgets or adding a new receipt using the interface.

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, ensuring the requirements are met.

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. Additionally, the application should provide clear error messages and guide users through corrective actions, allowing them to recover from errors in no more than two steps.

   Type: Manual

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

   Input/Condition: A new user begins the account setup and interacts with the application, making intentional errors to test error handling.

   Output/Result: The user should receive informative error messages and be guided through corrective actions.

   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

   Initial State: The application is displaying its main dashboard.

   Input/Condition: The user/tester reviews the main dashboard and navigates through the menus.

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.

4.2.4 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: The system must implement secure password storage and authentication mechanisms, as well as ensure that all data transmitted between the client and server is encrypted to protect user data.

   Type: Automatic

   Initial State: The system is fully deployed and operational.

   Input/Condition: User attempts to create an account and log in, while data transmission occurs between client and server.

   Output/Result: The system must confirm successful account creation and authentication with appropriate messages, and all data transmitted must show as encrypted.

   How the Test Will Be Performed: The user/tester will create an account, log in, and monitor the data transmission to verify that passwords are securely stored and that encryption protocols are in place for all data exchanged.

4.2.5 Maintainability Tests

This subsection covers tests required to verify the maintainability and compatibility of the application.
Nonfunctional Requirements: NFR20 - NFR 22

1. test-MTB-1

   Description: New releases must maintain backward compatibility with previous versions, ensuring users can seamlessly update the application without experiencing disruptions in their data or workflows. Dependencies must be regularly updated to prevent security vulnerabilities, and at least 80% of the codebase must be covered by automated unit tests to ensure maintainability.

   Type: Automatic

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

   Input/Condition: A new release is applied to the application, along with dependency updates.

   Output/Result: The system must output a confirmation message indicating a successful update, and system logs should show no errors or disruptions. Additionally, automated test results must indicate at least 80% code coverage.

   How the Test Will Be Performed: The user/tester will apply the new release, check for compatibility by performing typical user tasks, and run automated tests to verify code coverage, ensuring the requirements are met.

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: All source code must be thoroughly documented with inline comments and external documentation, and variables, functions, classes, and modules must follow consistent and meaningful naming conventions.

   Type: Static

   Initial State: The codebase is complete and ready for review.

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

   Output/Result: The system must show that code components are well-documented, with meaningful names that describe functionality.

   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. The results will be documented in a compliance report, which will be reviewed by the development team.

2. test-UND-2

   Description: The application's UI must be designed with clarity, using clear labels, icons, and tooltips, and the codebase must be organized logically, allowing easy navigation.

   Type: Manual

   Initial State: The application is fully developed.

   Input/Condition: The user/tester interacts with the UI and examines the code structure.

Output/Result: The system must provide a UI that is easy to navigate, with clear labels and organized code files.

How the Test Will Be Performed: The user/tester will manually navigate the UI and review the directory structure of the codebase to assess clarity and organization.

3. test-UND-3

   Description: The code must follow industry-standard formatting guidelines, and the application must provide clear, descriptive error messages for users and developers.

   Type: Static

   Initial State: The codebase and application are fully developed.

   Input/Condition: The user/tester reviews the code formatting and triggers error messages during testing.

   Output/Result: The system must show properly formatted code and clear error messages that explain issues and solutions.

   How the Test Will Be Performed: A review team will perform code inspections, using static analysis tools to evaluate code formatting against industry standards. During application testing, users will intentionally trigger errors to assess the clarity and helpfulness of error messages. Feedback will be collected and addressed in subsequent development cycles.

4. test-UND-4

   Description: Any APIs must include detailed documentation, and all major changes to the codebase should be accompanied by clear commit messages and changelogs.

   Type: Static

   Initial State: The APIs and codebase are implemented.

   Input/Condition: The user/tester reviews API documentation and commit history.

   Output/Result: The system must provide comprehensive API documentation and meaningful commit messages that clarify changes.

How the Test Will Be Performed: A designated reviewer will conduct a walkthrough of the API documentation and commit history, checking for completeness and clarity. Automated documentation generation tools may be used to assist in this process. The findings will be compiled into a report and discussed in a review meeting with the development team.

5. test-UND-5

   Description: The language and terminology used throughout the application's UI must align with users' mental models, using familiar terms.

   Type: Manual

   Initial State: The application is deployed.

   Input/Condition: The user/tester evaluates the UI language and terminology.

   Output/Result: The system must use non-technical, familiar terms throughout the UI.

   How the Test Will Be Performed: The user/tester will manually navigate the UI to assess the terminology used and provide feedback on its alignment with user expectations.

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

   Initial State: The system is fully implemented and operational.

Input/Condition: The user/tester 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 user/tester will manually 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

[Provide a table that shows which test cases are supporting which requirements. —SS]

5 Unit Test Description

[This section should not be filled in until after the MIS (detailed design document) has been completed. —SS]

[Reference your MIS (detailed design document) and explain your overall philosophy for test case selection. —SS]

[To save space and time, it may be an option to provide less detail in this section. For the unit tests you can potentially layout your testing strategy here. That is, you can explain how tests will be selected for each module. For instance, your test building approach could be test cases for each access program, including one test for normal behaviour and as many tests as needed for edge cases. Rather than create the details of the input and output here, you could point to the unit testing code. For this to work, you code needs to be well-documented, with meaningful names for all of the tests. —SS]

5.1 Unit Testing Scope

[What modules are outside of the scope. If there are modules that are developed by someone else, then you would say here if you aren't planning on verifying them. There may also be modules that are part of your software, but have a lower priority for verification than others. If this is the case, explain your rationale for the ranking of module importance. —SS]

5.2 Tests for Functional Requirements

[Most of the verification will be through automated unit testing. If appropriate specific modules can be verified by a non-testing based technique. That can also be documented in this section. —SS]

5.2.1 Module 1

[Include a blurb here to explain why the subsections below cover the module. References to the MIS would be good. You will want tests from a black box perspective and from a white box perspective. Explain to the reader how the tests were selected. —SS]

1. test-id1

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input:

   Output: [The expected result for the given inputs —SS]

   Test Case Derivation: [Justify the expected value given in the Output field —SS]

   How test will be performed:

2. test-id2

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input:

   Output: [The expected result for the given inputs —SS]

   Test Case Derivation: [Justify the expected value given in the Output field —SS]

   How test will be performed:

3. ...

5.2.2 Module 2

...

5.3 Tests for Nonfunctional Requirements

[If there is a module that needs to be independently assessed for performance, those test cases can go here. In some projects, planning for nonfunctional tests of units will not be that relevant. —SS]

[These tests may involve collecting performance data from previously mentioned functional tests. —SS]

5.3.1 Module ?

1. test-id1

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input/Condition:

   Output/Result:

   How test will be performed:

2. test-id2

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input:

   Output:

   How test will be performed:

5.3.2 Module ?

...

5.4 Traceability Between Test Cases and Modules

[Provide evidence that all of the modules have been considered. —SS]

References

6 Appendix

This is where you can place additional information.

6.1 Symbolic Parameters

The definition of the test cases will call for SYMBOLIC_CONSTANTS. Their values are defined in this section for easy maintenance.

6.2 Usability Survey Questions?

[This is a section that would be appropriate for some projects. —SS]

Appendix — Reflection

[This section is not required for CAS 741 —SS]

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

The purpose of reflection questions is to give you a chance to assess your own learning and that of your group as a whole, and to find ways to improve in the future. Reflection is an important part of the learning process. Reflection is also an essential component of a successful software development process.

Reflections are most interesting and useful when they're honest, even if the stories they tell are imperfect. You will be marked based on your depth of thought and analysis, and not based on the content of the reflections themselves. Thus, for full marks we encourage you to answer openly and honestly and to avoid simply writing "what you think the evaluator wants to hear."

Please answer the following questions. Some questions can be answered on the team level, but where appropriate, each team member should write their own response:

1. What went well while writing this deliverable?

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

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.

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?