Software Requirements Analysis and Design <T108>

Christian Do
Nhu Ly
Trang Nguyen
Adam Simcoe
Nhan Tran

COMP 3059: Capstone Project I

Professor Laily Ajellu

November 8, 2024

Contents

1.0 Introduction

BUDGET BOSS is a comprehensive financial budgeting application designed to simplify personal finance management. Through software requirements analysis, the app identifies essential user requirements, including income tracking, expense categorization, and budget customization. The intuitive design of the system provides a user-friendly interface, offering a visually engaging dashboard to monitor financial patterns and facilitate informed decision-making. Secure, profile-based storage ensures data accessibility across devices, while potential API integrations enable localized tax considerations, enhancing the budgeting relevance. Consequently, BUDGET BOSS serves as an accessible and versatile financial tool tailored to meet the specific budgeting needs of individuals.

1.1 Purpose

This document outlines the high-level software requirements for the BUDGET BOSS financial budgeting system. It defines the essential features and functionalities necessary to meet user needs without specifying technical implementation details. This document provides clarity for stakeholders, including developers, designers, and end-users, by detailing the system's primary capabilities, such as income and expense tracking, budget customization, and secure data storage. This requirements analysis serves as a guiding framework for subsequent stages of software design and development, ensuring that all project components align with user expectations and the overarching objective of developing an intuitive and accessible budgeting tool.

1.2 Scope

The scope of the *BUDGET BOSS* system defines the project's boundaries, detailing the core features that will be included and the objectives the system aims to achieve in its initial version. This scope ensures alignment with the project plan, focusing on deliverables that will make personal finance management easy and intuitive. By clarifying what the system will and will not do, the scope outlines key functionalities for development and establishes realistic expectations for stakeholders.

In Scope

The following functionalities are included in the project's scope, providing essential benefits for users:

- **Dashboard for Tracking Income and Expenses**: A central view where users can monitor their finances, gaining insights into their spending and earning patterns.
- **Budget Creation and Management Tools**: Features to set up and adjust personalized budgets, promoting responsible financial planning.

- **Notification System**: Timely reminders for user-set saving goals, helping users stay on track with their financial objectives.
- **Data Visualization**: Graphs and charts to display expense and saving patterns clearly, enhancing user understanding of financial habits.
- **User Profile System**: A secure registration system enabling users to save and access their data across devices.
- **Data Security**: Ensure data security by encryption for all sensitive information by applying strong password hashing to protect against authorized access and data breaches.

Out of Scope

The following features are not included in the scope for version 1.0, as they fall outside the immediate goals of the project:

- API Integration for Ontario Tax Service for regional tax service integration, enabling localized budgeting support for Ontario users.
- Tax filing support or broader financial advisory services.
- International currency conversion or exchange rate features.
- Investment platform integration, including stock trading functionalities.
- Direct integration with user banking accounts.
- Corporate financial management tools for business use.

This scope is designed to establish a streamlined, user-friendly budgeting tool that meets personal finance needs for individual users, promoting clear financial awareness and planning.

2.0 System Overview

The System Overview section introduces the system context and design.

2.1 Project Perspective

The BUDGET BOSS system is a new, self-contained financial budgeting tool designed to address the growing need for accessible personal finance management. Unlike complex financial software or corporate budgeting tools, BUDGET BOSS is tailored for individuals seeking a straightforward solution for tracking income, managing expenses, and setting savings goals. As an independent system, it does not build upon or replace any existing financial applications, allowing for a fresh approach focused on simplicity, user-friendly design, and features specific to personal budgeting. The project's origin is rooted in creating an intuitive, stand-alone solution that enhances financial awareness and planning without requiring extensive financial knowledge.

2.2 System Context

The BUDGET BOSS system addresses the need for a simplified, accessible approach to personal finance management within a market where

many users find traditional financial tools overly complex or inaccessible. The app is designed to support users in tracking their income and expenses, setting customized budgets, and visualizing financial trends, empowering individuals to make informed financial decisions. Strategically, BUDGET BOSS aligns with a growing emphasis on financial literacy, offering features that promote regular saving habits and financial accountability. By potentially integrating with regional tax services, the system aims to meet localized needs, making it a relevant tool for Ontario residents and adding value to its business case as a tailored budgeting solution.

2.3 General Constraints

The development of the *BUDGET BOSS* system is subject to several business and technical constraints that impact specification, design, implementation, and testing:

- **Budget and Financial Oversight**: Development and maintenance costs must stay within the allocated budget. Any additional expenses require approval from the Project Manager to maintain financial accountability.
- **Timeline and Project Deadlines**: The project must be completed within the specified timeframe. Any delays in development phases must be communicated promptly to assess impacts on the overall timeline and make necessary adjustments.
- **Technology Stack**: The app will use a predefined technology stack, including React, Node.js, and MongoDB. Changes to this stack require a thorough review and approval process to prevent compatibility and integration issues.
- **Team Availability**: The project relies on the consistent availability of skilled team members. If any critical team member is unavailable for over a week, a reassessment of the timeline and workload distribution will be necessary to prevent delays.
- **Feature Scope**: The initial release must include core features like income tracking and data visualization. Modifications to the feature set require team consensus to avoid disrupting the project timeline and development focus.
- **Third-Party API Dependencies**: Reliable access to third-party APIs is essential for core budgeting functions. Any downtime or modifications in these external services could impact the app's functionality, requiring contingency plans or alternative solutions.
- **Development Methodology**: The project follows Agile methodology, necessitating bi-weekly sprint reviews and daily stand-up meetings to maintain alignment and allow quick adaptation to evolving requirements.
- **Data Protection Compliance**: The system must comply with data protection regulations, such as GDPR, to ensure secure handling of user data. This compliance is crucial to avoid legal issues and ensure user trust, affecting both the design and testing phases.

These constraints set essential boundaries for the project, influencing decisions across all phases and ensuring adherence to project goals, timeline, and budget.

2.4 Assumptions and Dependencies

The following assumptions have been made during the initiation of the BUDGET BOSS project:

- **End-User Profile**: The app will primarily target individual users looking to manage personal finances, excluding businesses or corporate entities.
- **Device Accessibility**: End-users are assumed to have access to smartphones or tablets with internet connectivity for seamless use of the app.
- **User Engagement**: It is assumed that users will regularly interact with the app to track their finances, set savings goals, and make adjustments to their budget.
- **Technology Availability**: The necessary hardware and software technologies required for development, including devices and tools, will be readily accessible and compatible.
- **API Stability**: It is assumed that the Ontario tax service API and budget API (e.g., YNAB) will remain stable and available for integration into the app without significant downtime.
- **Alignment of Deliverables**: The identified project deliverables are assumed to align with the project's objectives, ensuring focus on critical features like budgeting, income tracking, and data visualization.
- **Stakeholder Consensus**: All stakeholders have provided input and have agreed on the project requirements, with no significant changes expected during development.
- **Feasibility of Project Schedule**: The timeline set for the project is assumed to be achievable within the allocated time frames without major disruptions or delays.
- **Accurate Task Dependencies**: Task dependencies will be correctly identified and managed to ensure smooth progression of development and timely delivery.
- **Sufficient Resources**: It is assumed that the project team will have adequate time, skills, and resources to complete all project deliverables successfully.
- **Quality Assurance**: The project will meet or exceed the defined quality standards, ensuring high usability and reliability for end-users.
- **Timely Feedback**: Stakeholders are expected to provide feedback promptly during the project to ensure timely decision-making and adjustments when necessary.

The following dependencies may impact the success or desired result of the project:

- **External Financial APIs**: The project relies on third-party APIs for real-time data, such as the Ontario tax service and budget APIs (e.g., YNAB). Any downtime, changes, or issues with these services could directly affect the functionality of the app.
- **Data Security Compliance**: The app must adhere to data privacy laws (e.g., GDPR) to protect user data and avoid legal risks. Any changes in data security regulations could affect design and implementation.
- **Development Team Capacity**: The project's timeline and success are dependent on the development team's ability to meet deadlines, resolve technical challenges, and maintain quality throughout the project.
- **Marketing and User Onboarding**: The success of the app depends on effective marketing efforts and successful user acquisition, which will directly influence the app's user base and adoption rates.
- **User Feedback Loop**: Continuous iteration based on user feedback is critical for post-launch improvements, meaning ongoing collection and integration of user input is necessary.
- **Operating System and Device Compatibility**: The app's performance and user experience depend on its compatibility with a variety of Android devices and operating system versions. Any issues with compatibility could limit its reach or user satisfaction.
- **App Store Approval**: The release of the app depends on approval from app stores (e.g., Google Play), which could delay the launch if there are any issues or rejections during the review process.

3.0 Functional Requirements

This section describes specific features of the software project. If desired, some requirements may be specified in the use-case format and listed in the Use Cases Section.

3.1.1 Functional Requirement 1: Registering Users

- Introduction
    - The system will provide a feature for new users to register. This feature will collect the user's email, username, and password and create a new user account.
- Inputs
    - Email
    - First Name
    - Last Name
    - Password
    - Confirm Password
- Processing
    - Input Validation: Ensure all fields are filled and valid (email format, password strength)
    - Duplication Check: Verify that the email is unique and not already registered.

    - Data Storage: Save the user's information securely in the database.
    - Confirmation: Send a confirmation toast notification if registration is successful.
- Outputs
    - A success message indicates that the registration was successful.
    - Error messages for any validation failures.
    - Error messages if the email is already registered.

3.1.2 Functional Requirement 2: Adding and Categorizing Income Entry

- Introduction
    - The "Input and Categorize Income" feature allows users to enter and categorize their income entries. This functionality enables users to record income details, categorize entries by type, and maintain an organized view of their income sources, essential for comprehensive financial tracking and analysis.
- Inputs: There are 2 main inputs:
    - Income Amount: The monetary amount received, specified in the chosen currency.
    - Category: The category assigned to the income
- Processing: There are 5 main processes:
    - Data Validation
    - Category Management
    - Data Storage
    - Data calculations
    - Data Presentation
- Outputs There are 3 output-types:
    - Income Summary
    - Graphical Representations
    - Error Messages

3.1.3 Functional Requirement 3: Budget Creation and Management

- Introduction
    - The Budget Creation and Management feature will allow users to create and manage pre-existing budgets within set date timeframes. The goal of this feature is to help users properly track and monitor their income and expenses in relation to their overall budgeting goals.
- Inputs
    - Budget Amount
    - Start Date: The date the budget begins to take place.
    - End Date: The proposed budget end date.
    - Notes: Potential descriptive notes for the budget that are user-defined.

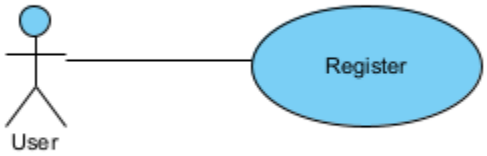
- Processing
  - Income/Expense Existence Validation
  - Budget Creation/Adjustment
  - Budget Validation
  - Budget Data Storage
  - Budget Summary Presentation
- Outputs
  - Budget Summary
  - Validation Errors

...

3.2 Use Cases

3.2.1 Use Case #1: User Registration

Diagram:



Main Scenario:

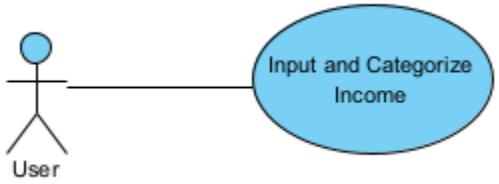
A new user decides to create an account and navigates to the registration page. They fill out the registration form, providing their first and last name, email, and password, along with a confirmation of the password. The system validates the inputs, ensuring that all fields meet specified requirements (e.g., password strength, valid email format). After confirming that the passwords match, the user submits the form. The system then checks for an existing account with the same email. Finding none, it securely saves the user information and sends a confirmation message. The user can now log in and access the app.

Alternative Path:

If the user submits the form but the system detects that the email is already associated with an existing account, an error message informs the user that the email is in use. The user is prompted to either try a different email or, if they already have an account, navigate to the login page. This ensures that duplicate registrations are avoided and encourages the user to use a unique email for account creation.

3.2.2 Use Case #2: Adding a New Income Entry and Categorize

Diagram:



Main Scenario:

When a user wants to track their income, they go to the "Input and Categorize Income" section in the app. They enter details like amount, date, source, and select or create a category.

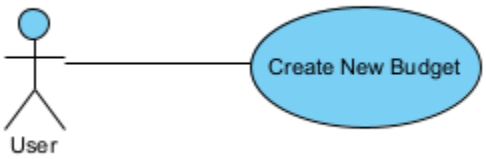
The app validates and saves the entry, updates the income summary, and shows a confirmation. The user can view a categorized summary with graphs for insight into income sources.

Alternative Path:

If there's an error, like a negative amount or missing category, the app prompts the user an error message and offers a way to correct it. Once corrected, the entry is saved, ensuring accurate income tracking.

3.2.3 Use Case #3: Creating a New Budget

Diagram:



Main Scenario:

The user will navigate to the Create/Update Budgets page and select the create option. The system will perform a check to ensure that the user has either an income or expense already existing within the database to build the new budget with. It will then prompt the user to enter details related to the total amount for the budget, any date parameters, and any descriptive notes related to its purpose/goals. It will then validate the budget entry, before saving the entry to the database and using the new entry to produce a detailed budget summary report to the user.

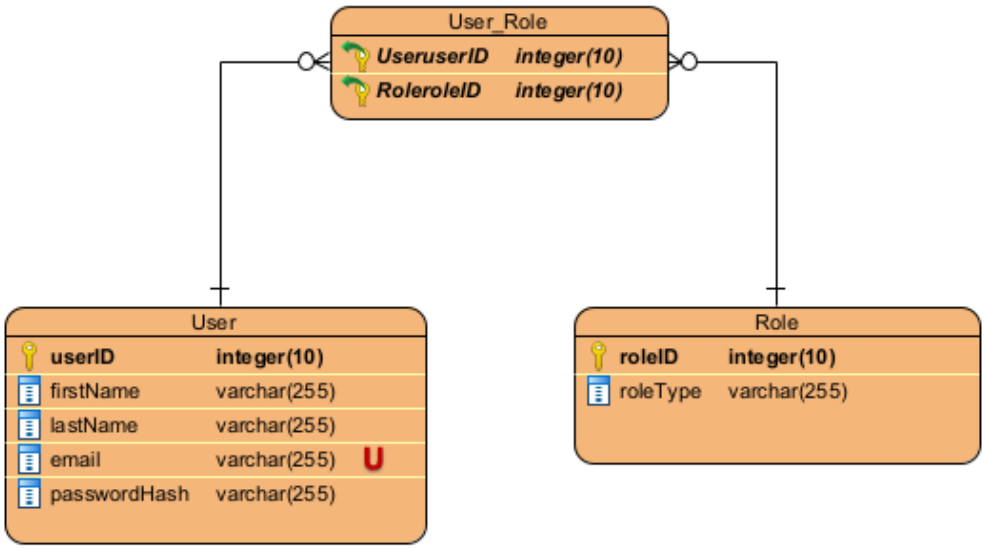
Alternative Path:

In the event any validation checks are failed, such as the income/expense exists check or the validate budget check, the system will produce an error message detailing what went wrong, followed by redirecting the user back to the step the process failed in. This allows the user to re-enter data without losing their place in the overall process.

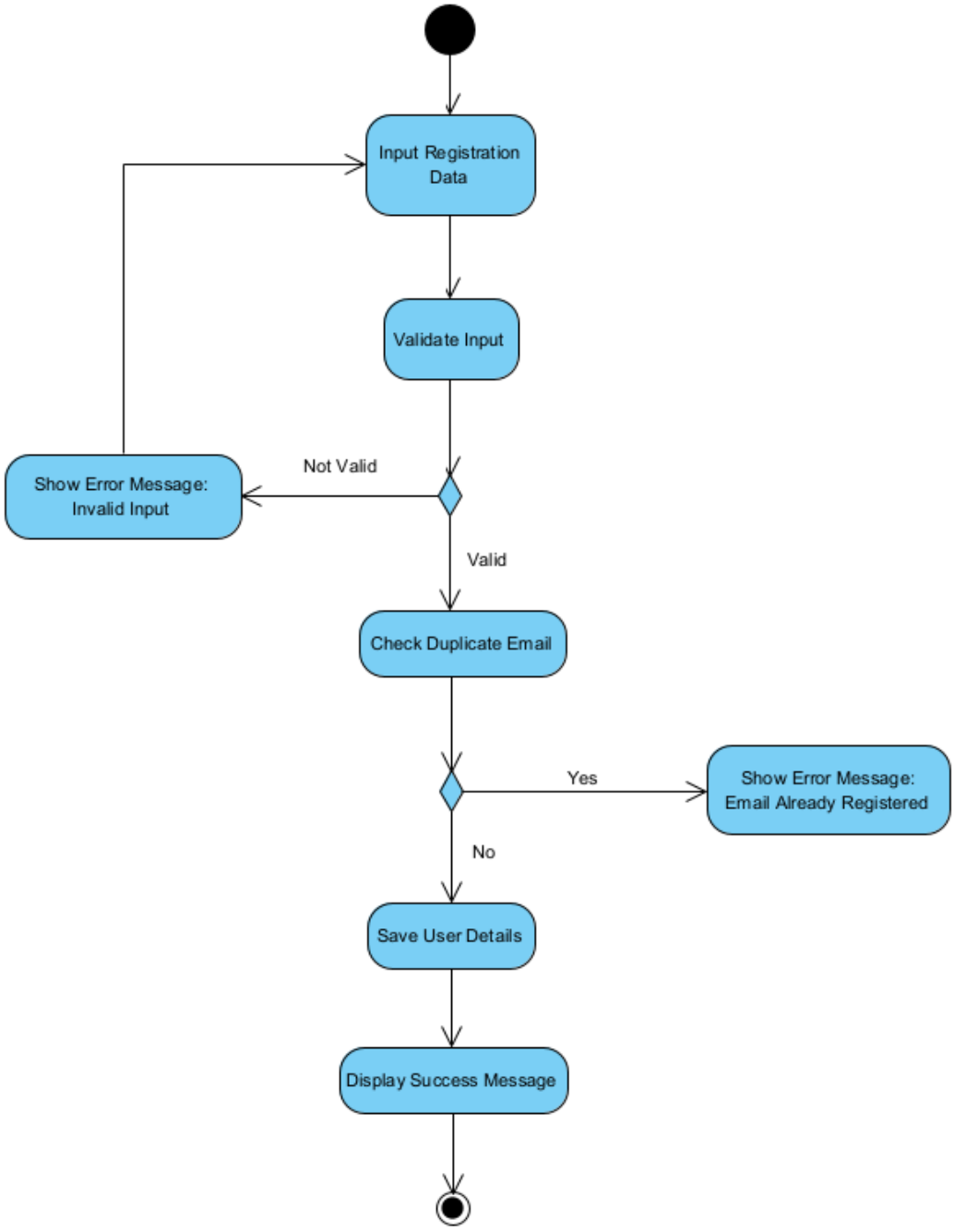
3.3 Data Modelling and Analysis

3.3.1 User Registration Diagrams

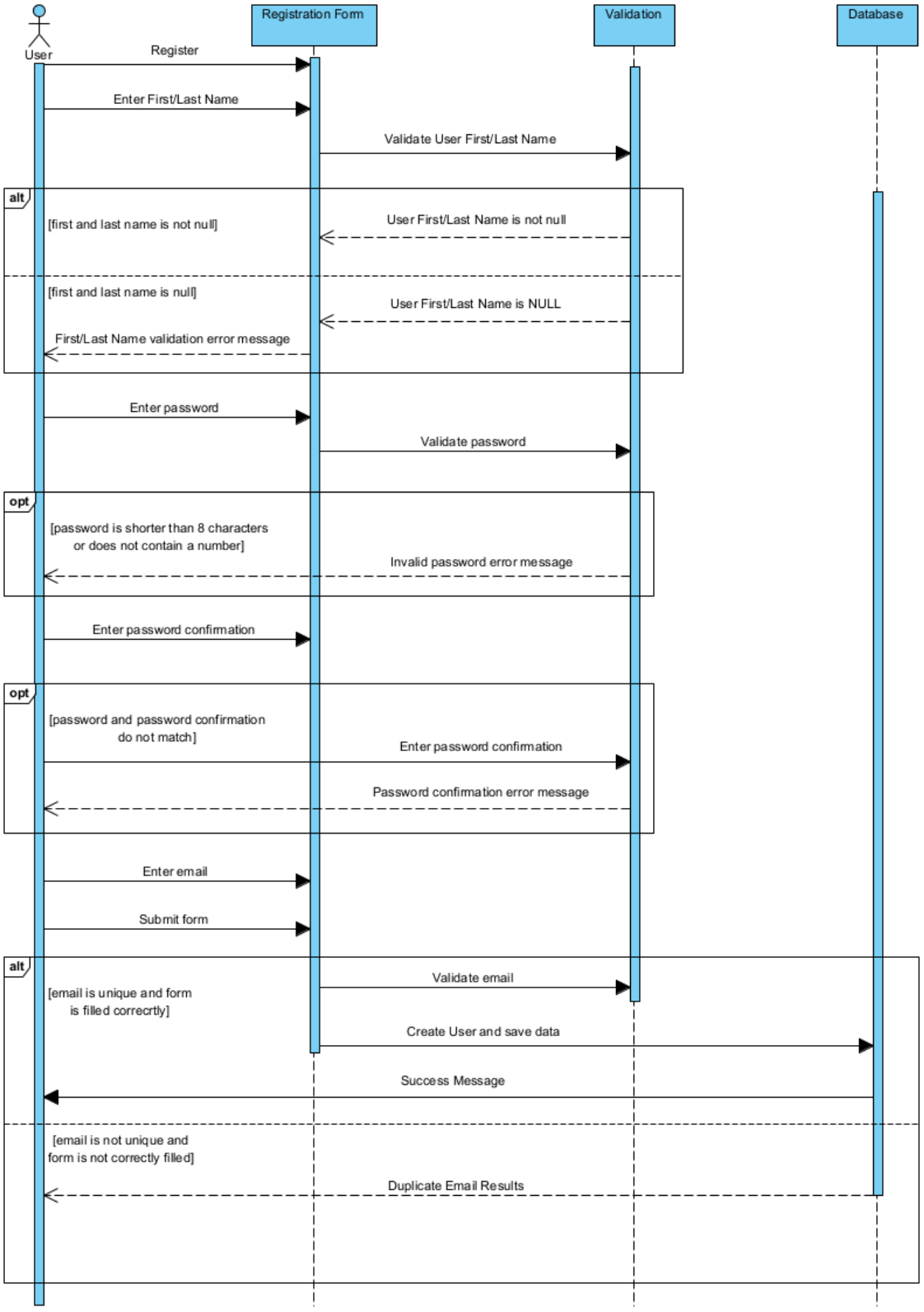
Normalized Data Model Diagram



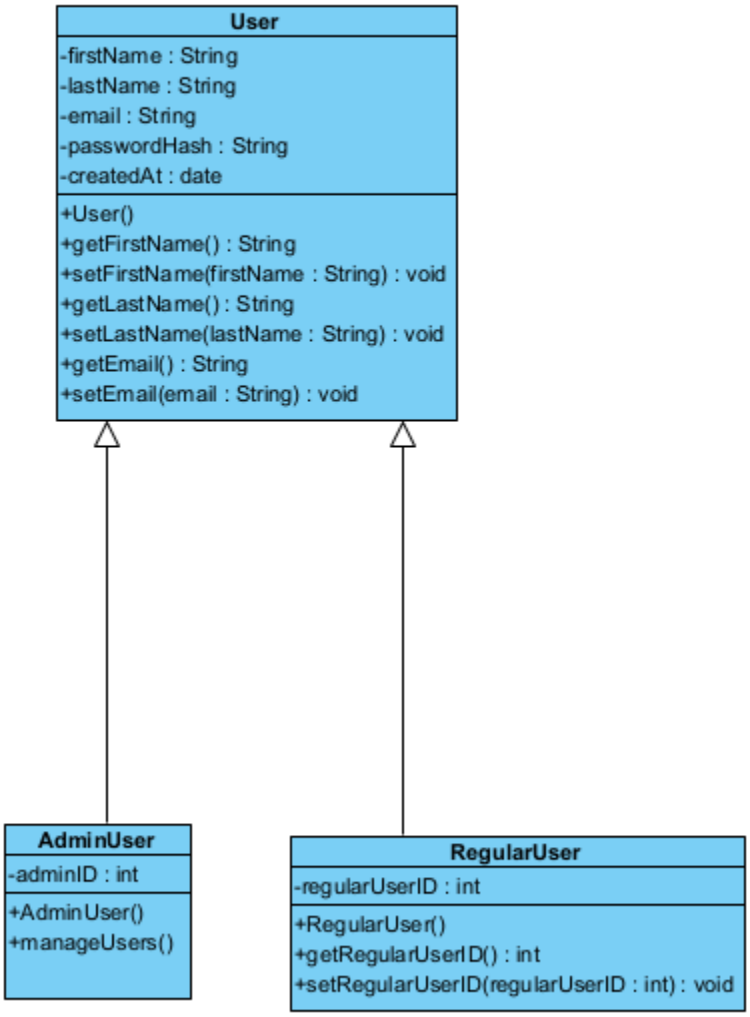
Activity Diagram



Sequence Diagram

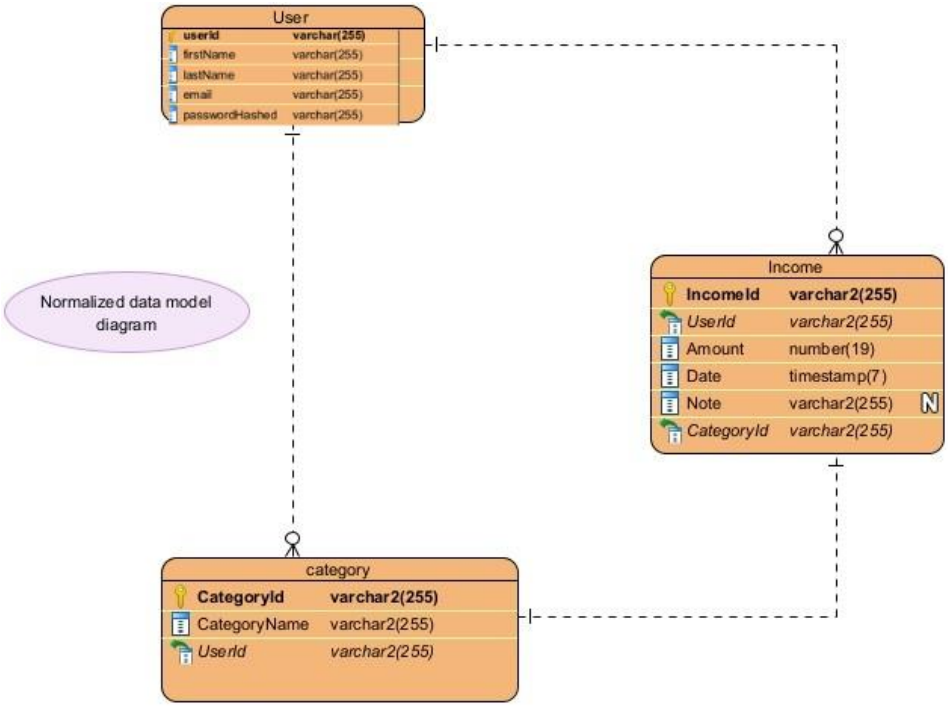


UML Class Diagram

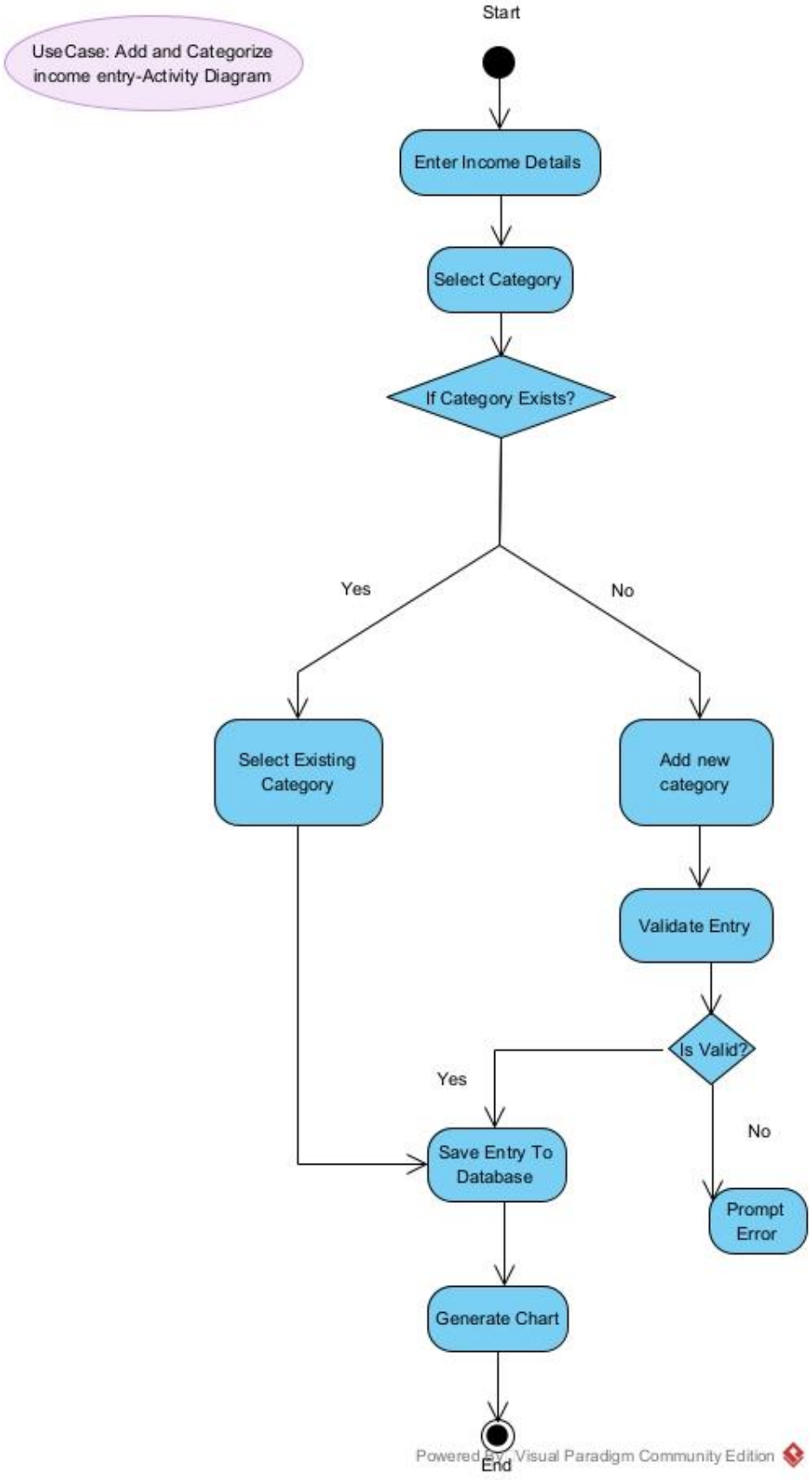


3.3.2 Adding and Categorizing Income Entry Diagrams

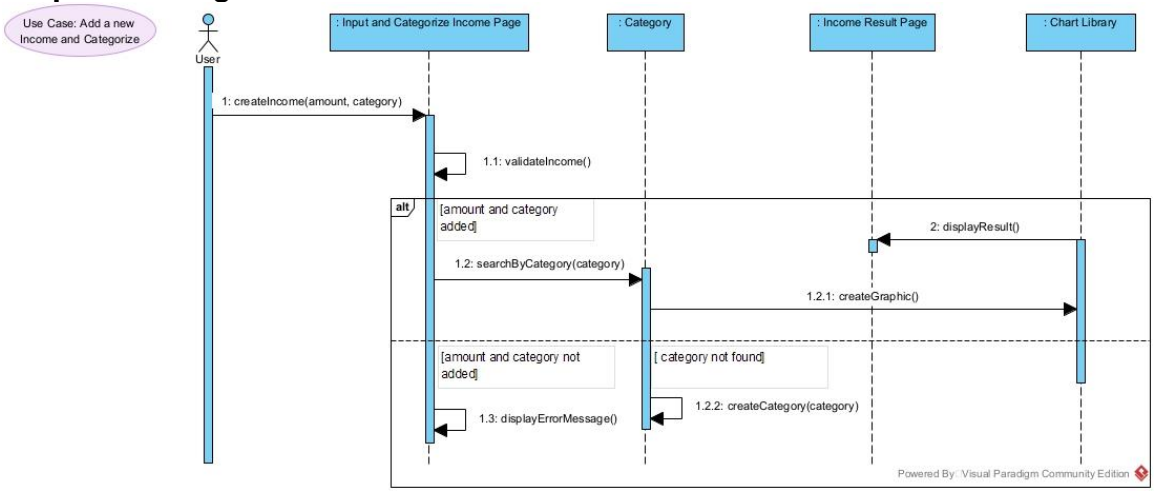
Normalized Diagram



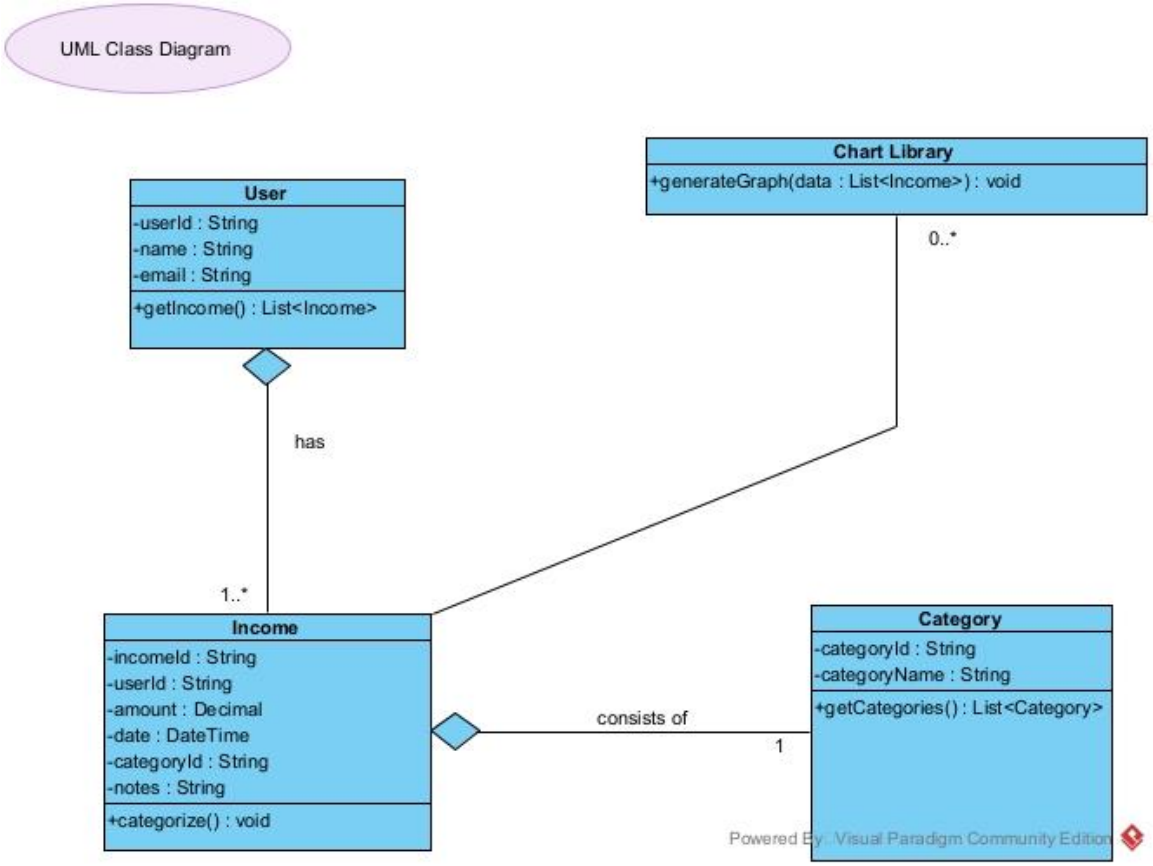
Activity Diagram



Sequence Diagram

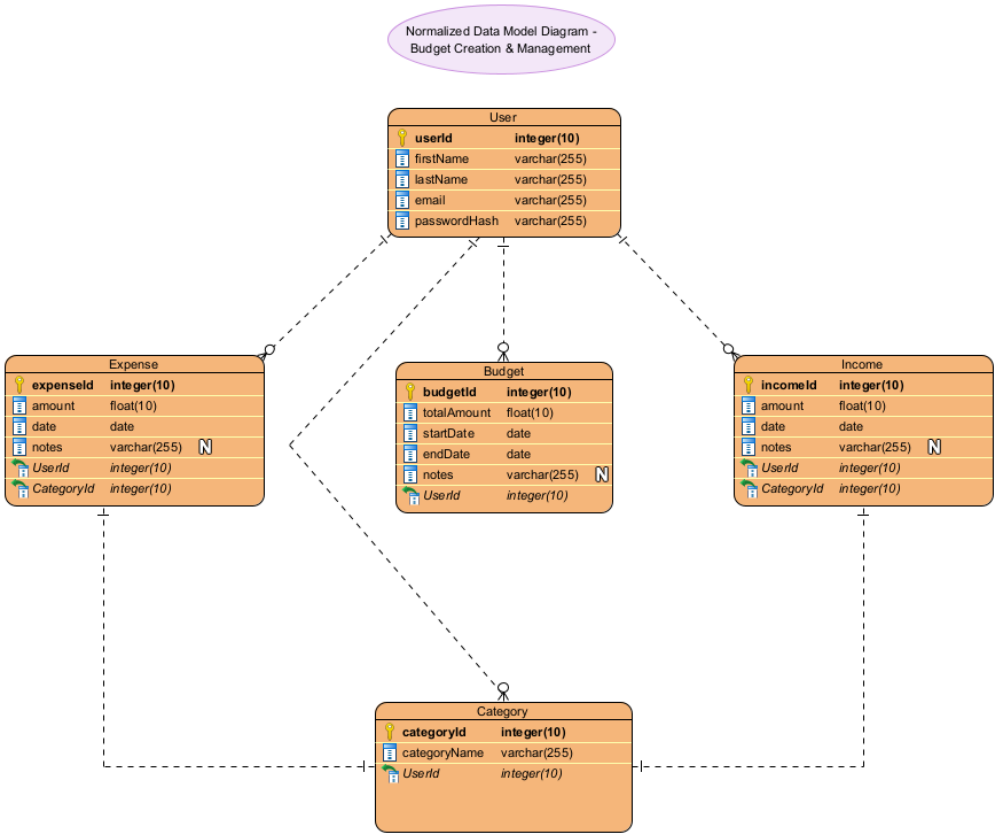


UML Class Diagram

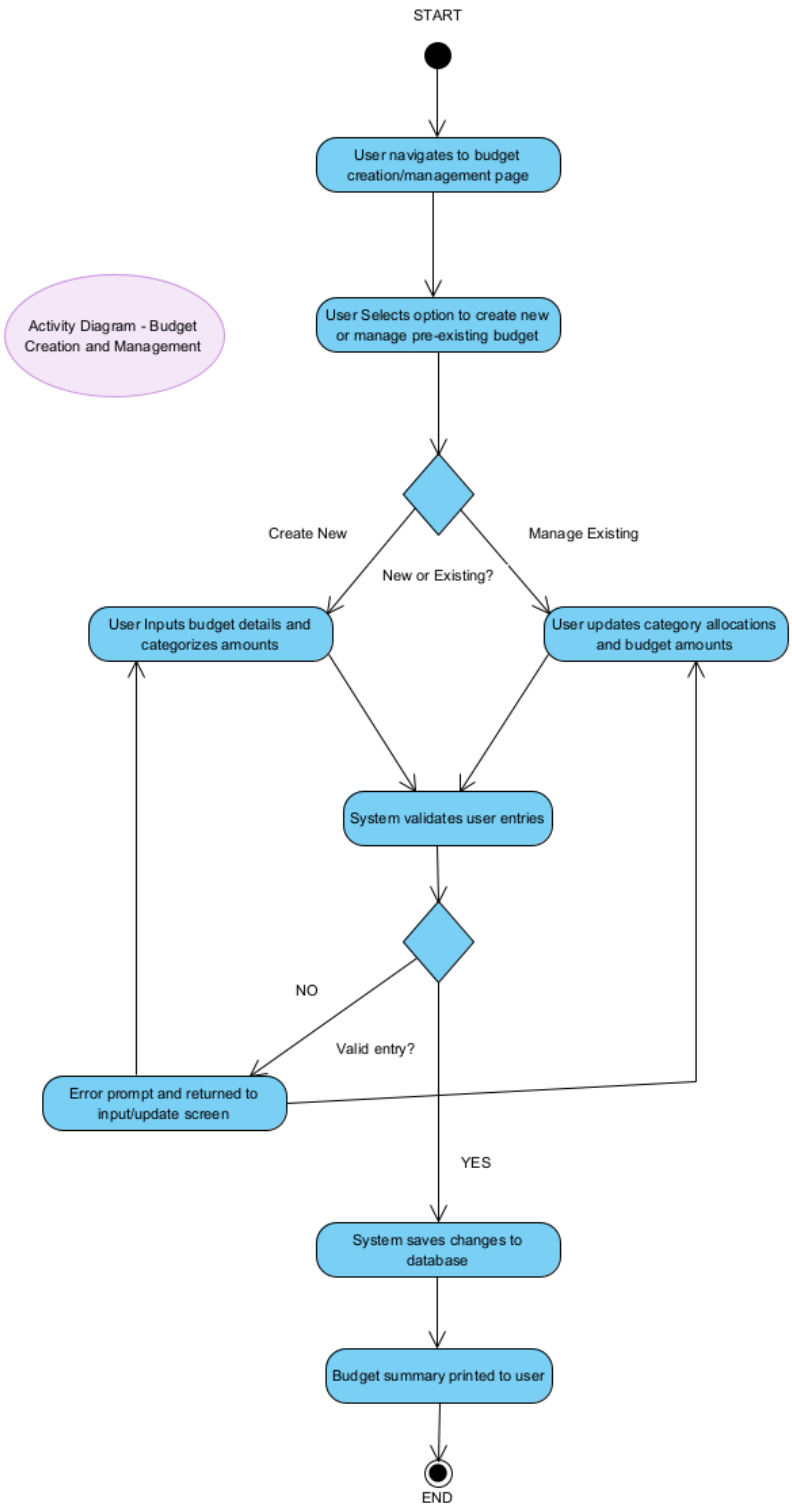


3.3.3 Creating a New Budget Diagrams

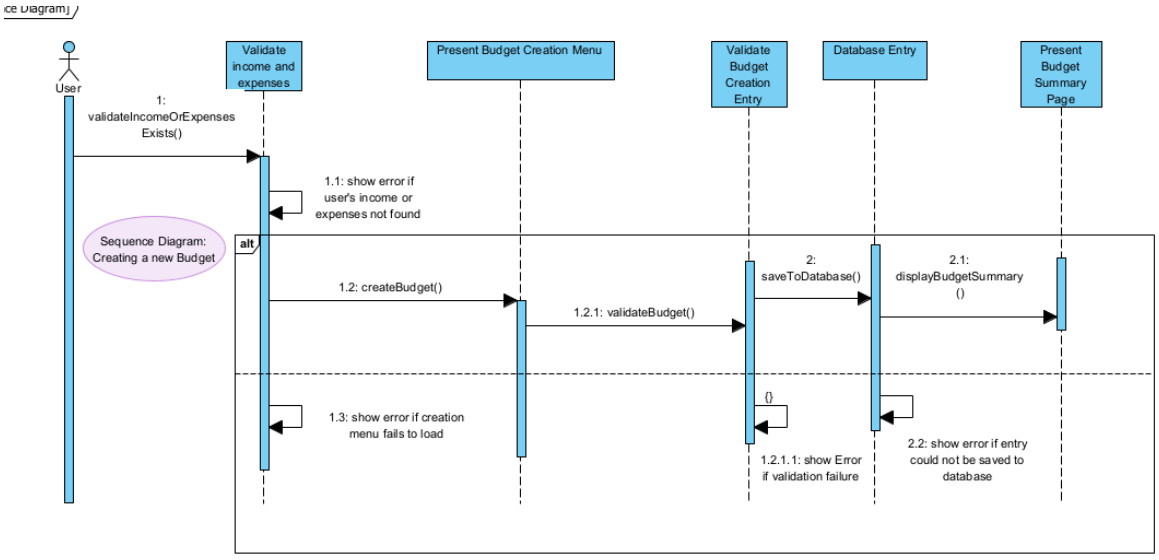
Normalized Diagram



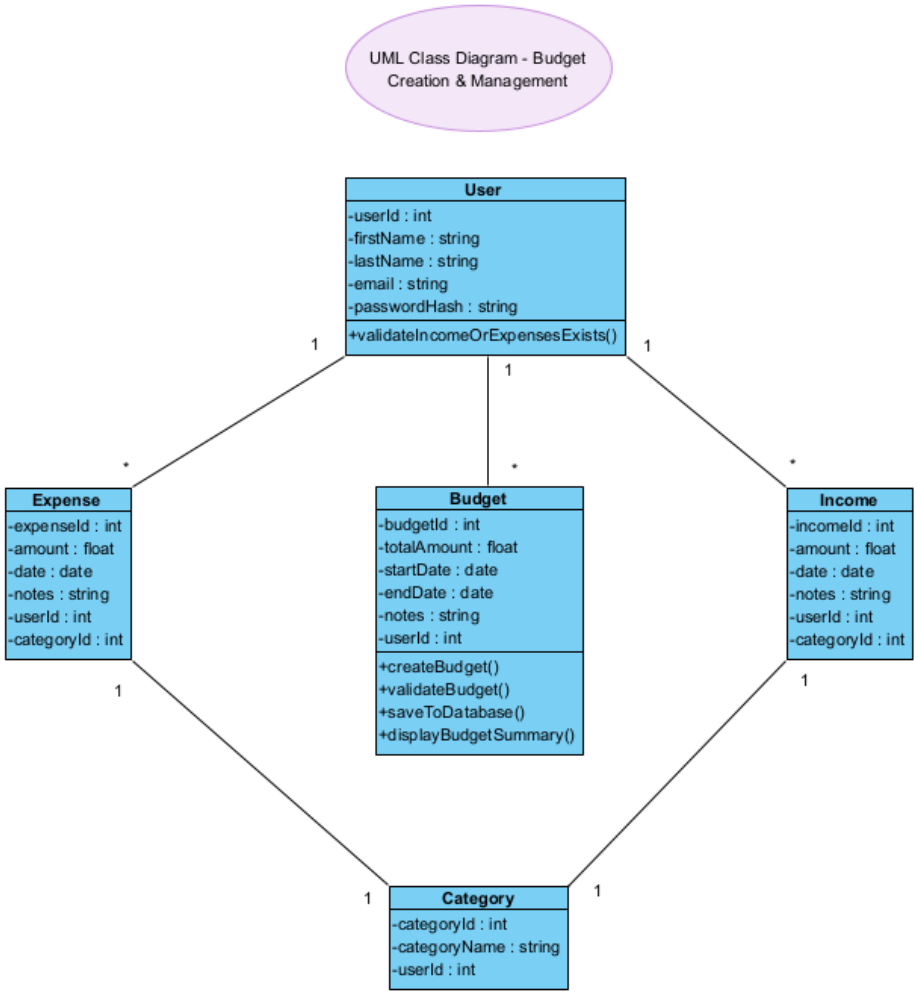
Activity Diagram



Sequence Diagram

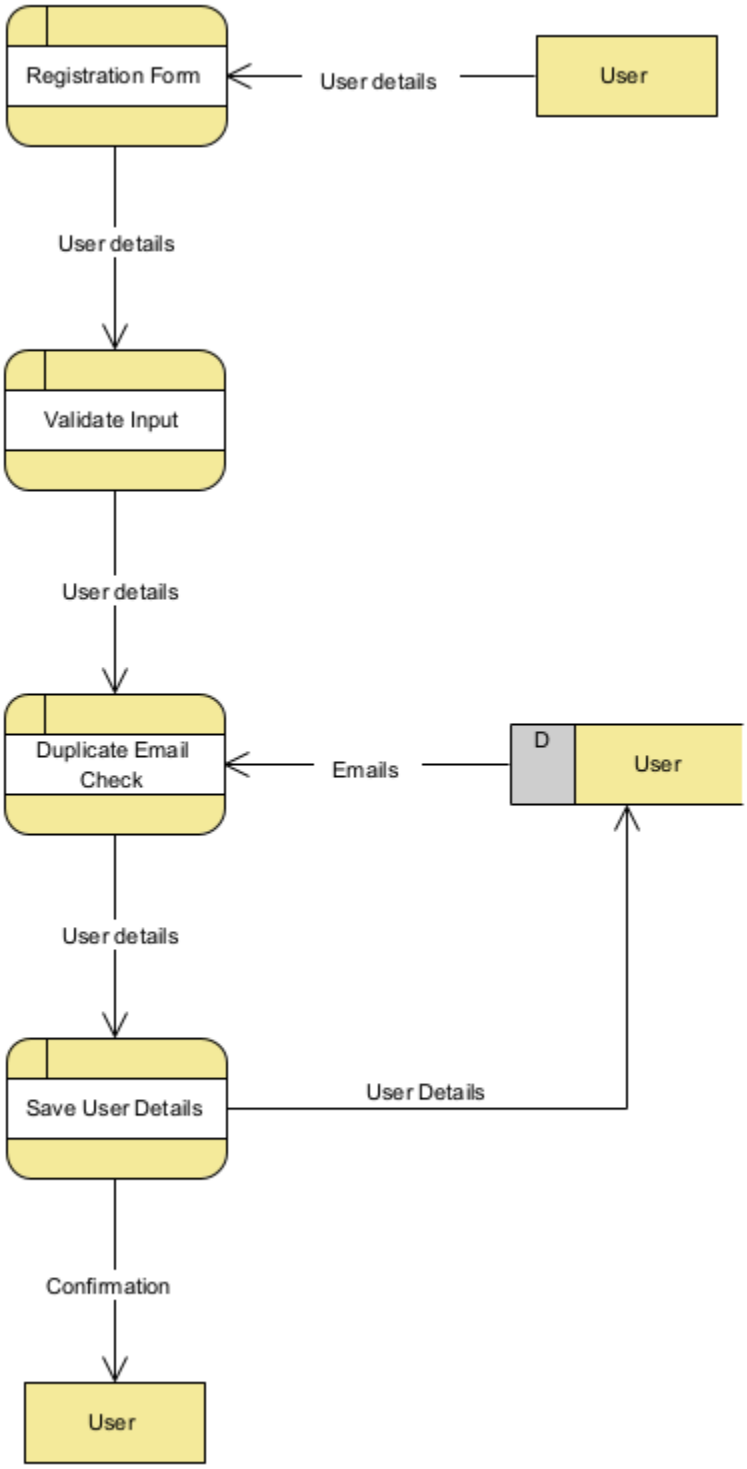


UML Class Diagram

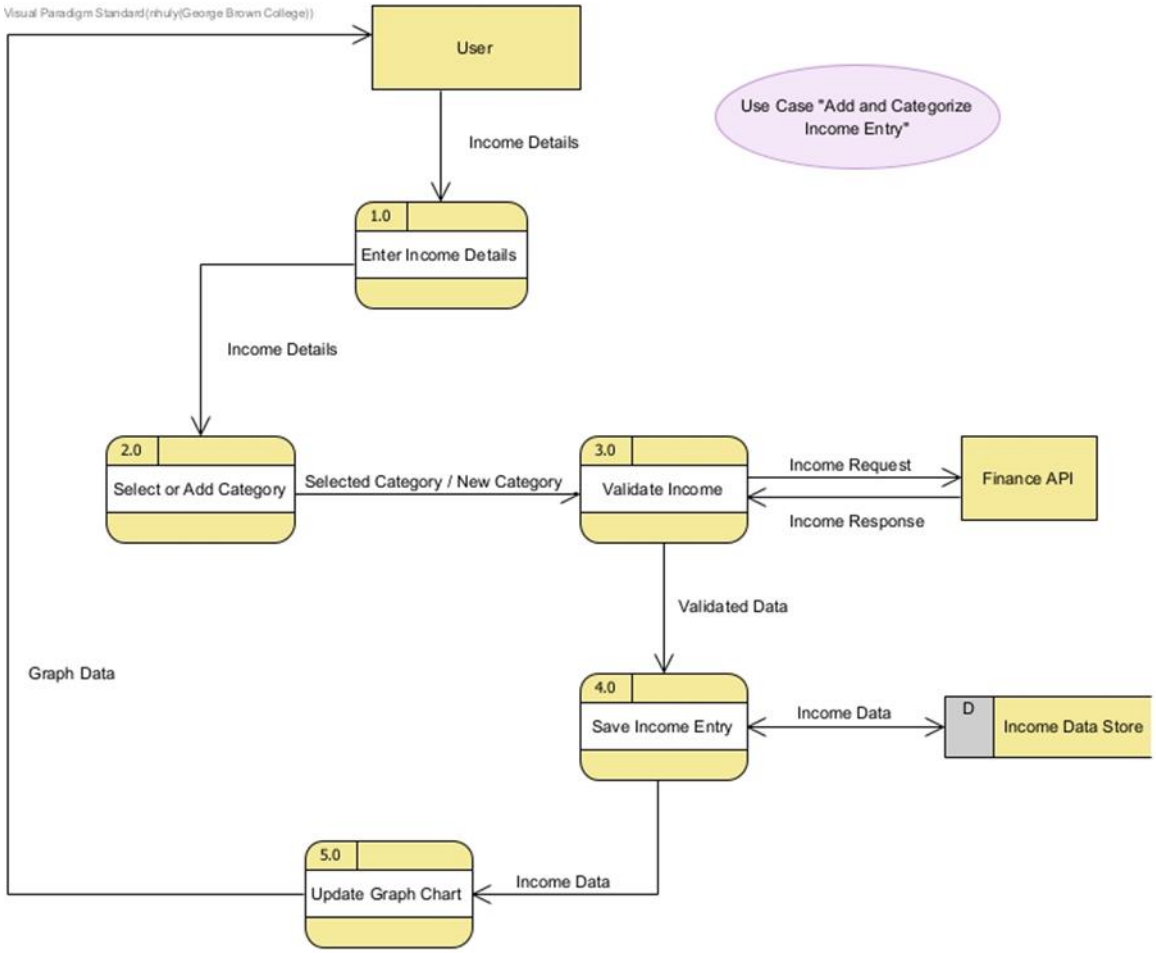


3.4 Process Modelling

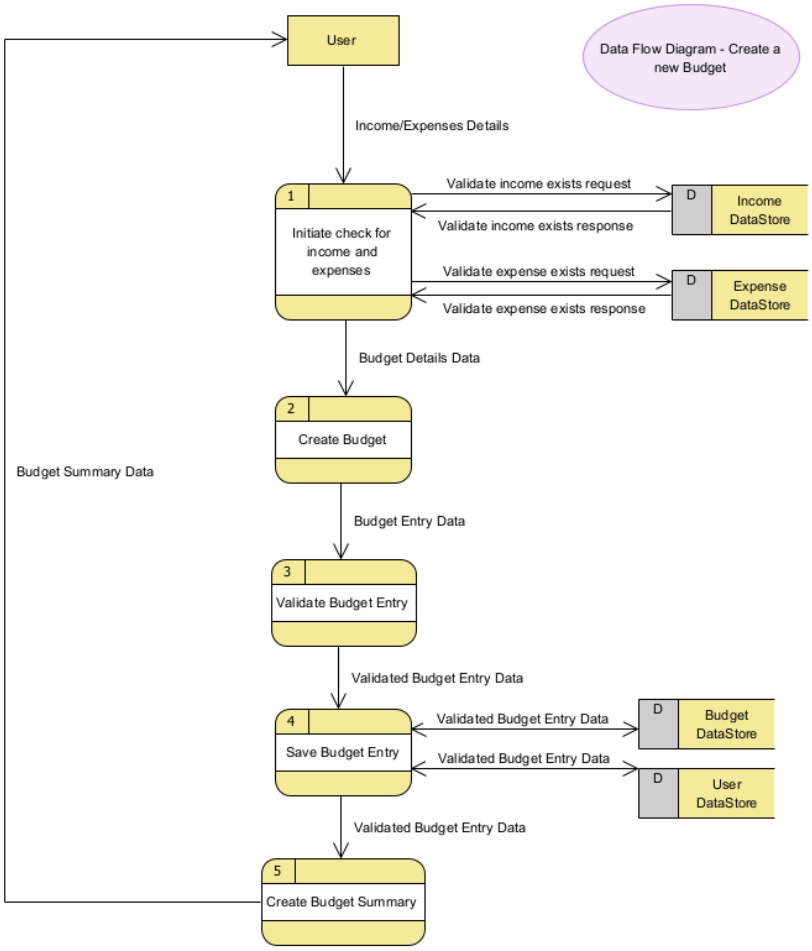
3.4.1 User Registration Data Flow Diagram



3.4.2 Adding & Categorizing Income Entry Data Flow Diagram



3.4.3 Creating a New Budget Data Flow Diagram



4.0 Non-Functional Requirements

- **Performance**:
  - 90% of user inputs (income and expenses) should have a category.
  - Graph rendering and category-based calculations should complete within 3 seconds after data entry.
- **Reliability**:
  - The app should maintain a 99% uptime.
  - Data loss in the event of a crash or unexpected shutdown should be minimal, with automated data backups performed every hour.
- **Availability**:
  - The app should be available 24/7, with scheduled maintenance downtimes not exceeding 30 minutes per week.
- **Security**:
  - Authentication measures, such as password or biometric login, should be implemented to protect sensitive financial data.

- **Maintainability**:
  - The codebase should be modular, allowing easy updates or fixes.
  - Documentation must be maintained to assist future developers in understanding the system.

5.0 Logical Database Requirements

- **Database Use:**
  - A relational database will be used to store user data, including income, expenses, and categorized spending data.
- **Data Formats:**
  - Monetary values should be stored with two decimal places.
  - Dates should follow the format MM-DD-YYYY for consistency in expense and income records.
- **Storage Capabilities:**
  - The database should support high storage capacity for storing large volumes of user data as the user base grows.
- **Data Retention:**
  - User financial records should be retained indefinitely or until a user requests deletion.
- **Data Integrity:**
  - Unique constraints should ensure that duplicate entries (e.g., duplicate expense records) are not created.
  - Referential integrity should be enforced, linking income and expense records to the respective user profile.

7.0 Approval

The signatures below indicate their approval of the contents of this document.

Project Role	Name	Signature	Date
Marketing Team	Adam Simcoe	Adam Simcoe	November 8, 2024
Project Manager	Trang Nguyen	Trang Nguyen	November 8, 2024
Development Team	Nhan Tran	Nhan Tran	November 8, 2024
Requirement Engineer	Nhu Ly	Nhu Ly	November 8, 2024
Financial Team	Christian Do	Christian Do	November 8, 2024