Software Requirements Specification
Plutos: Smart Budgeting Expense Tracker

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

October 11, 2024

Contents

Revision History

Date	Version	Notes
10/07/2024	1.0	Added in the following sections to the SRS: Reference Materials (1), NFRs (5.2), Likely Changes (6), Unlikely Changes (7), Development Plan (9), Auxiliary Constraints (10)
10/08/2024	1.1	Added in the following sections to the SRS: Purpose of Document (2.1), Organization of Document (2.4), User Characteristics (3.2)
10/09/2024	1.2	Added in the following sections to the SRS: Characteristics of Intended Reader (2.3), Functional Requirements (5.1)
10/10/2024	1.3	Added in the following sections to the SRS: System Constraints (3.3), Requirements Rationale (5.3)

[This template is intended for use by CAS 741. For CAS 741 the template should be used exactly as given, except the Reflection Appendix can be deleted. For the capstone course it is a source of ideas, but shouldn't be followed exactly. The exception is the reflection appendix. All capstone SRS documents should have a refelection appendix. —TPLT]

1 Reference Material

This section records information for easy reference.

1.1 Table of Units

N/A; units are not used in this SRS.

1.2 Table of Symbols

N/A; symbols are not used in this SRS.

1.3 Abbreviations and Acronyms

symbol	description
NLP	Natural Language Processing
OCR	Optical Character Recognition
JWT	JSON Web Token
AI	Artificial Intelligence
ML	Machine Learning
LC	Likely Change
ULC	Unlikely Change
SRS	Software Requirements Specification
TM	Theoretical Model
Plutos	[put an expanded version of your program name here (as appropriate) —TPLT]

1.4 Mathematical Notation

N/A; mathematical notation is not used in this SRS.

[This SRS template is based on Smith and Lai (2005); Smith et al. (2007); Smith and Koothoor (2016). It will get you started. You should not modify the section headings, without first discussing the change with the course instructor. Modification means you are not following the template, which loses some of the advantage of a template, especially standardization. Although the bits shown below do not include type information, you may need to add this information for your problem. If you are unsure, please can ask the instructor. —TPLT]

[Feel free to change the appearance of the report by modifying the LaTeX commands. —TPLT]

[This template document assumes that a single program is being documented. If you are documenting a family of models, you should start with a commonality analysis. A separate template is provided for this. For program families you should look at Smith (2006); Smith et al. (2017). Single family member programs are often programs based on a single physical model. General purpose tools are usually documented as a family. Families of physical models also come up. —TPLT]

[The SRS is not generally written, or read, sequentially. The SRS is a reference document. It is generally read in an ad hoc order, as the need arises. For writing an SRS, and for reading one for the first time, the suggested order of sections is:

- Goal Statement
- Instance Models
- Requirements
- Introduction
- Specific System Description

—TPLT]

[Guiding principles for the SRS document:

- Do not repeat the same information at the same abstraction level. If information is repeated, the repetition should be at a different abstraction level. For instance, there will be overlap between the scope section and the assumptions, but the scope section will not go into as much detail as the assumptions section.

—TPLT]

[The template description comments should be disabled before submitting this document for grading. —TPLT]

[You can borrow any wording from the text given in the template. It is part of the template, and not considered an instance of academic integrity. Of course, you need to cite the source of the template. —TPLT]

[When the documentation is done, it should be possible to trace back to the source of every piece of information. Some information will come from external sources, like terminology. Other information will be derived, like General Definitions. —TPLT]

[An SRS document should have the following qualities: unambiguous, consistent, complete, validatable, abstract and traceable. —TPLT]

[The overall goal of the SRS is that someone that meets the Characteristics of the Intended Reader (Section 2.3) can learn, understand and verify the captured domain knowledge. They should not have to trust the authors of the SRS on any statements. They should be able to independently verify/derive every statement made. —TPLT]

2 Introduction

[The introduction section is written to introduce the problem. It starts general and focuses on the problem domain. The general advice is to start with a paragraph or two that describes the problem, followed by a "roadmap" paragraph. A roadmap orients the reader by telling them what sub-sections to expect in the Introduction section. —TPLT]

2.1 Purpose of Document

The purpose of this SRS document is to describe the functional and non-functional requirements of the Plutos budgeting application. This document serves as a formal agreement between stakeholders, including developers, project managers, and end users, to ensure a shared understanding of the system's objectives, capabilities, and constraints.

The SRS defines the expectations for the project, detailing the system features, behaviour, and performance requirements. It serves as a reference throughout the development lifecycle, guiding the design, implementation, testing, and validation phases to ensure the final product meets the agreed-upon specifications. Additionally, this document will support future maintenance and scalability of the application by providing clear and detailed requirements that facilitate ongoing enhancements.

2.2 Scope of Requirements

The scope of this project encompasses the development of a budgeting application, Plutos, that automates expense tracking and categorization using AI. The system is designed to address key pain points for young adults struggling to manage their finances, specifically focusing on automating the process of tracking spending through receipt scanning, categorization of expenses, providing metrics to users regarding their spending habits, and providing feedback on how users can meet their budgeting goals.

However, some features that are outside the scope of the project (though may be implemented later) are:

- The system will not account for complex financial scenarios such as investments, stock portfolios, or retirement planning.

- Advanced financial forecasting or predictive analytics beyond simple budgeting trends will not be implemented.
- The AI model will not handle non-standard or highly complex receipts (e.g., multi-page invoices or handwritten receipts).
- The application will not offer integration with external financial accounts such as credit cards or bank accounts.
- The model will be trained to detect receipt data from Fortinos and will not initially support various receipt formats.

These exclusions allow the project to concentrate on core features, such as receipt scanning and expense categorization, while ensuring a manageable scope for the development process.

2.3 Characteristics of Intended Reader

The primary audience for this SRS document is the group of undergraduate software engineering students who will oversee and complete the project's design and implementation. This group of specialists, which includes system architects, software developers, and quality assurance testers, has extensive experience in the range of technologies needed for the project. Through their coursework and personal experiences, they are familiar with the process of developing mobile applications, and as they move through the project milestones, they will deepen their understanding of artificial intelligence and machine learning. They will actively learn how to apply AI/ML technologies in real-world circumstances throughout the project, which will advance their development as software engineers. The collaborative effort will help them improve their real-world application development abilities as the project progresses through seven milestones, ensuring that they are prepared to produce a practical and user-friendly budgeting solution.

2.4 Organization of Document

The following sections of the SRS will further describe the system and its requirements. The sections will be as follows:

3. General System Description
4. Specific System Description
5. Requirements
6. Likely Changes
7. Unlikely Changes
8. Traceability Matrices and Graphs

9. Development Plan
10. Values of Auxiliary Constants

3 General System Description

This section provides general information about the system. It identifies the interfaces between the system and its environment, describes the user characteristics and lists the system constraints. [This text can likely be borrowed verbatim. —TPLT]

[The purpose of this section is to provide general information about the system so the specific requirements in the next section will be easier to understand. The general system description section is designed to be changeable independent of changes to the functional requirements documented in the specific system description. The general system description provides a context for a family of related models. The general description can stay the same, while specific details are changed between family members. —TPLT]

3.1 System Context

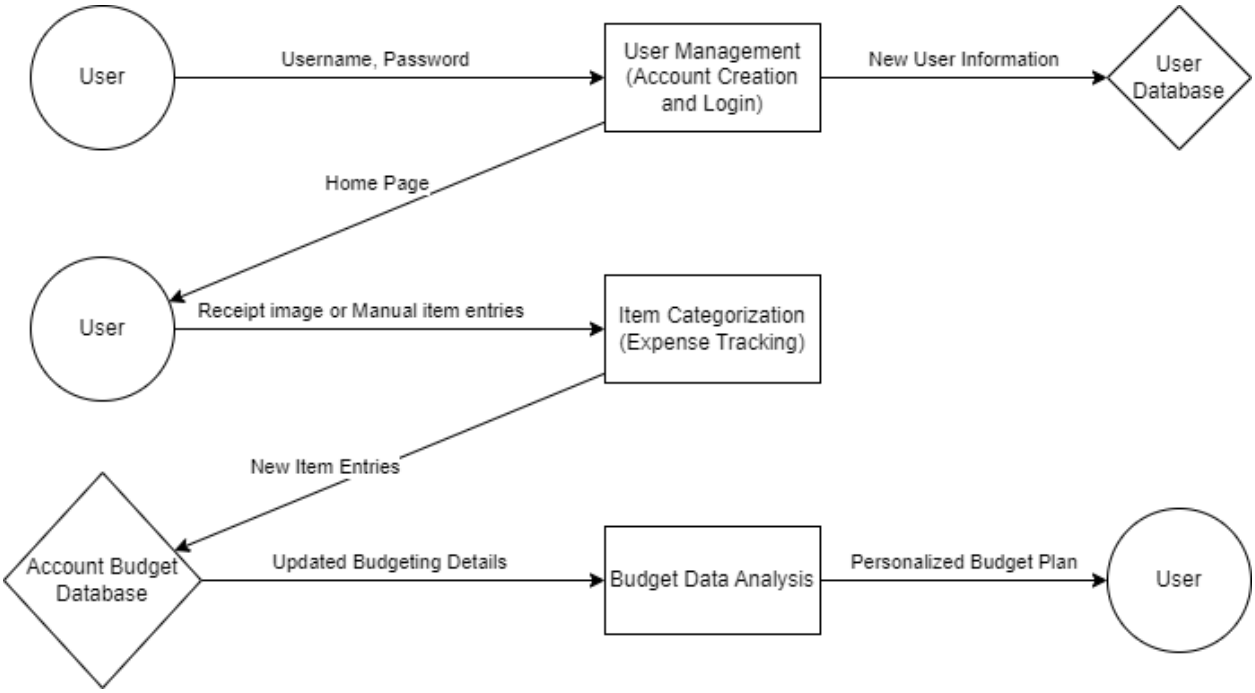

Figure 1: System Context

The system context for this project consists of a user(s), an expense tracking system, a budgeting analysis generator, and several databases. The user provides expense tracking inputs, such as physical/digital receipts or manual expense entries, and receives financial assessments. The expense tracking system processes user inputs, generates budgetary evaluations, and interacts with the financial analysis program and databases. The expense tracking system is also responsible for ensuring the validity of all inputs and provides a response if an invalid input is identified. The budgeting analysis generator provides expense categorization and evaluation functionalities, such as data analysis and visualization, and supports

the expense tracking system in generating outputs. The database will be used to store user account information, user budget data, and supports the expense tracking system in storing and retrieving specific items and their respective categories. The system will typically be used for personal finance management, budgeting, and expense tracking, and may also be used for educational purposes. While the system is not safety-critical, it is important to ensure that the system is reliable and accurate in its calculations and analysis to provide users with trustworthy insights into their financial situation.

[Your system context will include a figure that shows the abstract view of the software. Often in a scientific context, the program can be viewed abstractly following the design pattern of Inputs → Calculations → Outputs. The system context will therefore often follow this pattern. The user provides inputs, the system does the calculations, and then provides the outputs to the user. The figure should not show all of the inputs, just an abstract view of the main categories of inputs (like material properties, geometry, etc.). Likewise, the outputs should be presented from an abstract point of view. In some cases the diagram will show other external entities, besides the user. For instance, when the software product is a library, the user will be another software program, not an actual end user. If there are system constraints that the software must work with external libraries, these libraries can also be shown on the System Context diagram. They should only be named with a specific library name if this is required by the system constraint. —TPLT]

Figure 2: System Context

[For each of the entities in the system context diagram its responsibilities should be listed. Whenever possible the system should check for data quality, but for some cases the user will need to assume that responsibility. The list of responsibilities should be about the inputs and outputs only, and they should be abstract. Details should not be presented here. However, the information should not be so abstract as to just say "inputs" and "outputs". A summarizing phrase can be used to characterize the inputs. For instance, saying "material properties" provides some information, but it stays away from the detail of listing every required properties. —TPLT]

- User Responsibilities:
  - –
- Plutos Responsibilities:

- Detect data type mismatch, such as a string of characters instead of a floating point number
-

[Identify in what context the software will typically be used. Is it for exploration? education? engineering work? scientific work?. Identify whether it will be used for mission-critical or safety-critical applications. —TPLT] [This additional context information is needed to determine how much effort should be devoted to the rationale section. If the application is safety-critical, the bar is higher. This is currently less structured, but analogous to, the idea to the Automotive Safety Integrity Levels (ASILs) that McSCert uses in their automotive hazard analyses. —TPLT]

[The —SS]

3.2 User Characteristics

The users of the *Plutos* budgeting application can be grouped into four main categories based on their financial experience and needs. These groups represent varying levels of financial literacy and desired interaction with budgeting tools.

1. **First and Second-Year Undergraduate Students [Primary]**
   This group consists of young adults who are new to managing their finances independently, such as those living away from home for the first time. The characteristics of these users are as follows:

   - **Financial Knowledge:** Limited experience with budgeting, managing expenses like rent and groceries. Usually have limited financial independence.
   - **Time Management:** Busy schedules juggling school, work, and social commitments. Likely to prefer simple, intuitive interfaces that reduce time spent on budgeting.
   - **Pain Points:** Lack of education on budgeting, potential to overspend due to unfamiliarity with managing day-to-day expenses.

2. **Upper-Year Undergraduate and Post-Graduate Students [Secondary]** This group includes more experienced students who have more financial independence as they have had experience managing their finances (in previous years living independently). They have a better understanding of budgeting and have developed more mature spending habits.

   - **Financial Knowledge:** Some experience balancing school and work, likely to have a clearer understanding of budgeting needs and differentiating between wants and needs.
   - **Time Management:** Likely to have more experience managing limited time and money, though they may still struggle with large financial decisions such as housing or loans.

- **Pain Points:** May underestimate or overestimate budget needs, particularly with larger expenses.

3. **Early Career Professionals** [**Tertiary**] New graduates or individuals recently introduced to the workforce fall into this category. They likely have more stable incomes and financial independence.

    - **Financial Knowledge:** More knowledgeable about saving, budgeting, and managing recurring expenses but still learning to navigate significant financial decisions.
    - **Time Management:** May experience stress due to work-related issues and life changes such as moving to a new city and budgeting/tracking expenses may be another stress inducer on top of the new environment.
    - **Pain Points:** Struggles with planning for long-term financial goals or managing joint accounts with a partner.

4. **Retirees** [**Tertiary**] Although retirees are not a primary focus, they may use the application to simplify financial tracking and planning for a fixed income.

    - **Financial Knowledge:** Likely to have extensive experience with financial management but may struggle to adapt to modern budgeting tools.
    - **Time Management:** Older users may face physical limitations (e.g., difficulty typing or navigating) and slower adaptation to new technologies.
    - **Pain Points:** Difficulty managing finances without a steady income and adapting to increased living expenses.

3.3 System Constraints

C1: **Technical Constraints**

- The system must be developed as a mobile application using a cross-platform framework (e.g., React Native, Flutter) to ensure compatibility with both iOS and Android devices.
- The system must utilize a cloud-based backend technology (e.g., AWS, Firebase) to provide scalability and reliability.
- The system must integrate with a third-party OCR library (e.g., Tesseract OCR, Google Vision API) to provide optical character recognition functionality.
- The system must use a NoSQL database (e.g., MongoDB, Firebase) to store user account information and expense tracking data.

C2: **Performance Constraints**

- The system must be able to process user inputs and generate budgeting data and analysis outputs within a reasonable time frame (e.g., 2-3 seconds).
- The system must be able to handle a minimum of 10 concurrent users without significant performance degradation.

C3: **Security Constraints**

- The system must implement secure password storage and authentication mechanisms using a third-party authentication service (e.g., Auth0, Firebase).
- The system must ensure that all data transmitted between the client and server is encrypted using SSL/TLS protocol.

C4: **Environmental Constraints**

- The system must be designed to operate on a variety of mobile devices, including smartphones and tablets, with different screen sizes and operating systems.
- The system must be able to operate in a variety of network environments, including Wi-Fi, cellular networks, and offline mode.

C5: **Regulatory Constraints**

- The system must comply with relevant data protection regulations (e.g., GDPR, CCPA) to ensure the privacy and security of user data.
- The system must comply with relevant financial regulations (e.g., PCI-DSS) to ensure the secure handling of financial information.

[System constraints differ from other type of requirements because they limit the developers' options in the system design and they identify how the eventual system must fit into the world. This is the only place in the SRS where design decisions can be specified. That is, the quality requirement for abstraction is relaxed here. However, system constraints should only be included if they are truly required. —TPLT]

4 Specific System Description

This section first presents the problem description, which gives a high-level view of the problem to be solved. Following this, a solution characteristics specification is typically included; however this aspect is not applicable for this SRS.

4.1 Problem Description

The problem description can be found in Section 1 of the Problem Statements and Goals document.

4.1.1 Terminology and Definitions

This subsection provides a list of terms that are used in the subsequent sections and their meaning, with the purpose of reducing ambiguity and making it easier to correctly understand the requirements:

- **User Database**: A secure storage location for user information, including usernames, passwords, and personal financial profiles.
- **Item Categorization**: The classification of expenses and income into distinct categories, such as groceries, rent, and salary, to clarify spending habits.
- **Expense Tracking**: The practice of recording and monitoring expenditures to understand spending patterns and make informed financial decisions.
- **Budget Data Analysis**: The examination of financial data related to budgets to identify trends, patterns, and areas for improvement.
- **Personalized Budget Plan**: A customized budget tailored to an individual's specific financial goals, income levels, and spending habits.
- **Account Budget Database**: A system for storing and managing users' budget data, including allocations, spending limits, and financial goals.
- **Digital Financial Tools**: Software applications designed to help users manage their finances, including budgeting, tracking expenses, and analyzing data.
- **Data Security**: Measures taken to protect personal financial information from unauthorized access or breaches.
- **User-Friendly Interface**: An application design that prioritizes ease of use and accessibility, making it simple for users to navigate financial tools.

4.1.2 Physical System Description

The physical system of Plutos, includes the following elements:

User Interface Elements

PS1 **Camera Interface**: A feature that enables users to capture images of receipts using their mobile device's camera.

PS2 **Receipt Preview**: A visual display showing the captured receipt for user verification before processing.

PS3 **Categorization Display**: An interface that shows identified items along with suggested categories for each item.

PS4 **Forms**: Input fields for entering financial data, such as income and expenses.

PS5 **Graphs and Charts**: Visual representations of spending patterns and budget performance, providing insights into financial health.

Data Processing Components

PS6 **Budgeting Algorithm**: An engine that analyzes user data to suggest personalized budgets based on historical spending.

PS7 **Notification System**: A feature that alerts users about important financial events, such as nearing budget limits.

PS8 **Optical Character Recognition (OCR)**: A technology that analyzes the scanned receipt image to extract text, identifying items, prices, and totals.

PS9 **Item Categorization Engine**: A module that automatically classifies extracted items into predefined categories (e.g., groceries, clothing, dining) based on user-defined rules or machine learning algorithms.

PS10 **Database**: A secure storage system for user data, including receipts, categorized items, and budget information.

Users interact with the application by entering financial data, setting budgets, and reviewing visualizations, using the User Interface Elements (PS1-PS5). The application responds by updating financial summaries, processing the data, and providing insights based on user behaviour (PS6-PS10).

4.1.3 Goal Statements

The goal statements can be found in Section 2 of the Problem Statement and Goals document.

4.2 Solution Characteristics Specification

This section is not applicable to the SRS; the solution characteristics are defined through the functional and nonfunctional requirements in Section 5 of this document.

4.2.1 Types

Not applicable.

4.2.2 Scope Decisions

Not applicable.

4.2.3 Modelling Decisions

Not applicable.

4.2.4 Assumptions

Not applicable.

4.2.5 General Definitions

Not applicable.

4.2.6 Data Definitions

Not applicable.

4.2.7 Data Types

Not applicable.

4.2.8 Instance Models

Not applicable.

4.2.9 Input Data Constraints

Not applicable.

4.2.10 Properties of a Correct Solution

Not applicable.

5 Requirements

[The requirements refine the goal statement. They will make heavy use of references to the instance models. —TPLT]

This section provides the functional requirements, the business tasks that the software is expected to complete, and the nonfunctional requirements, the qualities that the software is expected to exhibit.

5.1 Functional Requirements

FR1: **User Account Management**

- Users must create an account using a name, email address, and password.
- Users can update their information after account creation.
- Users must log in using their registered email and password.

FR2: **Receipt Scanning Input**

- Users can take a picture of a receipt and upload it.
- The system displays a preview for confirmation or retaking.

FR3: **Manual Receipt Input**

- Users can manually input receipt details such as date and items.
- The system validates the input to ensure required fields are completed.

FR4: **Database Management**

- The system shall maintain a secure, account-specific database for storing identified products and their respective categories.

FR5: **Item Recognition and Categorization**

- The system shall identify and display item names, quantities, and costs from the uploaded receipt image.
- The system shall sort the entered products into predefined categories.
- The system shall prompt the user to confirm identified items and their corresponding categories.
- The system shall allow the user to modify the name, quantity, price, and category of the identified items.
- The system shall save the extracted data from the receipt input and the image of the receipt to the user's profile.

FR6: **Financial Tracking**

- The system shall generate an overview of the user's spending history, including total spending by item category and trends over time.
- Users shall be able to set up budgets for each spending category.
- Users shall be able to view their spending in relation to their set budgets.
- The system shall notify users when they reach 80% of their set budget limit.
- The system shall notify users when they achieve specified savings goals.

R1: [Requirements for the inputs that are supplied by the user. This information has to be explicit. —TPLT]

R2: [It isn't always required, but often echoing the inputs as part of the output is a good idea. —TPLT]

R3: [Calculation related requirements. —TPLT]

R4: [Verification related requirements. —TPLT]

[Every IM should map to at least one requirement, but not every requirement has to map to a corresponding IM. —TPLT]

5.2 Nonfunctional Requirements

NFR1: **Accuracy**

- **Receipt Parsing Accuracy:** The machine learning model must achieve at least 80% accuracy in correctly identifying and extracting key information (e.g., item names, prices, dates) from receipts.
- **Data Categorization Precision:** The app should categorize expenses into predefined categories (e.g., groceries, utilities, entertainment) with at least 90% precision to ensure users' financial data is correctly organized.
- **Currency Calculation Precision:** All monetary calculations (totals, budgets, currency conversions) must maintain a precision of up to 2 decimal places to ensure accurate financial reporting.
- **OCR Model Accuracy:** The optical character recognition (OCR) model must correctly recognize text from images of receipts with a minimum accuracy rate of 95%, ensuring minimal manual corrections by users.
- **Expense Summation Accuracy:** The app must guarantee 100% accuracy in summing up expenses, incomes, and savings across different periods and categories, ensuring no rounding or summation errors.
- **Data Sync Consistency:** If the app syncs data across multiple devices or platforms, it should ensure 100% consistency of financial data across all instances with no discrepancies or delays.

NFR2: **Usability**

- **User Interface Simplicity:** The app must have a clean, intuitive, and easy-to-navigate interface, allowing users to perform common tasks (e.g., adding expenses, viewing budgets) within 2-3 clicks.
- **Onboarding Process:** New users should be able to complete the account setup and understand core app features within 5 minutes, with interactive tutorials and tooltips provided during the first use.
- **Responsive Design:** The app's interface must be fully responsive, providing an optimal user experience across different devices (smartphones, tablets, desktops) and screen sizes without layout or performance issues.
- **Error Prevention and Recovery:** The app should provide clear, informative error messages, and guide users through corrective actions when incorrect input is detected. Users should be able to recover from errors (e.g., wrong data entry) with no more than 2 steps.
- **Task Completion Time:** Common user tasks, such as adding a new receipt or setting a budget limit, should be completable within 10 seconds on average, assuming all required information is readily available.
- **Minimal Cognitive Load:** Information displayed to users must be concise and relevant, minimizing the cognitive effort needed to understand their financial data. Only essential details should be shown on the main dashboard, with advanced options tucked under menus.
- **Performance:** The app must load within 2 seconds for all major screens and respond to user actions (e.g., adding an entry, generating a report) within 1 second to ensure a smooth and responsive experience.

NFR3: **Maintainability**

- **Continuous Integration (CI) Pipeline:** The project must implement a CI pipeline that automatically runs tests, checks code quality, and verifies builds with every commit, ensuring that code is always in a deployable state.
- **Backward Compatibility:** New releases must maintain backward compatibility with previous versions, ensuring that users can seamlessly update the app without experiencing disruptions in their data or workflows.
- **Dependency Management:** The system should use a dependency management tool (e.g., npm for JavaScript, pip for Python) to track external libraries, and dependencies must be regularly updated to prevent security vulnerabilities and maintain compatibility with new features.
- **Automated Unit Testing Coverage:** At least 80% of the codebase must be covered by automated unit tests to ensure maintainability and minimize the risk of introducing bugs when making changes.

NFR4: **Portability**

- **Cross-Platform Compatibility:** The software should be compatible with Android and iOS mobile devices running the latest software.
- **Cloud-Based Data Storage:** The app must use cloud-based storage solutions (e.g., AWS S3, Google Cloud Storage) for user data to allow seamless access across devices without data loss or duplication.
- **Use of Platform-Agnostic Technologies:** The app should be developed using platform-agnostic frameworks or languages (e.g., React Native, Flutter, or web-based technologies like HTML5) to ensure easy deployment across different operating systems.
- **Portable Data Formats:** All data must be stored in platform-independent formats (e.g., JSON, CSV) for easy transfer and compatibility between devices, databases, and systems.
- **API Compatibility:** Any external APIs or third-party services integrated into the app must support cross-platform usage, ensuring the app can access necessary services from any supported platform.
- **Minimal Platform-Specific Dependencies:** The app should minimize reliance on platform-specific features or libraries, ensuring that any platform-dependent code can easily be replaced or adapted when porting the app to a new environment.

NFR5: **Reusability**

- **Component-Based Architecture:** The app must be developed using reusable components (e.g., UI components, API services), allowing those components to be easily reused across different parts of the app or in future projects.
- **Separation of Concerns:** The app must adhere to the principle of separation of concerns, ensuring that business logic, data access, and presentation layers are kept separate, allowing individual layers to be reused independently.
- **Reusable Libraries and Modules:** Common functionalities (e.g., receipt parsing, authentication, data validation) must be abstracted into reusable libraries or modules that can be shared across multiple applications or systems.
- **Reusable Design Assets:** Design elements (e.g., icons, typography, color schemes) should be created as reusable assets, ensuring they can be reused consistently across multiple projects or platforms.

NFR6: **Understandability**

- **Clear Code Documentation:** All source code must be thoroughly documented using inline comments and external documentation (e.g., README files, docstrings) to ensure that developers can easily understand the purpose and behavior of code components.
- **Consistent Naming Conventions:** Variables, functions, classes, and modules must follow consistent and meaningful naming conventions (e.g., camelCase, snake_case) that clearly describe their functionality and usage, making the code easier to understand.
- **User-Friendly Interface:** The app's user interface (UI) must be designed with simplicity and clarity in mind, using clear labels, icons, and tooltips to guide users through tasks such as adding expenses or reviewing their budgets, reducing confusion.
- **Logical Code Structure:** The app's codebase must be organized logically, with related files and components grouped together in well-defined directories, so developers can easily navigate and locate specific functionality.
- **Readable Code Formatting:** The code must follow industry-standard formatting guidelines (e.g., proper indentation, line length limits) to ensure that it remains readable and easy to follow, both for developers and code reviewers.
- **Clear Error Messages:** The app must provide clear, descriptive error messages (both for users and in logs) that explain the cause of an issue and provide actionable steps to resolve it, improving user and developer understanding.
- **Detailed API Documentation:** Any APIs developed for the app must include detailed and easy-to-understand documentation, describing available endpoints, request/response formats, and example usage scenarios, ensuring that developers can integrate with them easily.
- **Version Control Documentation:** All major changes to the codebase should be accompanied by clear commit messages and detailed changelogs, explaining the purpose of changes and their impact, helping future developers understand the evolution of the project.
- **User-Centric Terminology:** The language and terminology used throughout the app's UI must be aligned with the users' mental models and expectations, using non-technical and familiar terms to enhance understanding.

5.3 Rationale

5.3.1 Scope Decisions

The scope decisions were made with the intention of ensuring that the app delivers value to its target audience while remaining manageable within development constraints. The app focuses on personal budgeting features, including receipt scanning, expense categorization, budget tracking, and financial analysis.

Rationale:

- **User-Centered Design**: The scope focuses on features that users expect from a smart budgeting app, aligning with current market demands for personal finance tools that offer automation (e.g., receipt scanning) and financial insights.
- **Technical Feasibility**: The decision to include features like machine learning for receipt parsing is scoped within the team's ability to implement using existing AI frameworks (e.g., OCR and NLP libraries).
- **Time and Resource Constraints**: By narrowing the scope to core personal finance features, the team ensures that the project can be completed within the specified timeline and budget, avoiding feature creep.
- **Single Currency Support at Launch**: Focusing on a single currency simplifies the design of financial calculations, data presentation, and reporting. Multi-currency support introduces complexity in areas such as exchange rate management, which can be deferred to a later phase.
- **No Offline Mode in the Initial Version**: Implementing an offline mode adds complexity to data synchronization and storage. By focusing on online features initially, the development team can ensure that the core functionalities (e.g., receipt scanning and expense categorization) perform optimally without additional constraints.

5.3.2 Modeling Decisions

The app's architecture follows a modular, service-oriented design, with key components such as receipt scanning, expense categorization, and budgeting reports being developed as independent, reusable modules.

Rationale:

- **Scalability**: A modular approach allows the app to scale more easily by isolating functionalities into distinct services (e.g., a dedicated service for expense categorization). This reduces interdependencies and facilitates future growth.

- **Maintainability**: Modular components are easier to update and test, as changes to one service do not necessarily impact others. This also allows the app to accommodate feature updates and bug fixes more efficiently.
- **Reusability**: By designing reusable components (e.g., UI elements, API services), the app can easily extend its capabilities or integrate with other systems in the future.
- **Use of JSON for Data Exchange**: JSON is lightweight, human-readable, and widely used for web and mobile applications. Its simplicity makes it ideal for transmitting structured data between the client and server, ensuring smooth communication with minimal overhead.

5.3.3 Assumptions

Several assumptions were made during the development and design process. These assumptions include the expectation that users will regularly scan receipts, that mobile users are the primary audience, and that financial data privacy is of utmost importance.

Rationale:

- **Receipt Scanning**: The assumption that users will consistently scan receipts underpins the app's core functionality. This assumption is based on trends observed in personal finance apps where automation is a key value proposition for users.
- **Mobile-First Audience**: It is assumed that the majority of users will access the app via mobile devices. This assumption is driven by the current dominance of mobile platforms for personal finance management, offering convenience and accessibility.
- **Privacy and Security**: The assumption that users expect high standards of data privacy and security informs the app's adherence to regulations such as GDPR. Users handling sensitive financial data expect robust protection mechanisms.
- **Stable Internet Connection**: It is assumed that users will have access to a stable internet connection while using the app, especially for features that require cloud processing (e.g., receipt scanning, data syncing, and report generation).
- **Regular App Usage**: The assumption is that users will interact with the app on a regular basis (e.g., weekly or monthly), allowing the app to provide up-to-date financial insights and budgeting trends based on frequent input.
- **User Financial Literacy**: The app assumes a basic level of financial literacy among its users, meaning that they will be able to understand concepts like budgets, expenses, savings, and categories without extensive in-app education.

5.3.4 Typical Values

For certain features, typical values have been used to inform design and development, such as default budget categories, receipt parsing accuracy thresholds, and system performance benchmarks.

Rationale:

- **Default Categories**: Common budgeting categories (e.g., groceries, utilities, transportation) are provided as default to simplify user onboarding and offer a familiar framework. These categories were selected based on industry standards and user behavior in similar apps.
- **Parsing Accuracy**: An accuracy threshold of 90% for receipt parsing was established as the minimum acceptable standard, based on current machine learning capabilities and user expectations for automated processes.
- **Performance Benchmarks**: Typical values for system performance, such as load times of under 2 seconds for key operations (e.g., viewing reports), are set based on user experience research, ensuring that the app feels responsive and efficient.
- **Maximum Receipt Upload Size**: A limit of 5 MB per receipt ensures a balance between allowing high-resolution images for accurate OCR processing while keeping server storage and bandwidth requirements manageable. Users are encouraged to upload clear, focused images within this size limit.
- **User Transaction History Retention**: Storing up to 3 years of transaction history provides users with enough data to analyze long-term financial trends while limiting the storage requirements for each user. Older transactions could be archived or made available on demand.
- **Frequency of Budget Update Notifications**: Weekly budget update notifications help users stay on top of their spending without overwhelming them with too many alerts. Users are reminded to review their financial status regularly while avoiding notification fatigue.
- **Budget Category Limits**: Allowing up to 25 distinct budget categories provides users with enough flexibility to categorize their spending without overwhelming them with too many options. This value strikes a balance between personalization and simplicity.

[Provide a rationale for the decisions made in the documentation. Rationale should be provided for scope decisions, modelling decisions, assumptions and typical values. —TPLT]

6 Likely Changes

LC1: **Addition of New Features:** Based on user feedback, it is likely that new features (e.g., budgeting goal tracking, automatic expense categorization, or financial analytics) will be added to enhance user experience.

LC2: **User Interface Redesign:** As usability testing is conducted, it's likely that the UI will undergo several iterations to improve the overall user experience and incorporate user feedback.

LC3: **Integration with New APIs:** The app may need to integrate with new financial data APIs (e.g., bank transaction retrieval services) to enhance functionality, requiring changes to the codebase.

LC4: **Change in Tech Stack:** If the team encounters difficulties with the current technology stack, such as performance or compatibility issues, a transition to new technologies (e.g., switching from React to Vue.js) is likely.

LC5: **Performance Optimizations:** As the app scales, there may be a need for performance enhancements, such as optimizing database queries or implementing caching strategies.

LC6: **Data Privacy Compliance Updates:** Changes in legal requirements (e.g., GDPR, CCPA) may require updates to the app's data handling and privacy policies to ensure compliance.

LC7: **Adjustment of User Roles and Permissions:** Changes to user roles and permissions may occur as new features are added, requiring adjustments to the authentication and authorization system.

LC8: **Updates to User Authentication Method:** There may be a transition to a more secure authentication method (e.g., implementing two-factor authentication) based on user feedback and security best practices.

LC9: **Feedback Mechanism for Users:** Adding a feedback mechanism within the app (e.g., surveys or feedback forms) to gather user insights and suggestions for improvements is likely, ensuring continuous enhancement of the app based on user needs.

7 Unlikely Changes

ULC1: **Complete Rewrite of the App:** A complete rewrite of the app's codebase is unlikely unless there are significant fundamental flaws that cannot be addressed through refactoring.

ULC2: **Change in Database Choice:** During the development phase, the initial database choice may need to change due to performance issues or scalability requirements (e.g., moving from SQLite to PostgreSQL or MongoDB).

ULC3: **Switching Platforms:** Transitioning from a mobile-first approach to a purely desktop application is unlikely if the initial target audience is primarily mobile users.

ULC4: **Adopting a New Programming Language:** Changing the programming language for the entire project (e.g., from JavaScript to Ruby) midway through development is unlikely due to the complexity and resource investment required.

ULC5: **Elimination of Existing Features:** Removing core features that are central to the app's purpose (e.g., expense tracking) is unlikely, as these are fundamental to user expectations and functionality.

ULC6: **Change in Target Audience:** A shift in the target audience (e.g., from personal budgeting to corporate finance management) is unlikely as it would require a fundamental reevaluation of the app's design and features.

ULC7: **Discontinuation of Data Storage:** Completely removing any form of data storage (e.g., opting for a stateless application) is unlikely, as the core functionality relies on data retention for budgeting purposes.

ULC8: **Pivot to a Social Networking Feature Set:** A major pivot to add social networking features (e.g., allowing users to connect with friends or share budgets) is unlikely, as it diverges from the core functionality of personal budgeting and may complicate the app's primary purpose.

8 Traceability Matrices and Graphs

The purpose of the traceability matrices is to provide easy references on what has to be additionally modified if a certain component is changed. Every time a component is changed, the items in the column of that component that are marked with an "X" may have to be modified as well. Table 1 shows the dependencies of theoretical models, general definitions, data definitions, and instance models with each other. Table 2 shows the dependencies of instance models, requirements, and data constraints on each other. Table 3 shows the dependencies of theoretical models, general definitions, data definitions, instance models, and likely changes on the assumptions.

[You will have to modify these tables for your problem. —TPLT]

[The traceability matrix is not generally symmetric. If GD1 uses A1, that means that GD1's derivation or presentation requires invocation of A1. A1 does not use GD1. A1 is "used by" GD1. —TPLT]

[The traceability matrix is challenging to maintain manually. Please do your best. In the future tools (like Drasil) will make this much easier. —TPLT]

	TM??	TM??	TM??	GD??	GD??	DD??	DD??	DD??	DD??	IM??	IM??	IM?
TM??												
TM??			X									
TM??												
GD??												
GD??	X											
DD??				X								
DD??				X								
DD??												
DD??								X				
IM??					X	X	X				X	
IM??					X		X		X	X		
IM??		X										
IM??		X	X				X	X	X		X	

Table 1: Traceability Matrix Showing the Connections Between Items of Different Sections

The purpose of the traceability graphs is also to provide easy references on what has to be additionally modified if a certain component is changed. The arrows in the graphs represent dependencies. The component at the tail of an arrow is depended on by the component at the head of that arrow. Therefore, if a component is changed, the components that it points to should also be changed. Figure **??** shows the dependencies of theoretical models, general definitions, data definitions, instance models, likely changes, and assumptions on each other.

	IM??	IM??	IM??	IM??	4.2.9	R??	R??
IM??		X				X	X
IM??	X			X		X	X
IM??						X	X
IM??		X				X	X
R??							
R??						X	
R??					X		
R2	X	X				X	X
R??	X						
R??		X					
R??			X				
R??				X			
R4			X	X			
R??		X					
R??		X					

Table 2: Traceability Matrix Showing the Connections Between Requirements and Instance Models

	A??	A??	A??	A??	A??	A??	A??	A??	A??	A??	A??	A??	A??	A??	A??	A??	A??	A??	A??
TM??	X																		
TM??																			
TM??																			
GD??		X																	
GD??			X	X	X	X													
DD??							X	X	X										
DD??			X	X						X									
DD??																			
DD??																			
IM??											X	X		X	X	X			X
IM??												X	X			X	X	X	
IM??														X					X
IM??													X					X	
LC??				X															
LC??								X											
LC??									X										
LC??											X								
LC??												X							
LC??															X				

Table 3: Traceability Matrix Showing the Connections Between Assumptions and Other Items

Figure **??** shows the dependencies of instance models, requirements, and data constraints on each other.

9 Development Plan

The development plan can be found here.

Note that information in this document, especially regarding dates and deadlines, are subject to change and will be updated accordingly.

All functional requirements specified in Section 5.1 are intended for implementation.

Non-functional requirements specified in Section 5.2 are to be followed but may be subject to change or constraints.

10 Values of Auxiliary Constants

There are no symbolic parameters used in this report.

References

W. Spencer Smith. Systematic development of requirements documentation for general purpose scientific computing software. In *Proceedings of the 14th IEEE International Requirements Engineering Conference, RE 2006*, pages 209–218, Minneapolis / St. Paul, Minnesota, 2006. URL http://www.ifi.unizh.ch/req/events/RE06/.

W. Spencer Smith and Nirmitha Koothoor. A document-driven method for certifying scientific computing software for use in nuclear safety analysis. *Nuclear Engineering and Technology*, 48(2):404–418, April 2016. ISSN 1738-5733. doi: http://dx.doi.org/10.1016/j.net.2015.11.008. URL http://www.sciencedirect.com/science/article/pii/S1738573315002582.

W. Spencer Smith and Lei Lai. A new requirements template for scientific computing. In J. Ralyté, P. Ågerfalk, and N. Kraiem, editors, *Proceedings of the First International Workshop on Situational Requirements Engineering Processes – Methods, Techniques and Tools to Support Situation-Specific Requirements Engineering Processes, SREP'05*, pages 107–121, Paris, France, 2005. In conjunction with 13th IEEE International Requirements Engineering Conference.

W. Spencer Smith, Lei Lai, and Ridha Khedri. Requirements analysis for engineering computation: A systematic approach for improving software reliability. *Reliable Computing, Special Issue on Reliable Engineering Computation*, 13(1):83–107, February 2007.

W. Spencer Smith, John McCutchan, and Jacques Carette. Commonality analysis for a family of material models. Technical Report CAS-17-01-SS, McMaster University, Department of Computing and Software, 2017.

[The following is not part of the template, just some things to consider when filing in the template. —TPLT]
[Grammar, flow and LaTeXadvice:

- For Mac users `*.DS_Store` should be in `.gitignore`
- LaTeX and formatting rules
  - Variables are italic, everything else not, includes subscripts (link to document)
    - Conventions
    - Watch out for implied multiplication
  - Use BibTeX
  - Use cross-referencing
- Grammar and writing rules
  - Acronyms expanded on first usage (not just in table of acronyms)
  - "In order to" should be "to"

—TPLT]
[Advice on using the template:

- Difference between physical and software constraints
- Properties of a correct solution means *additional* properties, not a restating of the requirements (may be "not applicable" for your problem). If you have a table of output constraints, then these are properties of a correct solution.
- Assumptions have to be invoked somewhere
- "Referenced by" implies that there is an explicit reference
- Think of traceability matrix, list of assumption invocations and list of reference by fields as automatically generatable
- If you say the format of the output (plot, table etc), then your requirement could be more abstract

—TPLT]

Appendix — Reflection

[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. How many of your requirements were inspired by speaking to your client(s) or their proxies (e.g. your peers, stakeholders, potential users)?

4. Which of the courses you have taken, or are currently taking, will help your team to be successful with your capstone project.

5. What knowledge and skills will the team collectively need to acquire to successfully complete this capstone project? Examples of possible knowledge to acquire include domain specific knowledge from the domain of your application, or software engineering knowledge, mechatronics knowledge or computer science knowledge. Skills may be related to technology, or writing, or presentation, or team management, etc. You should look to identify at least one item for each team member.

6. 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?