Budget Planner

COMP246

Section: 408

Professor: Parimala Alva

Group# 2

Group Members:

Gichul Jaun,

Joshua Padilla Vargas,

Lucas Fraser,

Maryam Khan,

Mathurangy Mathiyalagan,

Sara Hamoleila,

Rimsha Batool

Part A: Project Scope and Requirements

Section 1: Problem Statement

1.1.a. Problem Domain & Need

1.1.b. List of Capabilities and Benefits

1.2 Identify the stakeholders and their roles
1.3 Identify the sub-systems of your application (What are its functional components)
1.4 Who are the intended users of the SRS documentation
Section 2. General Overview Modelling
2.1 Context Flow Diagram (CFD)
Section 3. Requirements Modeling
3.1 Non-functional requirements
3.2 Functional requirements
3.2.1 Registration Subsystem
3.2.2 Expense Subsystem
3.2.3 Income Subsystem
3.2.4 Goals Subsystem
3.2.5 Reports Subsystem
Section 4. Domain Class Diagram
4.1 List of classes
4.2 Domain Class Diagram
Section 5. Entity Relationship Diagram
Section 6. Systems Sequence Diagrams
6.1 User Management System
6.2 Goal Management System
Section 7. Object State Machine Diagrams
7.1 Diagram 1
7.2 subheading
Section 8. Technologies
Section 9. Project Management (Gantt Chart)
Part B: Software Design Architecture
Section 1. Requirements Edits to Part A
1.1 subheading
1.2 subheading
1.3 subheading
Section 2. Overview Model
2.1 Intended users of the SSD document
2.2 Context Flow Diagram (CFD)
2.3 Architectural Context Diagram (ACD)
Section 3. Modularization
3.1 subheading
3.1.1 subheading
3.1.2 subheading
3.1.3 subheading
3.2 subheading
3.2.1 subheading

3.2.2 subheading
3.2.3 subheading
3.3 subheading
3.3.1 subheading
3.3.2 subheading
3.3.3 subheading
Section 4. Framework MVC (Model View Controller)
4.1 MVC pattern diagram to include
4.1.1 subheading
4.1.2 subheading
4.1.3 subheading
4.2 Full Sequence Diagrams
4.2.1 subheading
4.2.2 subheading
4.2.3 subheading
4.3 State Machine Diagrams
4.3.1 subheading
4.3.2 subheading
4.3.3 subheading
Section 5. Data Layer
5.1 Database Schema
5.2 Technology List Update
Section 6. Gantt Chart update (Reuben)
Part C: System Design Documents
Section 1. Corrections do Design Specifications Part B.
1.1 Corrections on MVC Pattern diagram
1.2 Corrections on Technology List Update
Section 2. Software Design Patterns
2.1 State Pattern
2.2 Implementation Pattern
2.3 Adapter Pattern
Section 3. UI/UX design
3.1 subheading
3.2 subheading
3.3 subheading
Section 4. High Level Component / Deployment Diagram
Section 5. Update the Gantt chart to include Part C Tasks

Part A: Project Scope and Requirements

Section 1: Problem Statement

There are so many individuals who find it difficult to manage their finances effectively, without a proper approach to a financial plan. Families find it challenging to handle their income and expenses in a proper way without spending a lot of time budgeting, which leads to overspending and lack of future savings. The need for a user-friendly tool is inevitable to plan and control finances properly.

1.1.a. Problem Domain & Need

The problem domain for the “Budget Planner ” is to provide personal financial management. Budget Planner application includes financial planning for creating goals and the monthly strategies to reach that goal, budgeting for proper plan to spend enough money effectively, expense tracker to monitor, and record expenses to confirm that they follow with the budget, income tracker to calculate the income from different sources and make a plan to use them effectively, get month and annual summary to figure out our achievement towards the goal.

The need of the “Budget Planner” application is more significant to support individuals to manage finances in a proper way with less time. The major need is to reduce finance stress, by providing notifications on overspending and maintaining healthy finance behavior. Develop savings behavior to notify the savings area and improve their spending habits.Enhance our budget plan which helps to track our financial goals.

1.1.b. List of Capabilities and Benefits

Capabilities:

- User Management: Primary users can add family members or additional users to the account.
- Expense tracking: Log and categorize daily expenses.
- Income tracking: Record sources of income.

- Goal setting: Track progress towards savings or budget goals.
- Financial summaries: Monthly reports and graphical insights.
- Notifications:Alerts for bill due dates and overspending.
- Privacy: Data security for sensitive information.
- User-friendly interface for accessibility.

Benefits:

- **Improved Financial Management**: Users can easily track and manage their finances, helping them stay within budget and achieve savings goals.
- **Enhanced Decision-Making**: Detailed reports and graphical insights provide valuable information for making informed financial decisions.
- **Better Collaboration**: Multiple users, such as family members, can manage and review finances together, promoting transparency and joint planning.
- **Increased Awareness**: Notifications and alerts help users stay aware of their financial status and avoid overspending or missed payments.
- **Secure Data Handling**: Data protection measures ensure that users' sensitive financial information remains confidential and secure.
- **Ease of Use**: A user-friendly interface makes the app accessible to a wide range of users, including those who may not be tech-savvy.

1.2 Identify the stakeholders and their roles

- Individual users (end-users)
  - Primary users who will interact with the budget planner app
  - Provide feedback on app features and user experience
  - Test the app for usability and efficiency
- UI/UX Designers
  - Design user-friendly interface
  - Ensure the app is easy to use, and visually appealing
  - Conduct user experience testing
- Customer Support
  - Provide assistance to users facing issues
  - Respond to user inquiries and troubleshoot issues
  - Provide feedback to the developers on common user problems
- Developers

    - **Implement UI Designs:** Translate designs into code, ensure responsive design.
    - **Integrate Visual and Interactive Elements:** Implement features, ensure smooth transitions.
    - **Develop server-side Logic:** Create server-side components, manage database interactions.
    - **Build and Maintain Database Schemas:** Design schemas, optimize queries.
- Quality Assurance (QA) Testers:
    - **Conduct Functional and Non-Functional Test:** Test features, performance, security.
    - **Report Issues:** Document and report bugs, collaborate with developers.
- Product Managers:
    - **Oversee Project Development:** Define goals, scope, and timelines.
    - **Gather and prioritize Requirements:** Collect feedback, prioritize features.

1.3 Identify the sub-systems of your application (What are its functional components)

1. **User Management System**
    - **Sign Up/Registration:** Allows users to create new accounts.
    - **Login/Authentication:** Provides secure access to the application, including password recovery and multi-factor authentication.
    - **User Roles Management:** Enables primary users to add, remove, and manage additional users with varying access levels.
2. **Expense Tracking System**
    - **Expense Logging:** Lets users log daily expenses with details such as amount, category, and date.
    - **Expense Categorization:** Allows users to categorize expenses into predefined or custom categories.
    - **Expense Editing/Deletion:** Provides functionality to edit or delete previously recorded expenses.
3. **Income Tracking System**

    - **Income Recording:** Facilitates recording of income from various sources.
    - **Income Categorization:** Enables categorization of income sources for better tracking.
    - **Income Editing/Deletion:** Allows updating or removing of previously recorded income entries.
4. **Goal Management System**
    - **Goal Creation:** Lets users create financial goals.
    - **Goal Progress Tracking:** Tracks progress towards financial goals with visual indicators.
    - **Goal Updates:** Allows users to modify goals based on changing financial situations.
5. **Reporting System**
    - **Monthly Reports:** Generates monthly financial summaries, including income, expenses, and budget utilization.
    - **Annual Reports:** Provides comprehensive annual financial reports.
    - **Graphical Insights:** Offers charts and graphs to visualize financial data and trends.
6. **Notification System**
    - **Alerts:** Sends notifications for important events such as overspending and bill due dates.
    - **Reminders:** Allows users to set and receive reminders for recurring expenses or savings targets.
7. **Data Security System**
    - **Data Encryption:** Ensures sensitive financial data is encrypted to maintain privacy.
    - **Access Control:** Implements role-based access controls to manage user permissions and data access.
8. **Customization System**
    - **Personalization:** Allows users to customize settings, such as budget categories and notification preferences.

1.4 Who are the intended users of the SRS documentation

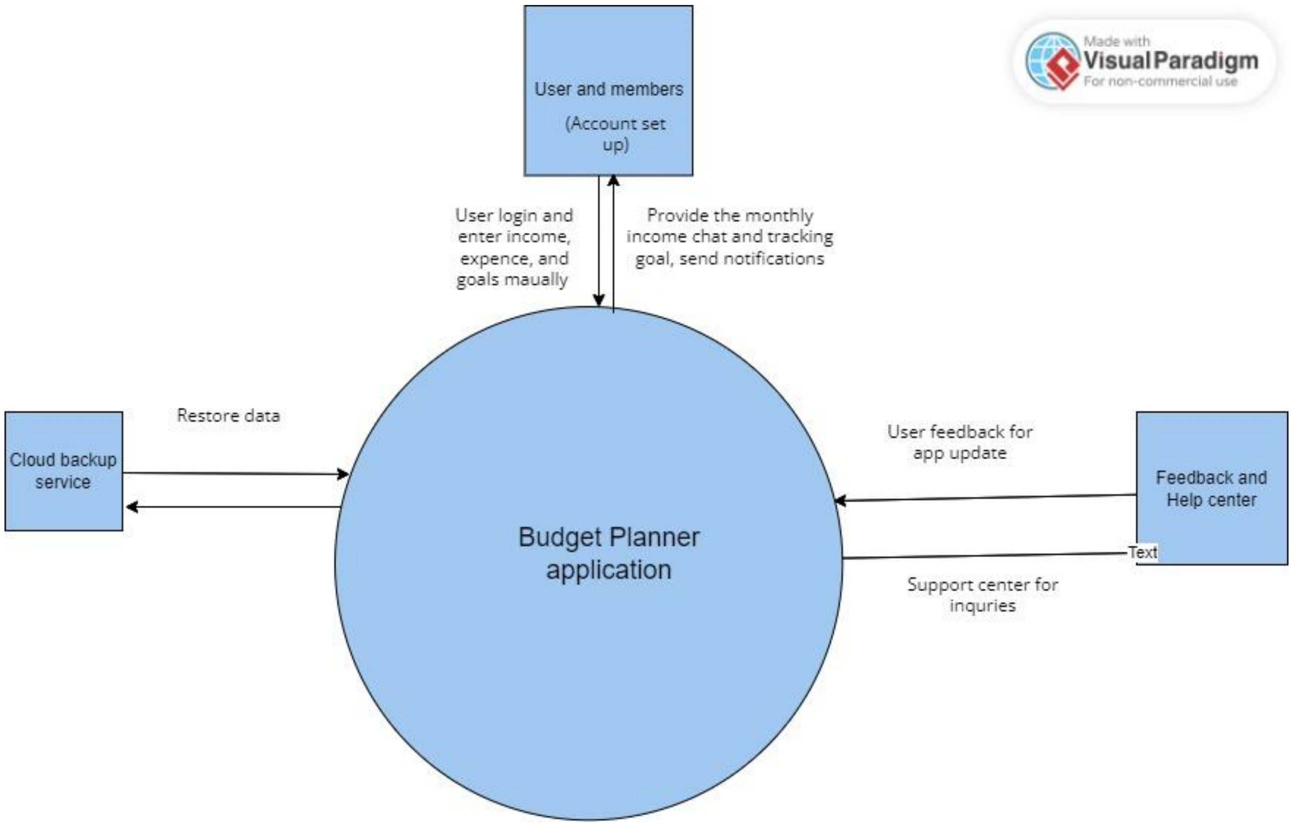
- **Developers**
    - **Purpose:** To understand the functional and non-functional requirements for coding and implementing the application features.

They use the SRS to guide their development process, ensure alignment with requirements, and address any technical constraints.

- **Designers (UI/UX)**
    - **Purpose:** To design user interfaces and user experiences that align with the requirements outlined in the SRS. They use the SRS to ensure that their designs meet the user needs and functional requirements specified.
- **Project Managers**
    - **Purpose:** To oversee project progress and ensure that all aspects of the development are on track. They use the SRS to manage project scope, resource allocation, timelines, and deliverables.
- **Quality Assurance (QA) Testers**
    - **Purpose:** To create test plans and test cases based on the requirements documented in the SRS. They use the SRS to ensure that all functionalities are tested and meet the specified requirements.
- **Stakeholders (e.g., Product Owners, Clients)**
    - **Purpose:** To review and validate that the requirements and specifications align with their expectations and business needs. They use the SRS to ensure that the final product meets the agreed-upon goals and requirements.
- **Technical Writers**
    - **Purpose:** To create user manuals, help guides, and other documentation based on the requirements outlined in the SRS. They use the SRS to ensure accuracy and completeness in their documentation.
- **Support Staff**
    - **Purpose:** To understand the features and functionalities of the application to provide effective support and assistance to end-users. They use the SRS to familiarize themselves with the application’s capabilities and limitations.

Section 2. General Overview Modelling

2.1 Context Flow Diagram (CFD)



Section 3. Requirements Modeling

3.1 Non-functional requirements

NFR#	Name	Description
NFR001	Performance	The application should handle up to 10,000 transactions efficiently without significant delay.
NFR002	Scalability	The system should be able to scale to accommodate an increasing number of the users and data volume.
NFR003	Security	Sensitive financial data must be encrypted using industry-standard encryption methods.
NFR004	Availability	The application should have 99.9% uptime, with scheduled maintenance windows communicated in

		the advance.
NFR005	Usability	The user interface should be intuitive and easy to navigate, with accessibility features for users with disabilities.
NFR006	Compatibility	The app should be compatible with major web browsers (Chrome, Firefox, Safari) and mobile devices (iOS and Android).
NFR007	Data Backup	Regular backups of user data should be performed to prevent data loss.
NFR008	Response Time	The application should respond to user inputs within 2 seconds for a smooth user experience.
NFR008	Privacy	User data should only be accessible to authorized users, with strict controls on data access and sharing.

3.2 Functional requirements

FR#	Name	Stakeholder	Description
FR01	Sign Up/Registration	Individual Users/ End user	Users can create a new account by providing personal details, email, and password.
FR02	Login/Authentication	Individual Users, Customer Support	Users can securely log in using their email and password; includes password recovery and multi-factor authentication.
FR03	User Roles Management	Individual Users, Developer	Primary users can add, remove, and manage additional users with different roles (e.g., family members with varying access levels).
FR04	Expense Logging	Individual Users	Users can log daily expenses, including amount, category, and date.
FR05	Expense Categorization	Individual Users	Users can categorize expenses into predefined or custom categories.

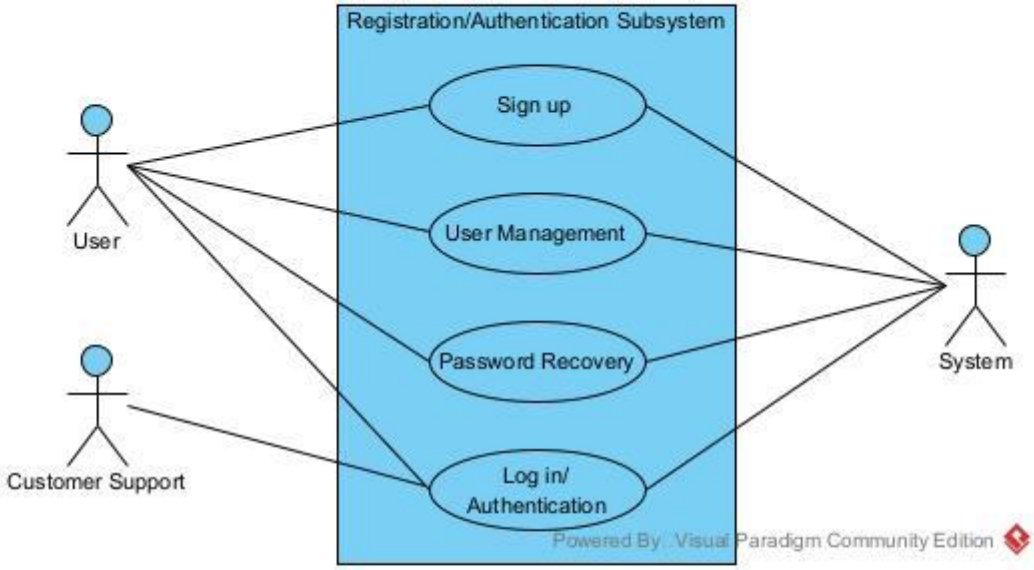
FR06	Expense Editing/Deletion	Individual Users	Users can edit or delete previously recorded expenses.
FR07	Income Logging	Individual Users	Users can record income from various sources, such as salary, freelance work, and investments.
FR08	Income Categorization	Individual Users	Users can categorize income sources for better tracking and reporting.
FR09	Income Editing/Deletion	Individual Users	Users can update or remove previously recorded income entries.
FR10	Goal Creation	Individual Users	Users can create financial goals, such as saving for a specific purpose or reducing debt.
FR11	Goal Progress Tracking	Individual Users	Users can track their progress towards financial goals with visual indicators and updates.
FR12	Goal Updates	Individual Users	Users can modify goals based on changing financial situations.
FR13	Monthly Reports	Individual Users	Users can generate and view monthly financial summaries, including income, expenses, and budget utilization.
FR14	Annual Reports	Individual Users	Users can generate and view comprehensive annual financial reports.
FR15	Graphical Insights	Individual Users	Users can view charts and graphs that provide visual representations of their financial data and trends.
FR16	Alerts	Individual Users / Customer support	Users receive notifications for important events such as overspending, upcoming bill due dates, and goal milestones.

FR17	Reminders	Individual Users	Users can set and receive reminders for recurring expenses or savings targets.
FR18	Access Control	Individual Users, Developers	Role-based access controls are implemented to manage user permissions and data access.
FR19	Dashboard Overview	Individual Users, UI/UX Designers	Users are presented with an overview of their financial status on the main dashboard, including income, expenses, and budget utilization.
FR20	Customization	Individual Users	Users can personalize settings, such as budget categories and notification preferences.

3.2.1 Registration Subsystem

FR#	Name	Stakeholder	Description
FR01	Sign Up/Registration	Individual Users/ End user	Users can create a new account by providing personal details, email, and password.
FR02	Login/Authentication	Individual Users, Customer Support	Users can securely log in using their email and password; includes password recovery and multi-factor authentication.
FR03	User Roles Management	Individual Users	Primary users can add, remove, and manage additional users with different roles (e.g., family members with varying access levels).
FR04	Password Recovery	Individual Users	Users can update their password if it is forgotten or unsecure.

Use Case Diagram



User Story

1. As a user I want to create an account so that I can use the program.

Acceptance Criteria:

- When the application is opened, the login/sign up should be displayed
- If the sign up information is verified, the account is to be created
- After sign up is verified, the user should be able to log in

2. As a developer I want account authentication so that users can log in securely.

Acceptance Criteria:

- User should be given the option to use 2-factor authentication for their account
- Users should be required to create a password consisting of letters, numbers, and special characters

3. As a user I want to log in so that I can save information within the application.

Acceptance Criteria:

- User accounts should be able to add secondary users to their account with unique login information

4. As a user I want to add a user to my account so that my spouse and I can work together.

Acceptance Criteria:

- User accounts should be able to add secondary users to their account with unique login information
- Primary user accounts should be able to add or remove secondary users associated with the account

5. As a user I want to change my password so that I can keep my account secure.

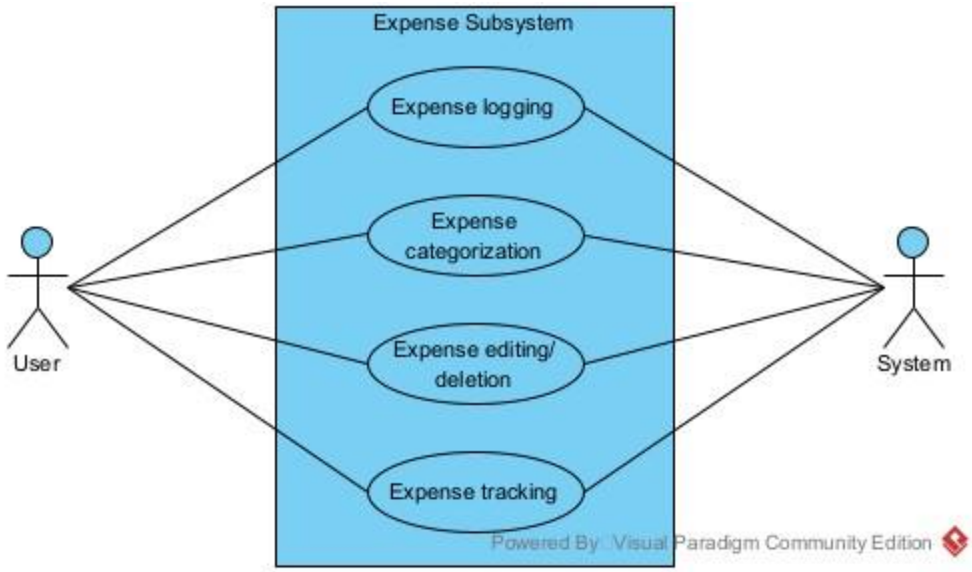
Acceptance Criteria:

- When login is selected, forgot password should be displayed as an option

3.2.2 Expense Subsystem

FR#	Name	Stakeholder	
FR01	Expense Logging	Individual Users	Users can log daily expenses, including amount, category, and date.
FR02	Expense Categorization	Individual Users	Users can categorize expenses into predefined or custom categories.
FR03	Expense Editing/Deletion	Individual Users	Users can edit or delete previously recorded expenses.
FR04	Expense Tracking	Individual Users	Users can track the expense daily or monthly to make changes on their finance plan.

Use Case Diagram



User Story

1. As a user I want to log daily expenses so that I can track my finances.

Acceptance Criteria:

- When I enter an expense amount, I should be able to assign it to a date and category

2. As a user I want to create custom categories so that I can track predefined expenses.

Acceptance Criteria:

- If I want categorize my expenses, I should be able to create custom categories

3. As a user I want to edit recorded expenses so that I can correct a mistake.

Acceptance Criteria:

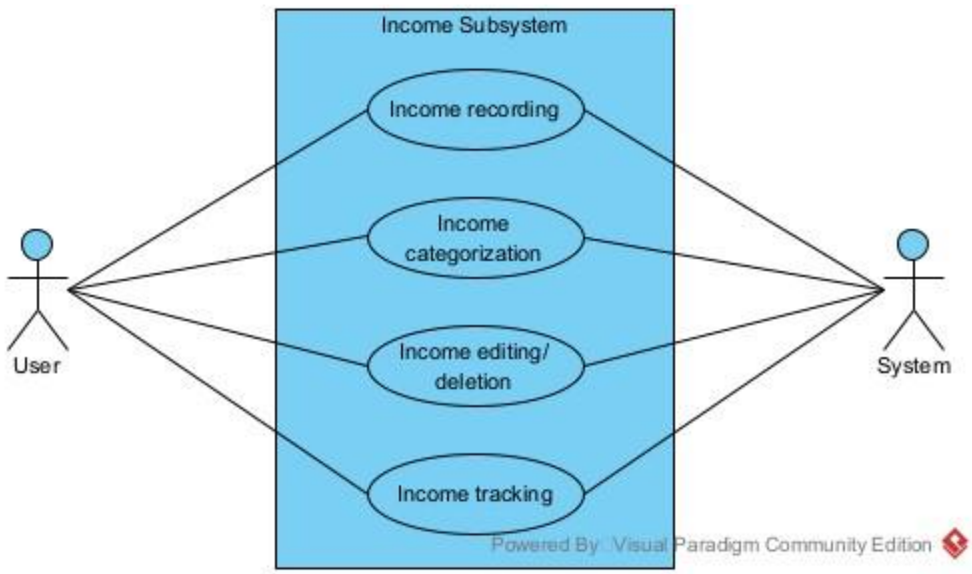
- I should be able to access the information of previously added expenses and modify the information as necessary

3.2.3 Income Subsystem

FR#	Name	Stakeholder	Description

FR01	Income Recording	Individual Users	Users can record income from various sources, such as salary, freelance work, and investments.
FR02	Income Categorization	Individual Users	Users can categorize income sources for better tracking and reporting.
FR03	Income Editing/Deletion	Individual Users	Users can update or remove previously recorded income entries.
FR04	Income Tracker	Individual Users	Users can track the income from various sources and use money based on their needs and savings.

Use Case Diagram



User Story

1. As a user I want to specify income sources so that I can easily divide my finances.

Acceptance Criteria:

- I should be able to segregate income amounts if it is obtained from different sources

2. As a user I can plan monthly expenses and savings based on available income.

Acceptance Criteria:

- I should be able to divide the information added for income and use it to calculate what can be saved and what can be used for monthly expenses

3. As a user I want to update or remove income sources so that I can keep up to date tracking.

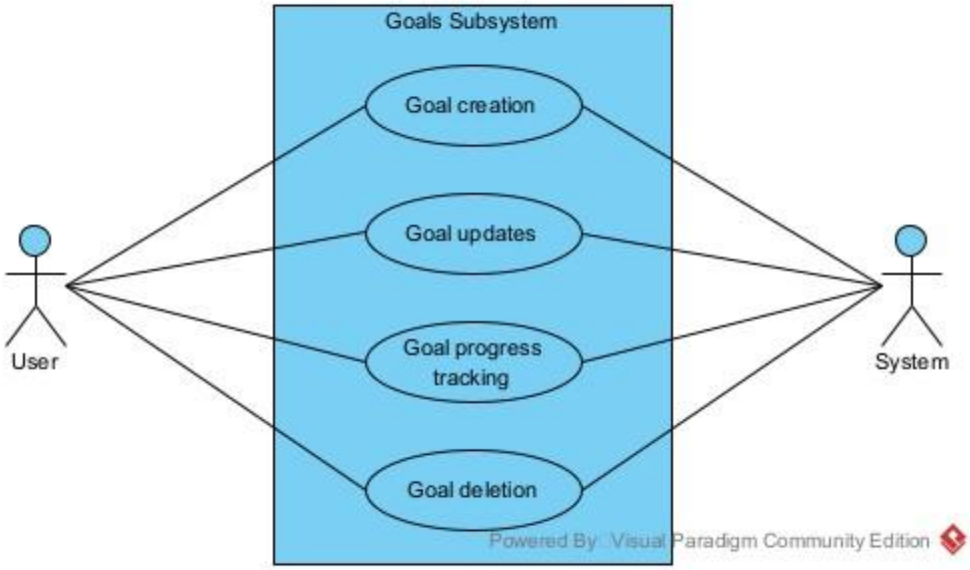
Acceptance Criteria:

- I should be able to access the information of previously added income and modify the information as necessary

3.2.4 Goals Subsystem

FR#	Name	Stakeholder	Description
FR01	Goal Creation	Individual Users	Users can create financial goals, such as saving for a specific purpose or reducing debt.
FR02	Goal Progress Tracking	Individual Users	Users can track their progress towards financial goals with visual indicators and updates.
FR03	Goal Updates	Individual Users	Users can modify goals based on changing financial situations.
FR04	Goal Deletion	Individual Users	Users can delete the goals once achieved and create new goals.

Use Case Diagram



User Story

1. As a user I want to create financial goals so that I can focus on debt reduction.

Acceptance Criteria:

- I should be able to create a financial goal with any arbitrary amount

2. As a program designer I want a financial overview on the main dashboard so that users have a visual representation of their finances.

Acceptance Criteria:

- The main dashboard should display a visually appealing overview of the users financial information

3. As a user I want to update or delete goals so that I can keep my financial needs up to date.

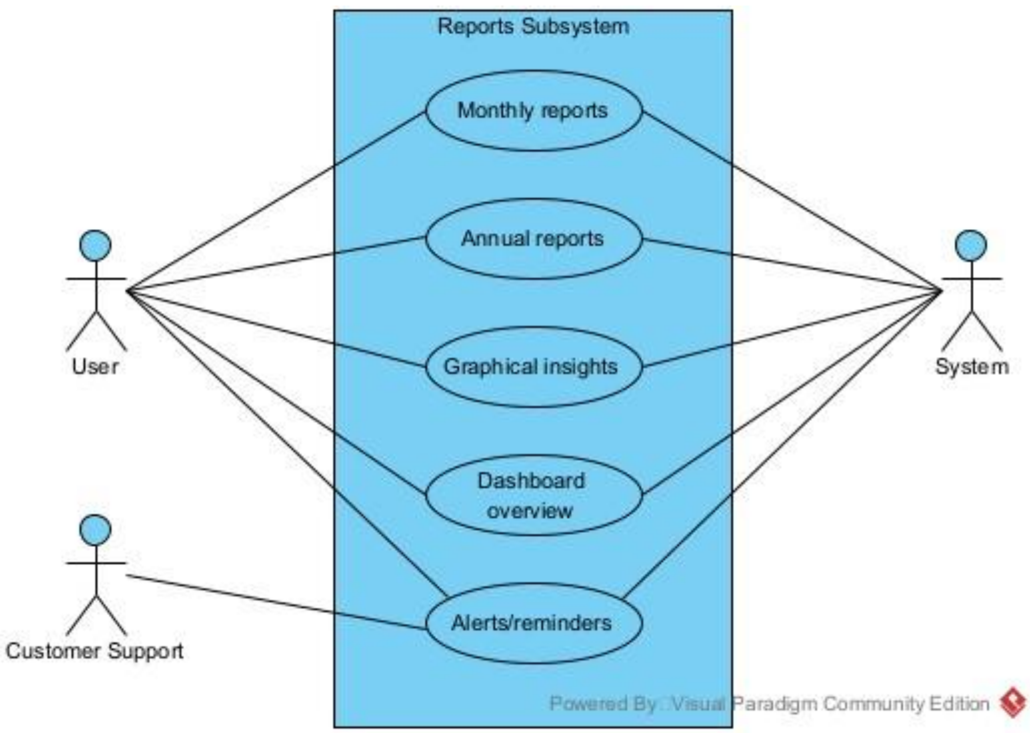
Acceptance Criteria:

- I should be able to access the information of previously added goals and modify the information as necessary

3.2.5 Reports Subsystem

FR#	Name	Stakeholder	Description
FR01	Monthly Reports	Individual Users	Users can generate and view monthly financial summaries, including income, expenses, and budget utilization.
FR02	Annual Reports	Individual Users	Users can generate and view comprehensive annual financial reports.
FR03	Graphical Insights	Individual Users	Users can view charts and graphs that provide visual representations of their financial data and trends.
FR04	Alerts	Individual Users / Customer support	Users receive notifications for important events such as overspending, upcoming bill due dates, and goal milestones.
FR05	Reminders	Individual Users	Users can set and receive reminders for recurring expenses or savings targets.
FR06	Dashboard Overview	Individual Users	Users are presented with an overview of their financial status on the main dashboard, including income, expenses, and budget utilization.

Use Case Diagram



User Story

1. As a user I want to view monthly and annual reports so that I can keep comprehensive records

Acceptance Criteria:

- I should be able to access reports containing all of the necessary income, expense, savings, and goals information

2. As a user I want to visual indicators so that I can see my financial progress

Acceptance Criteria:

- If the user is in a net negative after entering income or expense information, then the user should receive an alert in the form of a visual indicator notifying the user of their situation

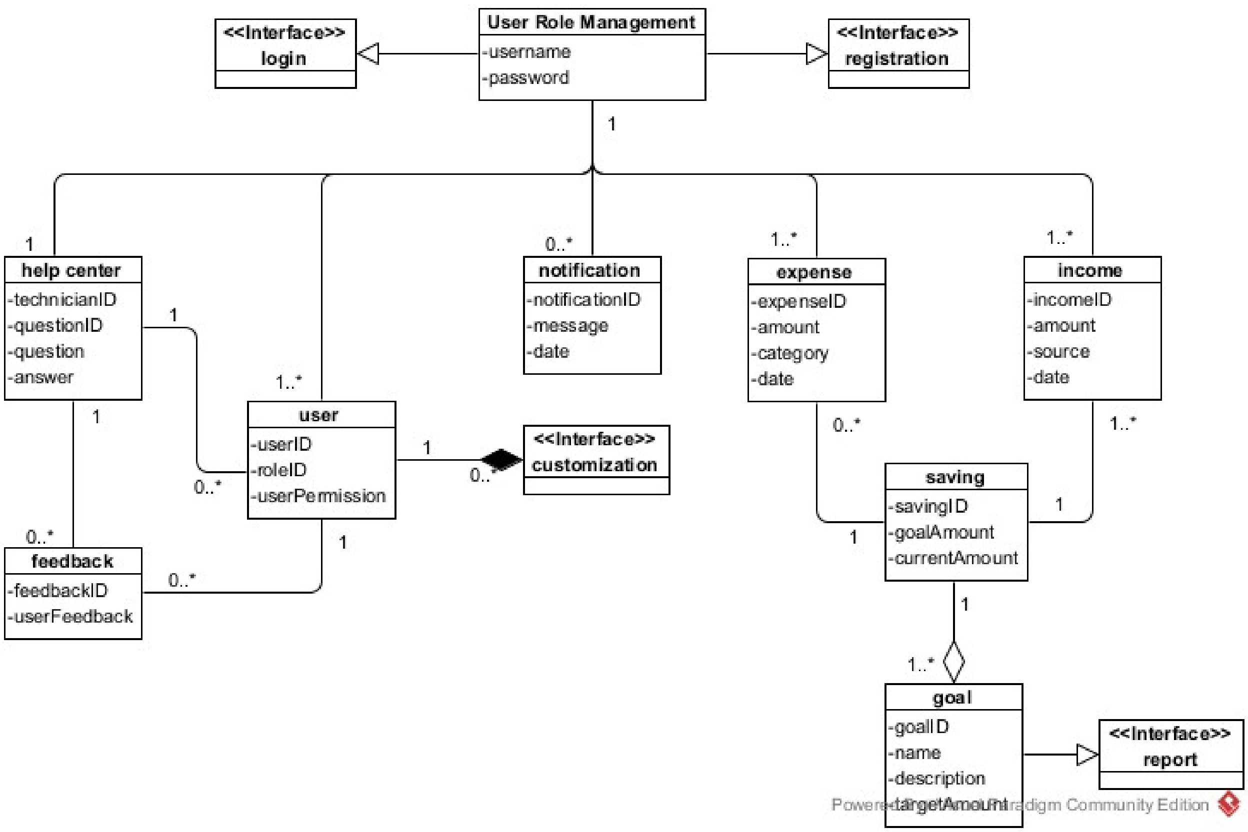
Section 4. Domain Class Diagram

4.1 List of classes

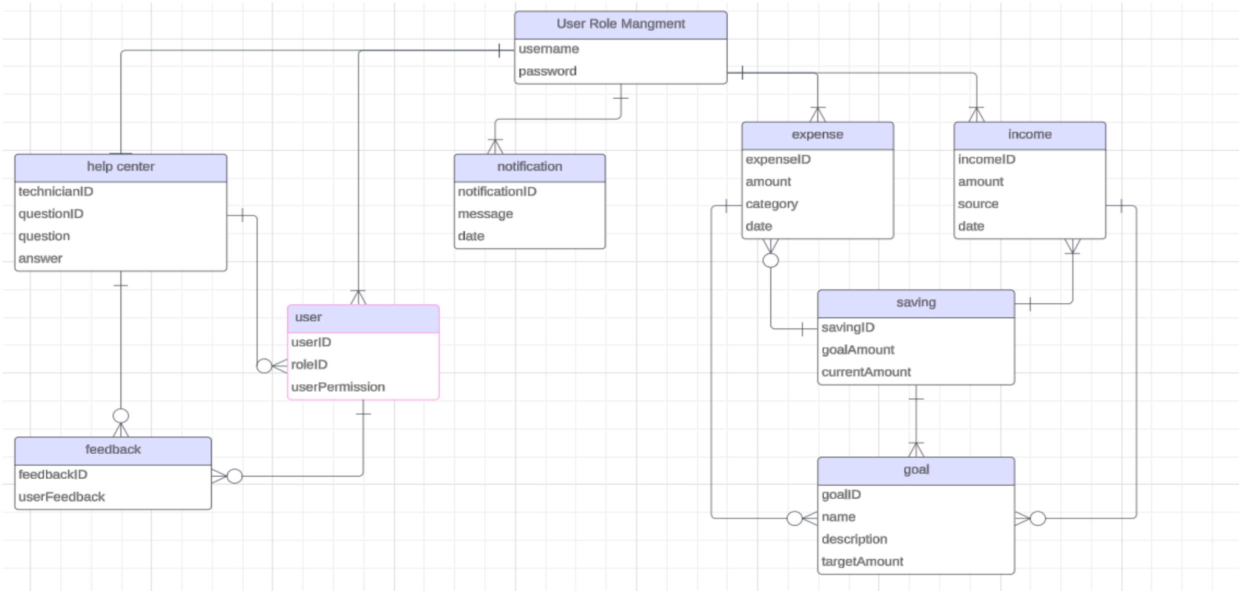
- **User** – Represents an individual or family using the budgeting tool, now including roles for user management.
- **Expense** – Tracks various spending categories like rent, groceries, and utilities.
- **Income** – Represents different income streams such as salary, freelance work, or investments.
- **Saving** – Refers to savings goals or emergency funds set by the user, now incorporating account details.
- **Goal** – Replaces the previous "Budget" and captures financial objectives, monitoring both income and expenses.
- **Report** – Generates financial summaries, such as income vs. expenses or spending trends.
- **Notification** – Sends alerts or reminders about upcoming bills, savings goals, or important events.

- **Customization** – Allows users to adjust settings such as notification preferences, app appearance, and other personalized options.
- **Feedback** - Enables user to send feedbacks about their overall app usage
- **User Role Management** - User can sign in, authenticate and set up the role.
- **Help Center** - Support and troubleshoot user requests and response to user feedbacks

4.2 Domain Class Diagram

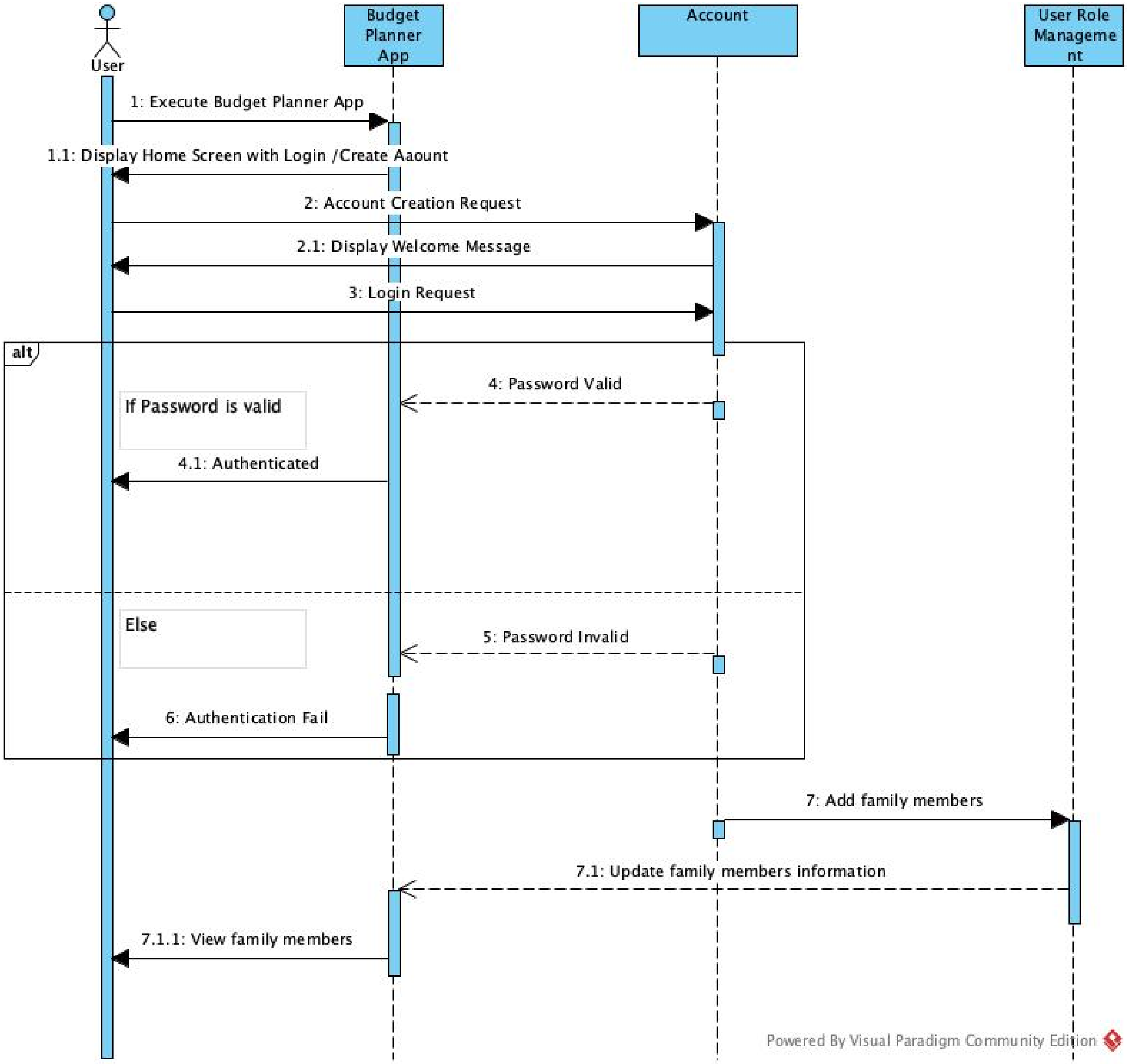


Section 5. Entity Relationship Diagram

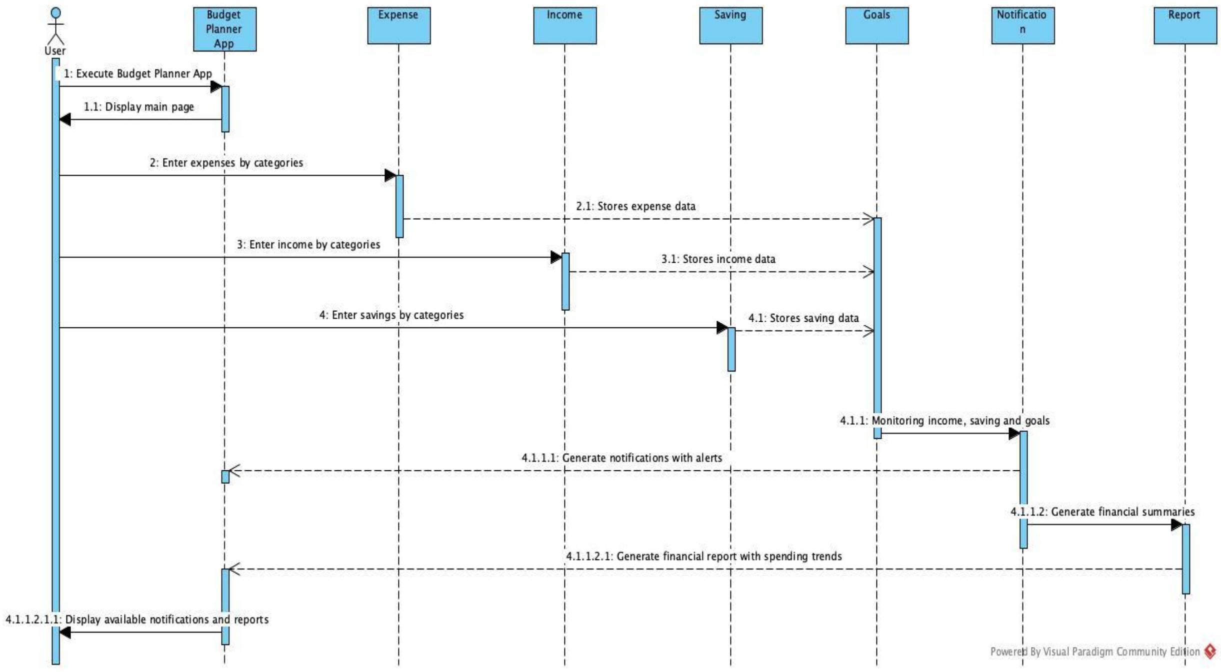


Section 6. Systems Sequence Diagrams

6.1 User Management System



6.2 Goal Management System



Section 7. Object State Machine Diagrams

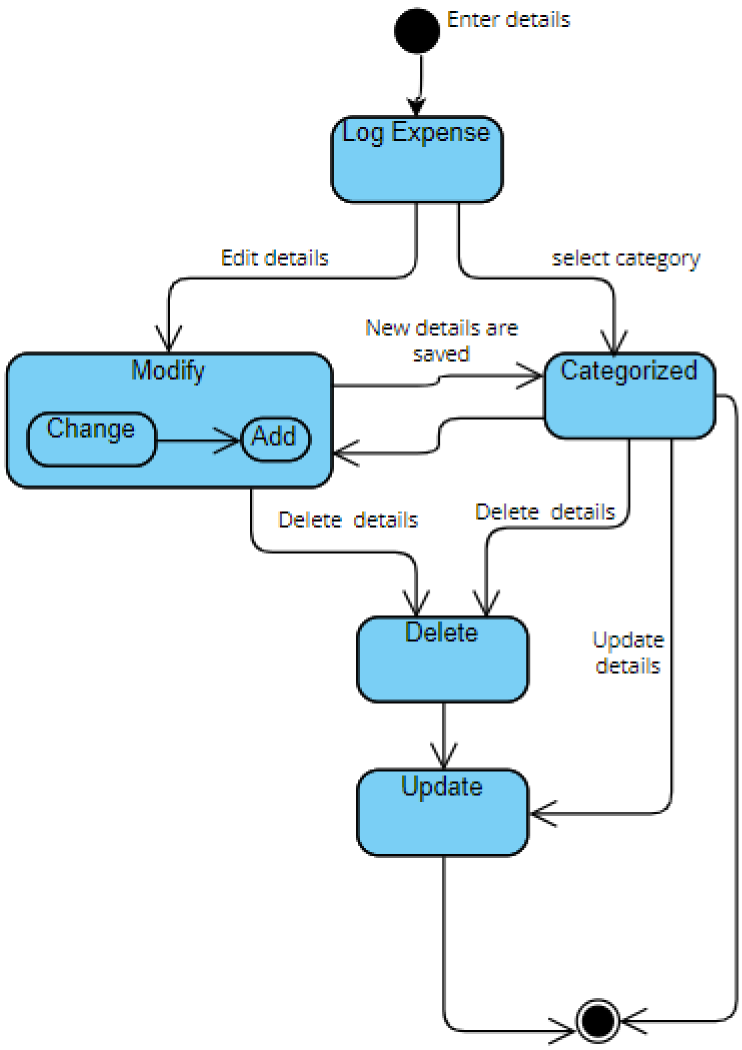
7.1 Diagram 1

Object: Expense tracking

Triggering event: User modifies expense details

State: ON, login, log expense, categorize, modify details, change, delete, update

Transitions: log in, log expense, categorize expense, edit expense, save, delete, update



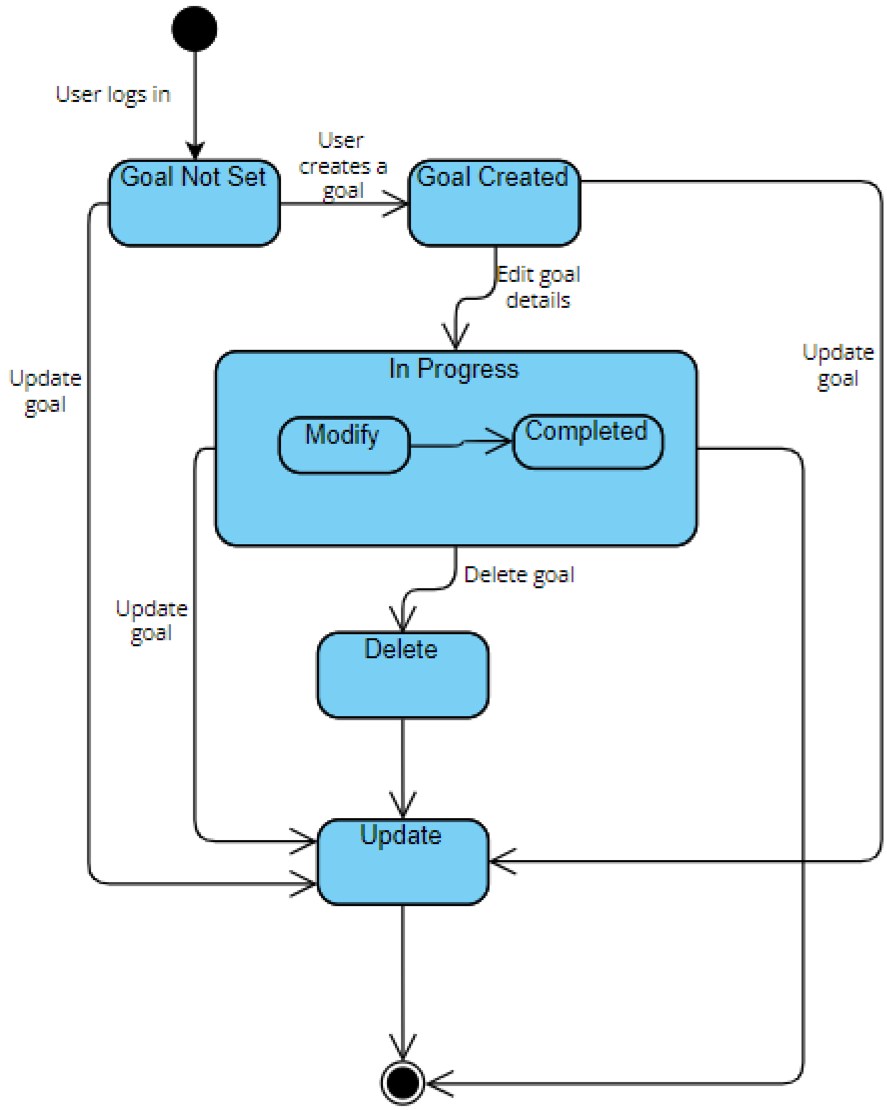
7.2 Diagram 2

Object: Goal

Triggering event: User creates, updates, deletes, or tracks progress

State: ON, login, Goal not set, Goal created, In progress, Goal updated, Goal completed, Goal deleted.

Transitions: Log in, log out, goal not set, goal created, in progress, goal complete



Section 8. Technologies

Our system will be a mobile app which can be accessed across all smartphones/devices.

The tools used will be as such:

Client Side (Front-end–GUI):

- **React Native**: For building a cross-platform mobile app (both iOS and Android) with a single codebase.

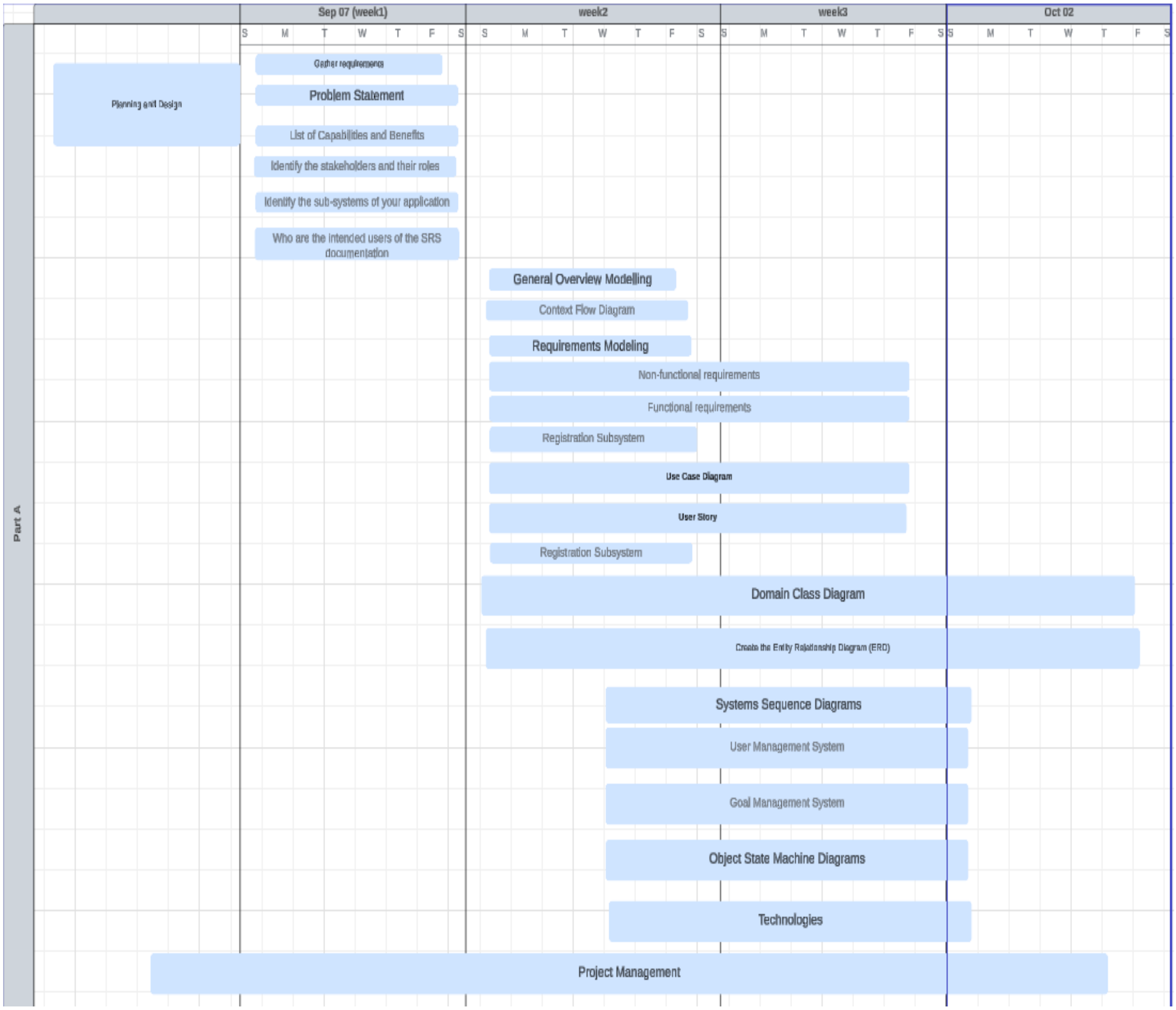
Business Logic (Middle layer–Class methods ets.):

- **Javascript/TypeScript:** JavaScript allows rapid development of interactive interfaces and business logic for the app. TypeScript adds static typing and other features which enable developers to write more robust, error-free code.

Back-End (Database):

- **Firebase Firestore**: A cloud NoSQL database where users can manually input and store their budget, expenses, income, and savings.
- **Firebase Authentication**: For simple user login, with email/password registration.

Section 9. Project Management (Gantt Chart)



Part B: Software Design Architecture

Section 1. Requirements Edits to Part A

1.1 Corrections on context flow diagram(section 2.1)

- Created an arrow connection from the budget planner to the user
- Mention the integration between the user and the budget planner application by means of text box

1.2 Corrections in the registration Subsystem (section 3.2.1)

- In 3.2.1 - Registration Subsystem- FR03, removed developer in the user role management requirement.
- In 3.2.1 - Registration Subsystem - FR02 - updated the use case diagram to include Customer Support as an actor in the Login/Authentication use case to reflect its involvement.
- In the Registration Subsystem, the 'Log In' user story had been described
- In all use-case diagrams, the name of the actor is added.

1.3 Corrections in State Machine Diagrams (Section 7.1, 7.2)

- Text indicating when the delete state and update state is triggered
- The action for each of the lines going out of "in progress" is specified by text

1.4 Corrections in Entity relationship Diagram (Section 5)

- Established a one-to-many relationship between income and goal, and between expense and goal.

Section 2. Overview Model

2.1 Intended users of the System Design Document(SSD) document

In the budget planner application the System Design Document includes following users such as Software Developers, Project Managers, System Architects, Quality Assurance team, Business Analysts, Stakeholders, and Maintenance and Support teams for the system development.

- **Software Developers:** Analyze the architecture of the system, design process and decisions and to implement the details in application.

- **Project Managers:** Ensure the alignment of the design with the requirements, timelines, and resource management. Design progress and changes were tracked by the project managers.

- **System Architects:**Analyze and validate the overall architecture and make sure that the design meets required security requirements, performance, and scalability.

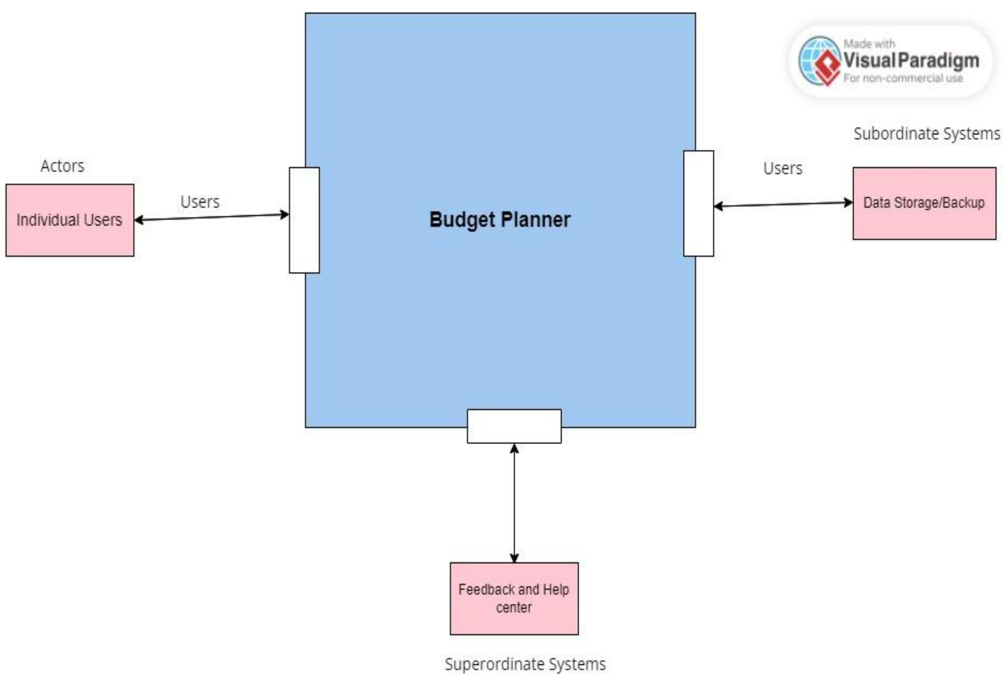
- **Quality Assurance team:** Develop the test cases and understand the design of the application. Track the system being built according to the design and meets functional and non-functional requirements.

- **Business Analysts:**Make sure the system design supports business requirements and goals, also ensures that the application accepts the needs of the end-users.

- **Stakeholders:**Approve the design and review the process to understand that it meets the expectations and intended functionality of the applicaiton.

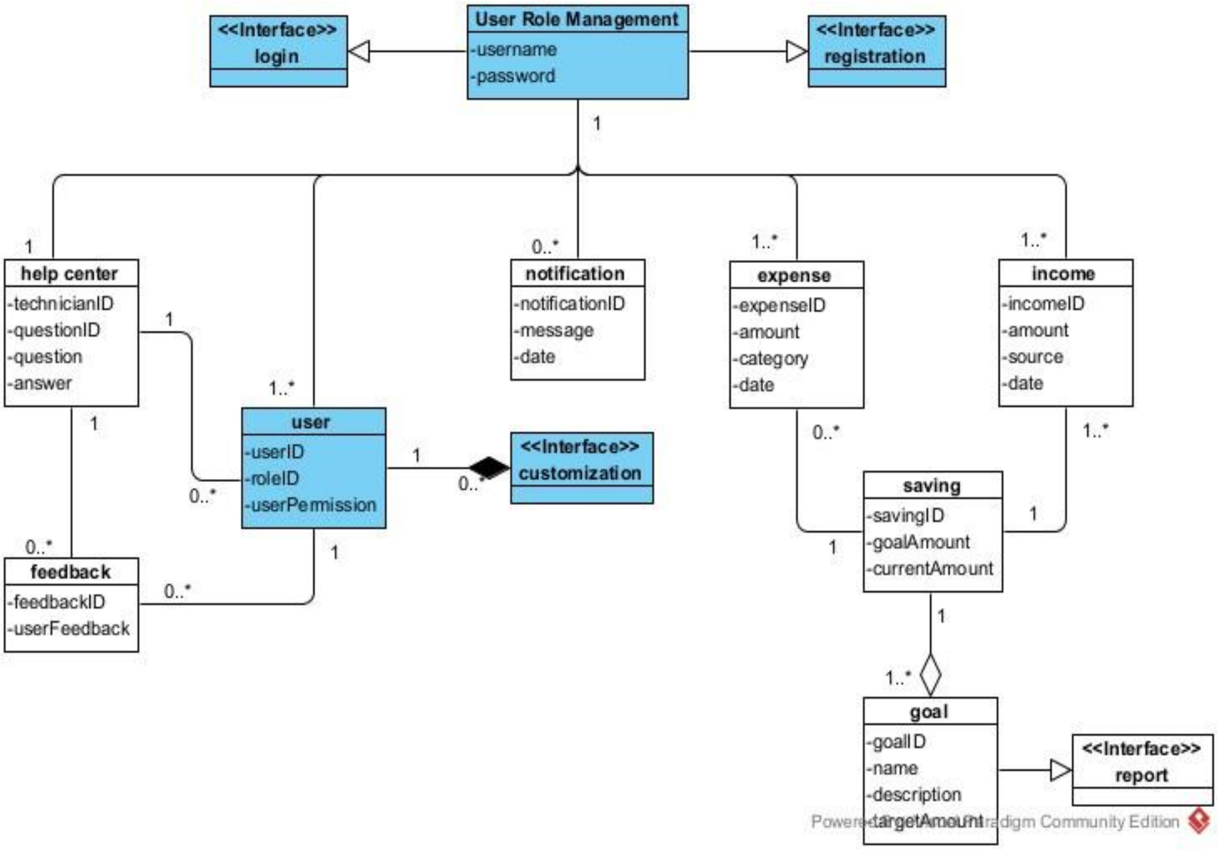
- **Maintenance and Support Teams:**Analyze and understand the architecture of the system and are able to provide support , troubleshoot issues, or make updates in the application based on the user needs.

2.2 Architectural Context Diagram (ACD)



Section 3. Modularization

3.1.1 Login & Registration Subsystem



3.1.2 Class Responsibility Collaboration cards (CRC cards)

Login Interface	
Super Classes: None	
Sub Classes: None	
Description: The Login Interface handles user authentication by verifying credentials with User Role Management and supports multi-factor authentication for added security.	
Attributes:	
Name	Description
Responsibilities:	
Name	Collaborator
Authenticate user credentials (username and password).	User Role Management
Ensure secure login process.	User Role Managment
Enable multi-factor authentication if required.	User Role Managment

User Role Managment	
Super Classes:	
Sub Classes:	
Description: It is responsible for managing user authentication, roles, and permissions within the system. It verifies login credentials, assigns roles during registration, and controls access based on user permissions.	
Attributes:	
Name	Description
username	Stores the username of the user
password	Stores the encrypted password of the user.
roleID	Identifier for the role associated with the user (e.g., admin, standard user).
userPermission	Defines the level of access for each role.
Responsibilities:	
Name	Collaborator
Verify credentials	Login Interface
Assign roles during registration	Registration Interface
Manage user permissions	User, Customization Interface
Enable/disable multi-factor authentication	Login Interface

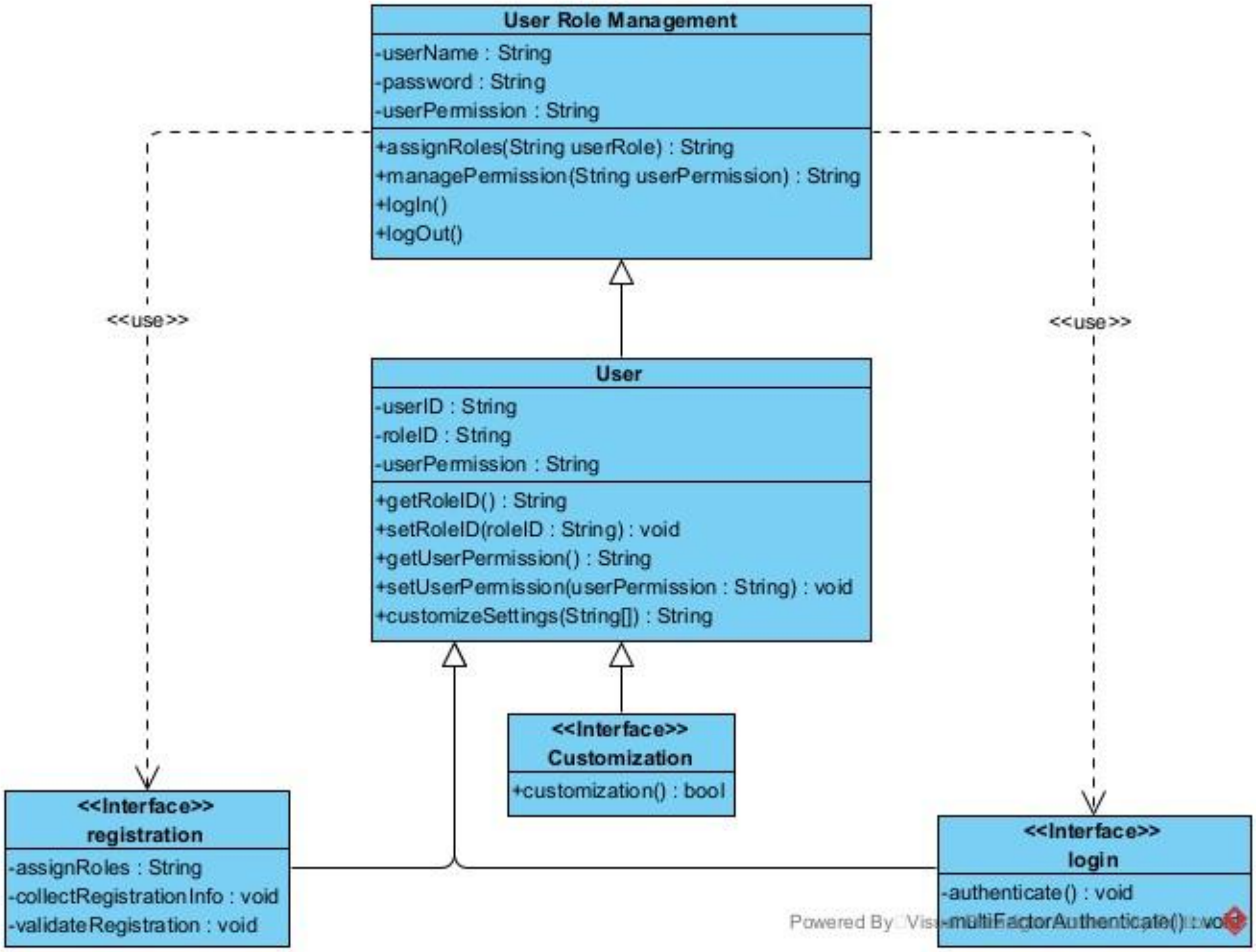
Double click: edit Super Classes:

Registration Interface	
Super Classes: None	
Sub Classes: None	
Description: The Registration Interface facilitates new user account creation by collecting registration details and working with User Role Management to assign roles and set up the account.	
Attributes:	
Name	Description
Responsibilities:	
Name	Collaborator
Collect registration information	User Role Management
Assign roles and permissions	User Role Management
Validate registration data	User Role Management

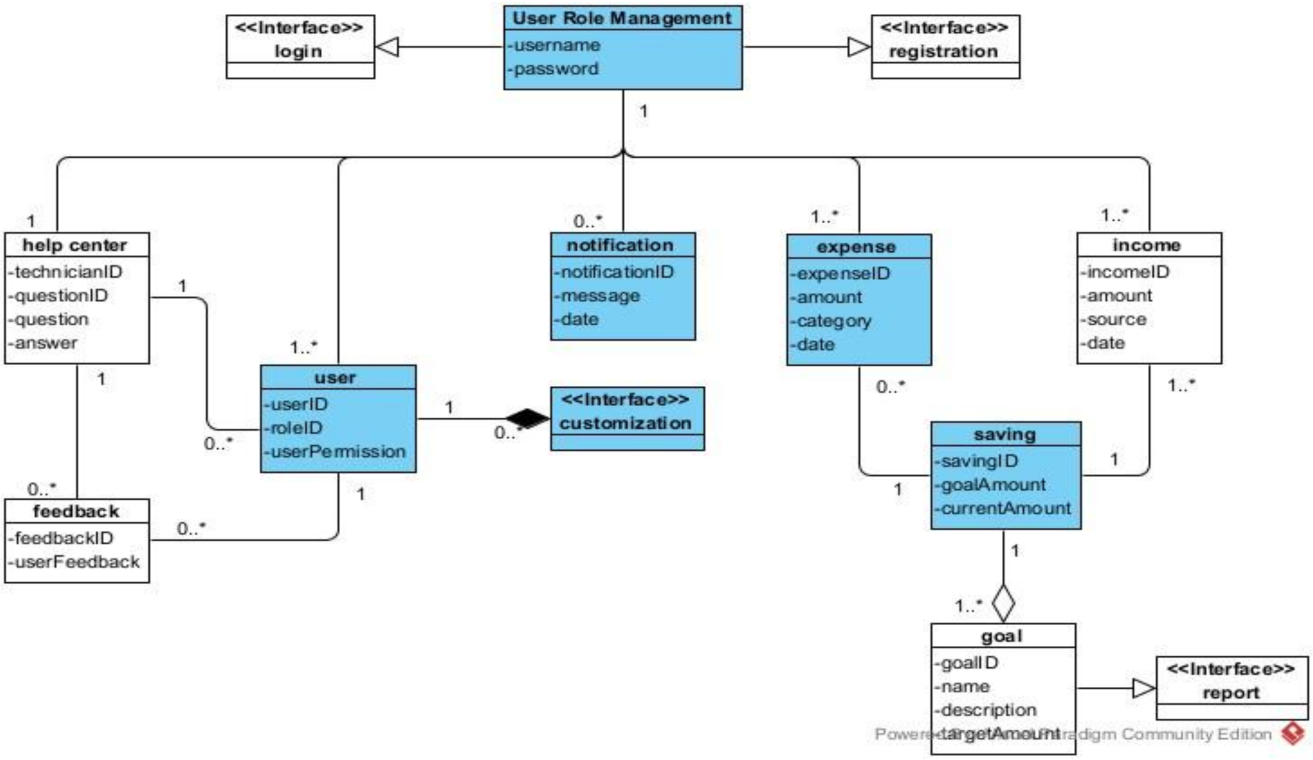
User	
Super Classes: None	
Sub Classes: None	
Description: The User class represents individuals using the system, storing essential information about each user, including roles and permissions. It interacts with various subsystems to manage user-specific data and actions.	
Attributes:	
Name	Description
userID	Unique identifier for each user.
roleID	Identifier for the user's role (e.g., admin, standard user).
userPermission	Defines the level of access or permissions associated with the user.
Responsibilities:	
Name	Collaborator
Store user information	User Role Management
Manage user permissions	User Role Management
Customize settings	Customization Interface

Customization	
Super Classes: None	
Sub Classes: None	
Description: The Customization interface allows for user-specific customization settings, enabling the system to save or retrieve a user's preferences.	
Attributes:	
Name	Description
Responsibilities:	
Name	Collaborator
Manage user customization settings	User
Return success or failure status of customization	User

3.1.3 Design Class Diagram(Login & Registration Subsystem)



3.2.1 Expense Subsystem



3.2.2 Class Responsibility Collaboration cards (CRC cards)

User Role Management

Super Classes:

Sub Classes:

Description: Responsible for the management of user authentication, roles, and permissions.

Attributes:

Name	Description
username	Stores the username.
password	Stores the encrypted password.
roleID	Identifier for user role (e.g., admin, standard user).
userPermission	Defines level of access for each user.

Responsibilities:

Name	Collaborator
Manage user permissions	User, Customization interface

User

Super Classes:

Sub Classes:

Description: Represents individuals using the system.

Attributes:

Name	Description
userID	Unique identifier for each user.
roleID	Identifier for user role (e.g., admin, standard user).
userPermission	Defines level of access for each user.

Responsibilities:

Name	Collaborator
Manage user permissions	User Role Management
Customize settings	Customization interface

Expense

Super Classes:

Sub Classes:

Description: Manages and records financial expenses and the details associated with each expense.

Attributes:

Name	Description
expenseID	Unique identifier for each expense.
amount	Monetary value of expense.
category	Category of expense.
date	recorded date of expense.

Responsibilities:

Name	Collaborator
Record expense	
Categorize expense	
Notify of overspending	Notification

Saving

Super Classes:

Sub Classes:

Description: Manages user savings, including goals and up to date progress towards goals.

Attributes:

Name	Description
savingID	Unique identifier for each savings entry.
goalAmount	Target amount set by user.
currentAmount	Current savings amount.

Responsibilities:

Name	Collaborator
Track savings progress	
Update savings balance	
Notify goal progress	Notification

Customization interface

Super Classes:

Sub Classes:

Description: User interface customization based on user preferences.

Attributes:

Name	Description

Responsibilities:

Name	Collaborator
Manage customization settings	User
Return success or failure status of customization	User

Notification

Super Classes:

Sub Classes:

Description: Handles alerts and reminders. Manages messages.

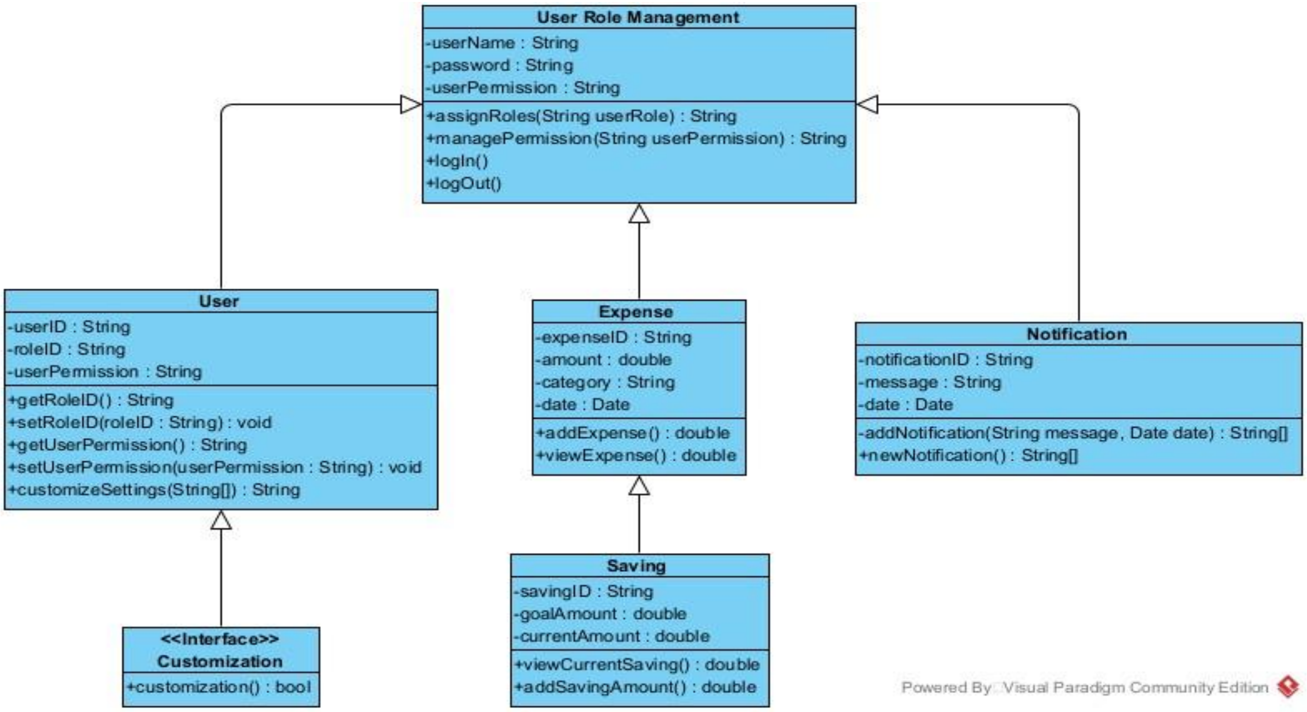
Attributes:

Name	Description
noticationID	Unique identifier for each notification.
message	Content of notification.
date	Date of notification.

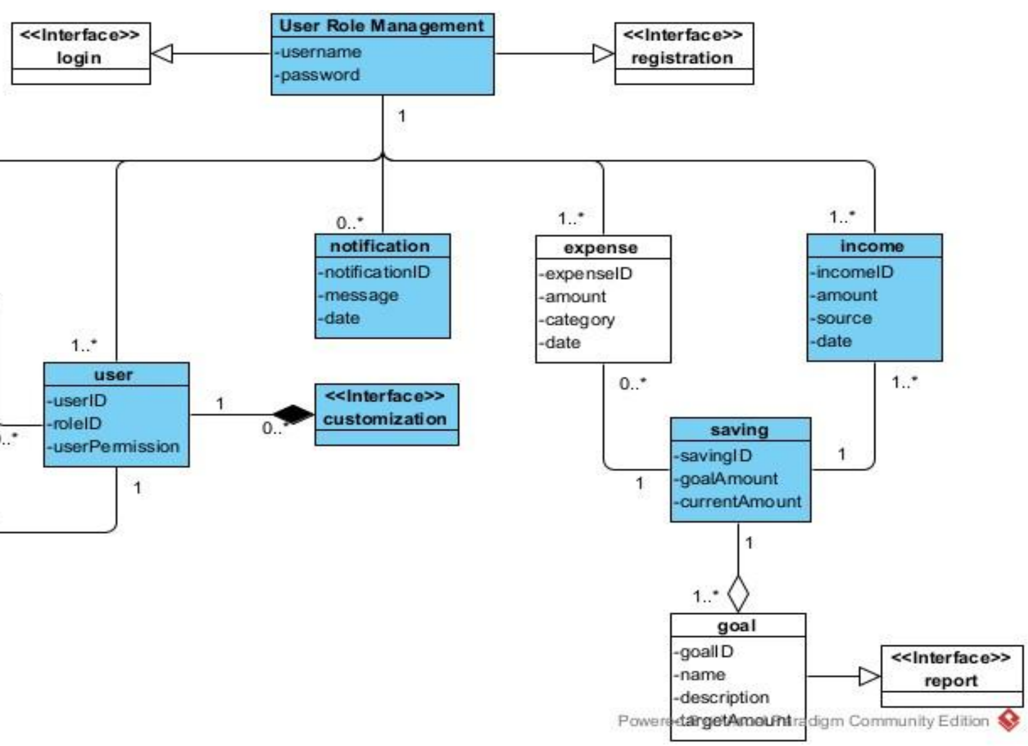
Responsibilities:

Name	Collaborator
Send expense alerts	Expense
Alert savings milestones	Saving

3.2.3 Design Class Diagram(Expense Subsystem)



3.3.1 Income Subsystem



3.3.2 Class Responsibility Collaboration cards (CRC cards)

COMP246-408 Budget Planner

User Role Management

Super Classes:

Sub Classes:

Description: Responsible for the management of user authentication, roles, and permissions.

Attributes:

Name	Description
username	Stores the username.
password	Stores the encrypted password.
roleID	Identifier for user role (e.g., admin, standard user).
userPermission	Defines level of access for each user.

Responsibilities:

Name	Collaborator
Manage user permissions	User, Customization interface

User

Super Classes:

Sub Classes:

Description: Represents individuals using the system.

Attributes:

Name	Description
userID	Unique identifier for each user.
roleID	Identifier for user role (e.g., admin, standard user).
userPermission	Defines level of access for each user.

Responsibilities:

Name	Collaborator
Manage user permissions	User Role Management
Customize settings	Customization interface

Income

Super Classes:

Sub Classes:

Description: Manages and records all sources of income and the details associated with each source of income.

Attributes:

Name	Description
incomeID	Unique identifier for each income.
amount	Monetary value of income received.
source	Source of income.
date	recorded date of income.

Responsibilities:

Name	Collaborator
Record income	
Categorize income	
Notify updates	Notification

Saving

Super Classes:

Sub Classes:

Description: Manages user savings, including goals and up to date progress towards goals.

Attributes:

Name	Description
savingID	Unique identifier for each savings entry.
goalAmount	Target amount set by user.
currentAmount	Current savings amount.

Responsibilities:

Name	Collaborator
Track savings progress	
Update savings balance	
Notify goal progress	Notification

Customization interface

Super Classes:

Sub Classes:

Description: User interface customization based on user preferences.

Attributes:

Name	Description

Responsibilities:

Name	Collaborator
Manage customization settings	User
Return success or failure status of customization	User

Notification

Super Classes:

Sub Classes:

Description: Handles alerts and reminders. Manages messages.

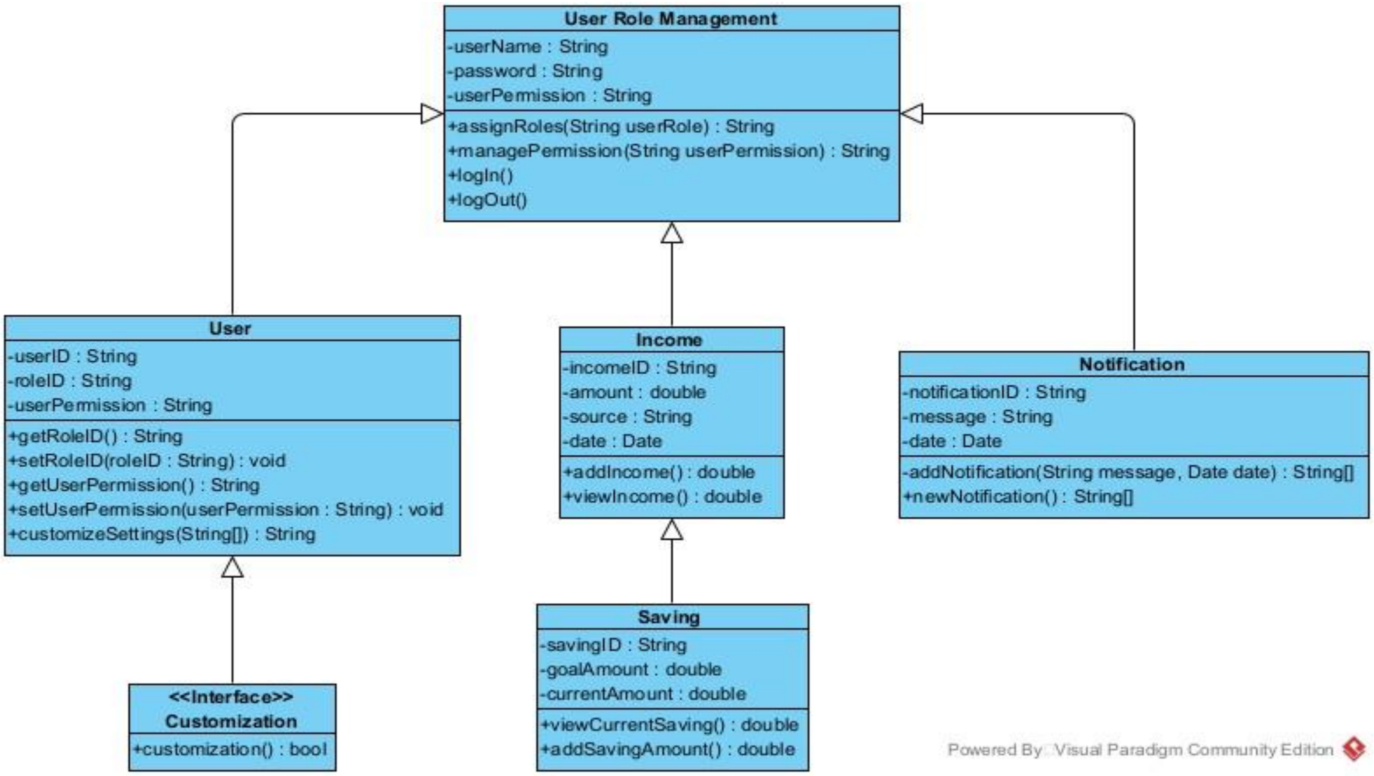
Attributes:

Name	Description
noticationID	Unique identifier for each notification.
message	Content of notification.
date	Date of notification.

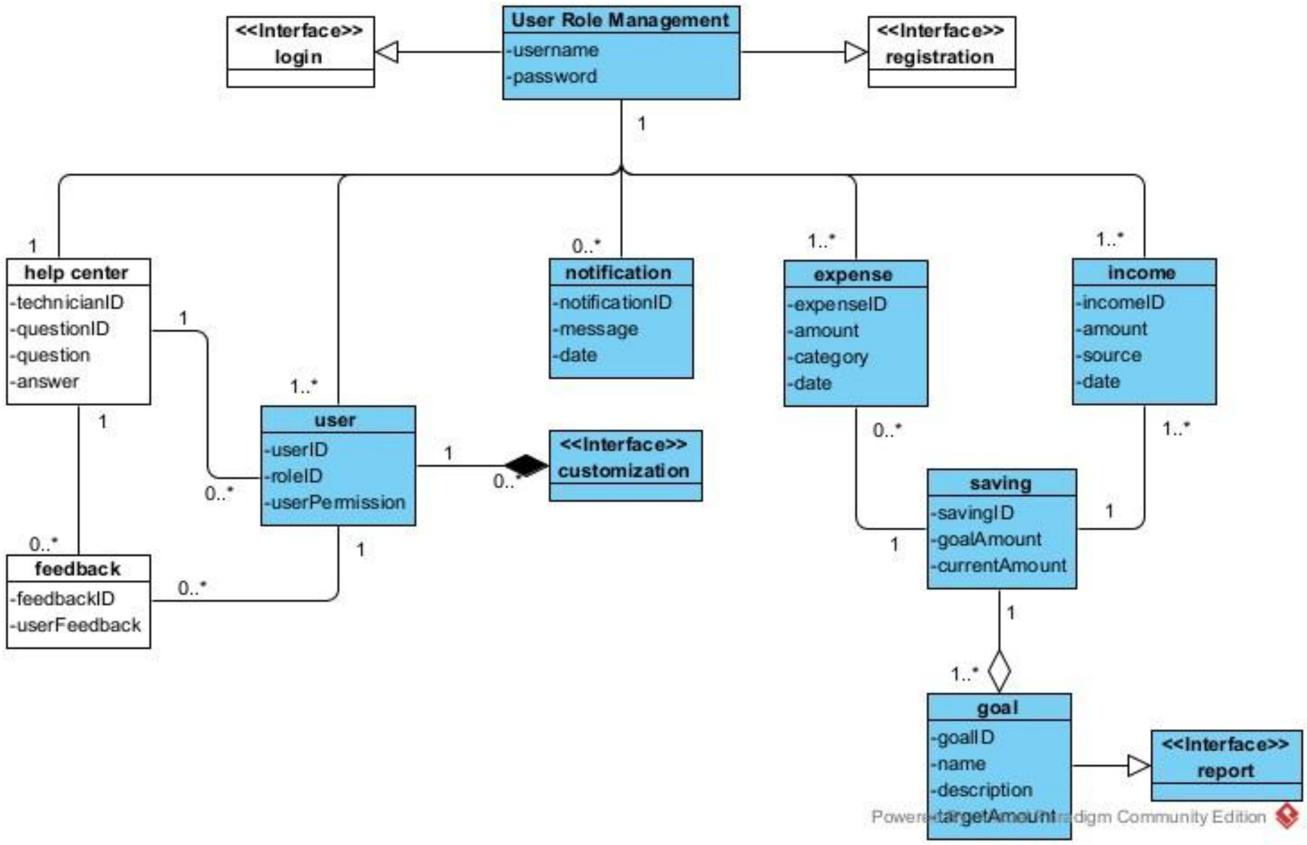
Responsibilities:

Name	Collaborator
Send expense alerts	Expense
Alert savings milestones	Saving

3.3.3 Design Class Diagram(Income Subsystem)



3.4.1 Goal Subsystem



3.4.2 Class Responsibility Collaboration cards (CRC cards)

COMP246-408 Budget Planner

User

Super Classes: None

Sub Classes: None

Description: The User class represents individuals using the system, storing essential information about each user, including roles and permissions. It interacts with various subsystems to manage user-specific data and actions.

Attributes:

Name	Description
userID	Unique identifier for each user.
roleID	Identifier for the user's role (e.g., admin, standard user).
userPermission	Defines the level of access or permissions associated with the user.

Responsibilities:

Name	Collaborator
Store user information	User Role Management
Manage user permissions	User Role Management
Customize settings	Customization Interface

Customization

Super Classes: None

Sub Classes: None

Description: The Customization interface allows for user-specific customization settings, enabling the system to save or retrieve a user's preferences.

Attributes:

Name	Description

Responsibilities:

Name	Collaborator
Manage user customization settings	User
Return success or failure status of customization	User

Saving

Super Classes: None

Sub Classes: None

Description: Manages user savings, including goals and current progress towards those goals. It helps users track their savings and the amount set aside for different goals.

Attributes:

Name	Description
savingID	Unique identifier for each saving entry.
goalAmount	Target amount set by the user for a particular saving goal.
currentAmount	Current amount saved towards the goal.

Responsibilities:

Name	Collaborator
Track savings progress	
Update savings balance	
Notify goal progress	Notification

User Role Management

Super Classes:

Sub Classes:

Description: Responsible for the management of user authentication, roles, and permissions.

Attributes:

Name	Description
username	Stores the username.
password	Stores the encrypted password.
roleID	Identifier for user role (e.g., admin, standard user).
userPermission	Defines level of access for each user.

Responsibilities:

Name	Collaborator
Manage user permissions	User, Customization interface

Income	
Super Classes: None	
Sub Classes: None	
Description: It is responsible for tracking and managing all income entries for the user. It records details of each income source and the amount received, allowing users to keep track of their income over time.	
Attributes:	
Name	Description
incomeID	Unique identifier for each income entry.
amount	The monetary value of the income received.
source	The source of income (e.g., salary, freelance work).
date	The date the income was recorded
Responsibilities:	
Name	Collaborator
Record income	
Categorize income	
Notify updates	Notification

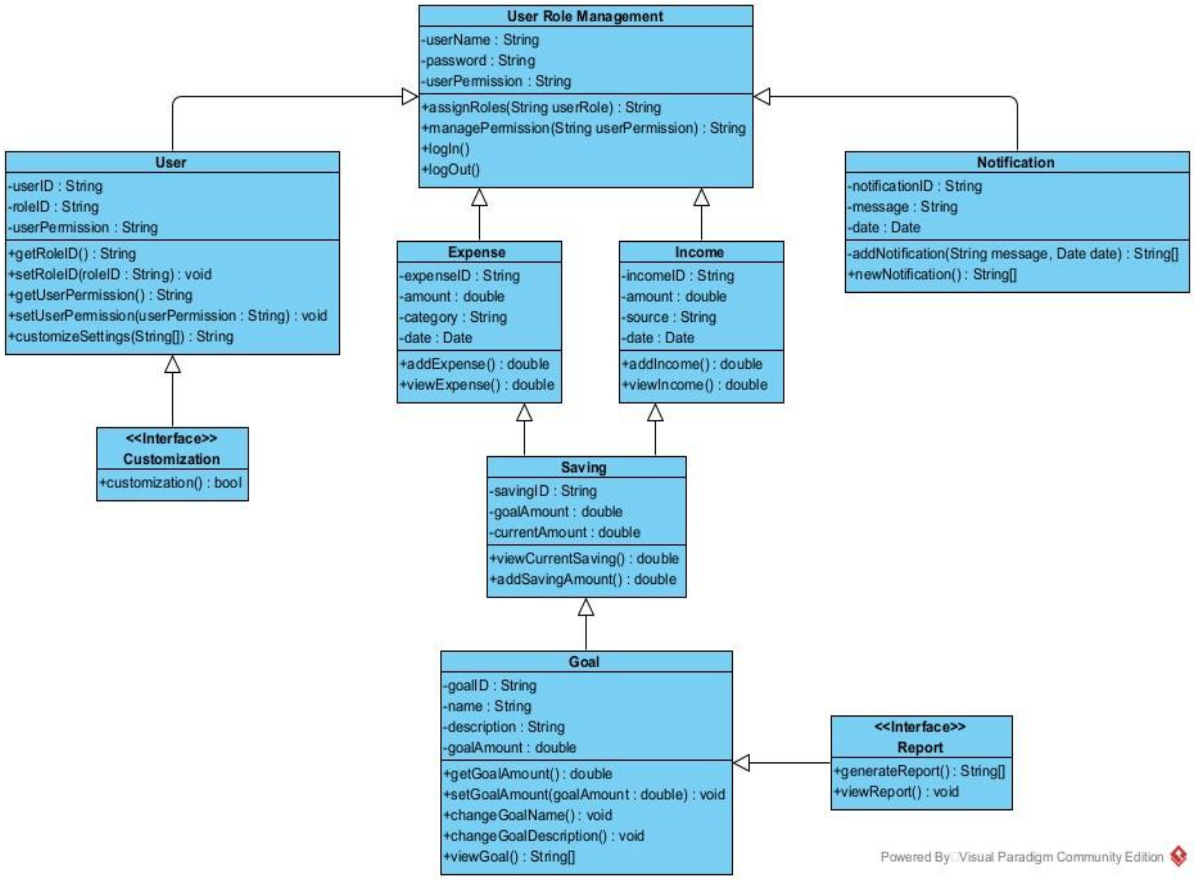
Expense	
Super Classes: None	
Sub Classes: None	
Description: The Expense class is responsible for recording and managing financial expenditures by the user. It tracks details of each expense, including amount, category, and date.	
Attributes:	
Name	Description
expenseID	Unique identifier for each expense entry.
amount	The monetary value of the expense.
category	The category to which the expense belongs (e.g., groceries, rent).
date	The date the expense was recorded.
Responsibilities:	
Name	Collaborator
Record expense	
Categorize expense	
Notify of overspending	Notification

Goal	
Super Classes: None	
Sub Classes: None	
Description: The Goal class manages financial goals set by the user, such as saving for a particular purpose. It tracks progress towards each goal and allows users to adjust or update their targets as needed.	
Attributes:	
Name	Description
goalID	Unique identifier for each financial goal.
name	Name or title of the goal
description	Brief description of the goal
targetAmount	The monetary target the user aims to reach for the goal.
Responsibilities:	
Name	Collaborator
Create goal	User
Track progress	Saving
Update or delete goal	Saving
Notify goal milestones	Notification

Report interface	
Super Classes: None	
Sub Classes: None	
Description: The Report class is responsible for generating and managing reports related to financial goals. It retrieves relevant data from the Goal class to provide summaries, progress tracking, and detailed goal reports.	
Attributes:	
Name	Description
reportID	Unique identifier for each report.
goalID	ID of the goal associated with this report.
reportDate	Date the report was generated.
progress	Stores the progress percentage towards the goal.
details	Contains detailed information or summary about the goal report.
Responsibilities:	
Name	Collaborator
Generate a report based on goal data	Goal
Retrieve details about a specific goal	Goal
Format and display goal report	Goal
Track progress towards a goal's target	Goal
Save or export the generated report	Goal

3.4.3 Design Class Diagram(Goal Subsystem)

COMP246-408 Budget Planner

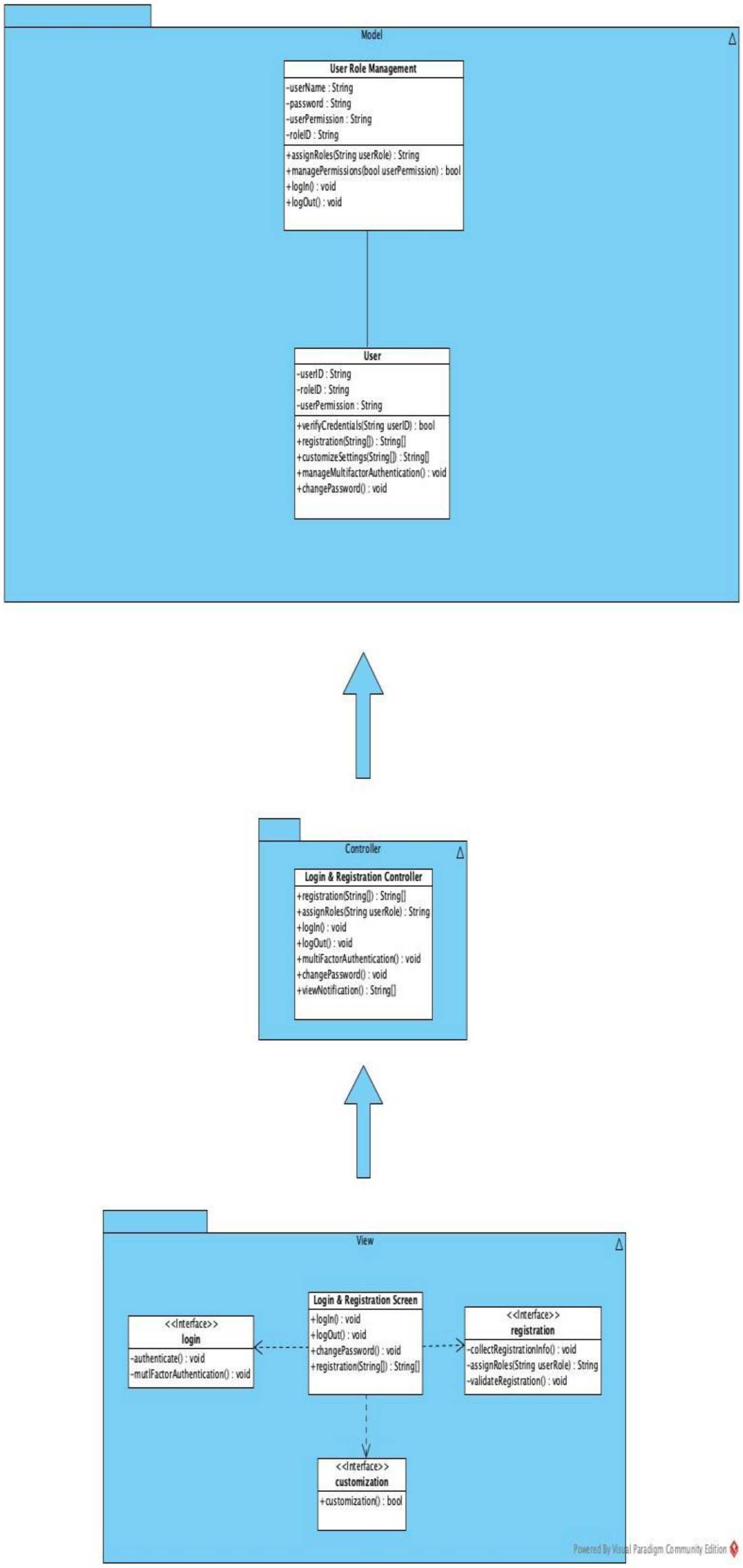


Section 4. Framework MVC (Model View Controller)

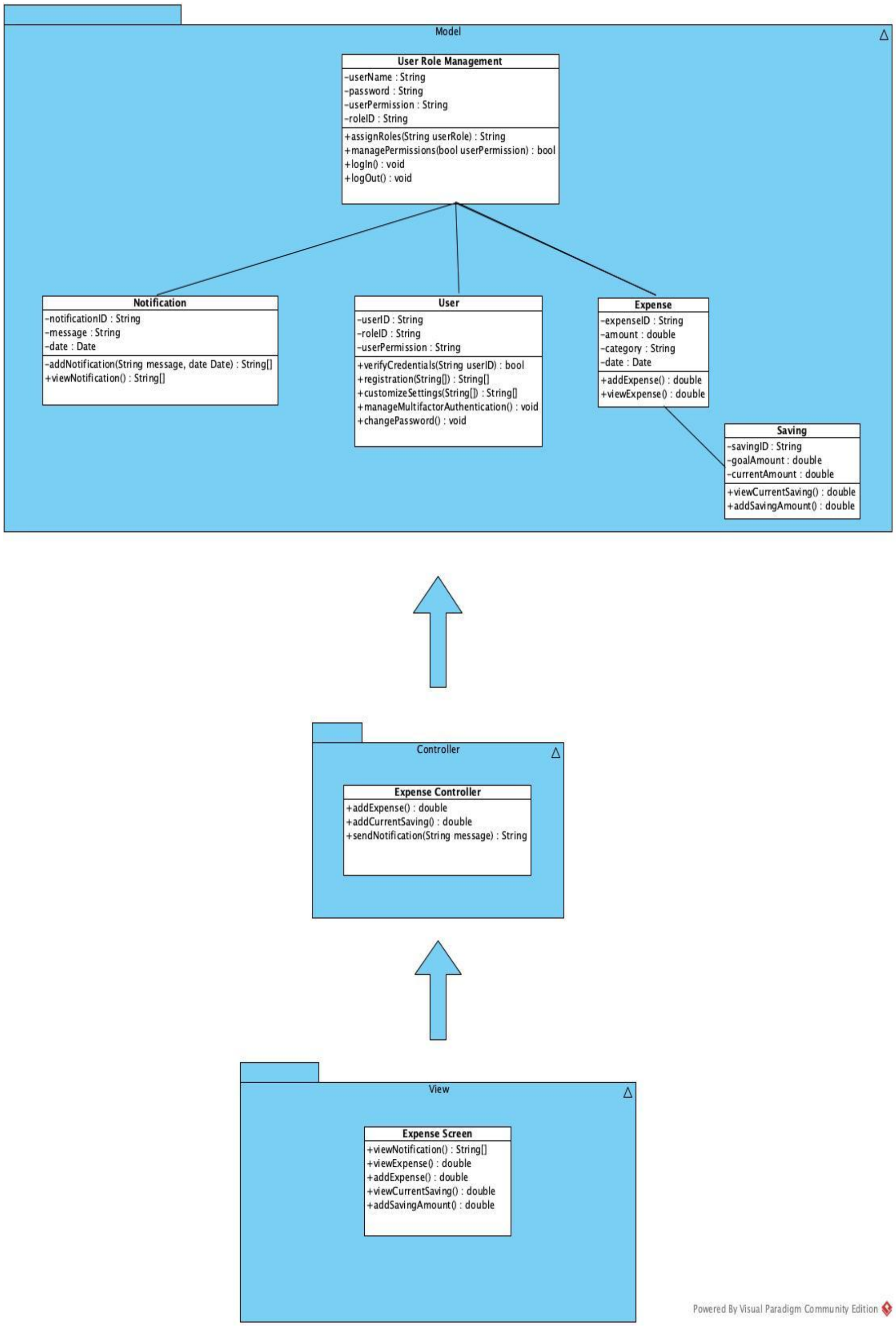
4.1 MVC pattern diagram to include

4.1.1 Login & Registration Subsystem

COMP246-408 Budget Planner

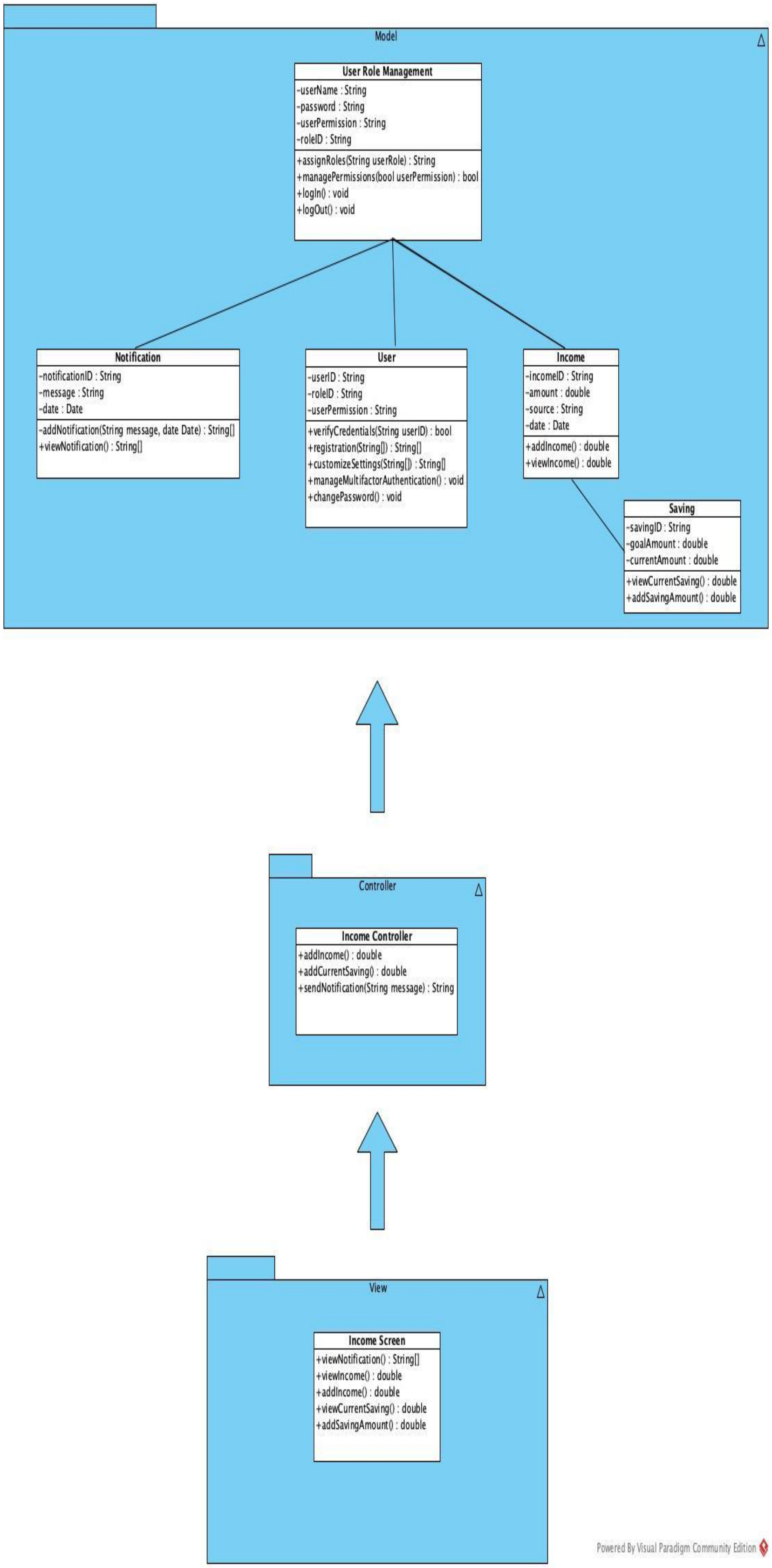


4.1.2 Expense Subsystem



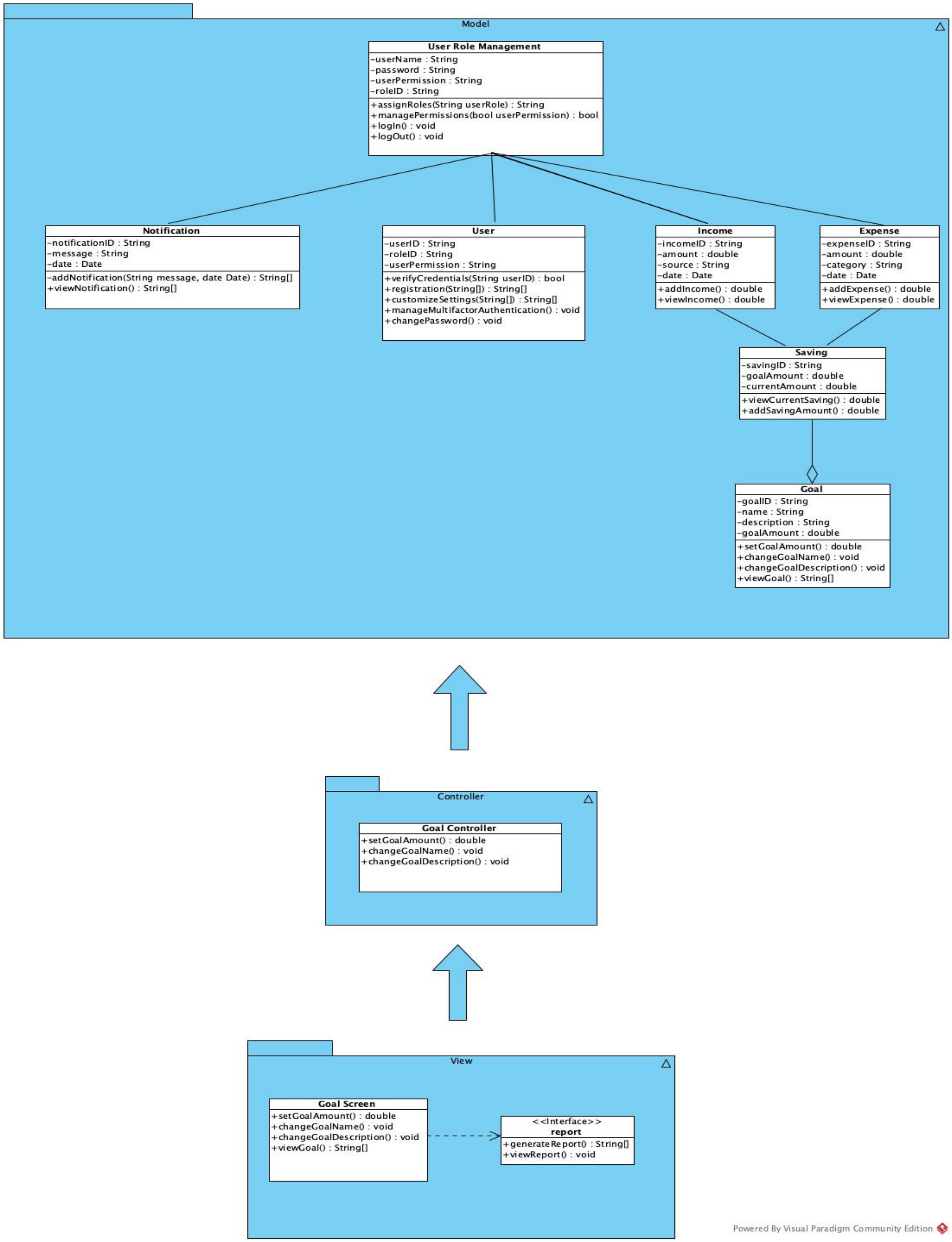
4.1.3 Income Subsystem

COMP246-408 Budget Planner



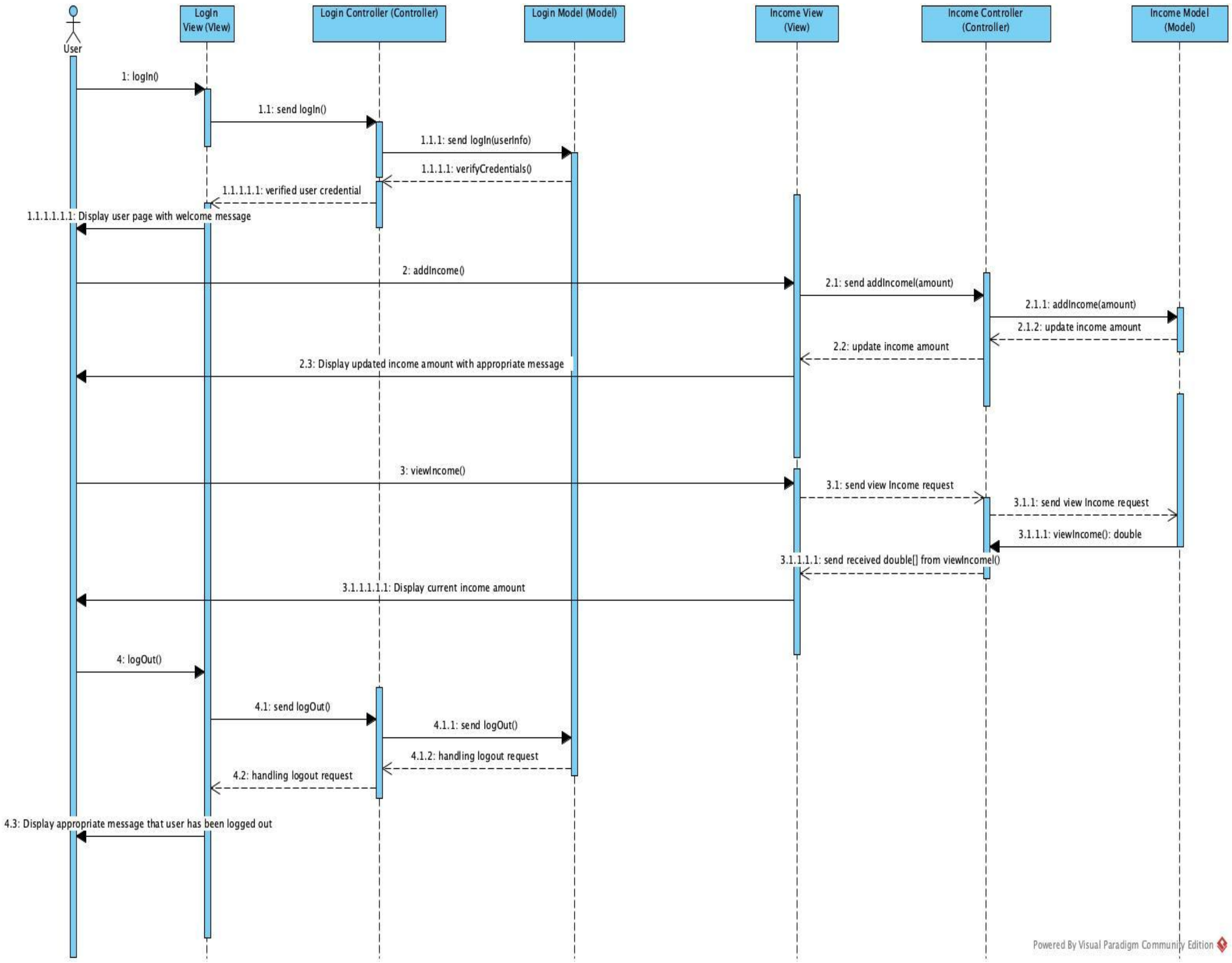
4.1.4 Goal Subsystem

COMP246-408 Budget Planner

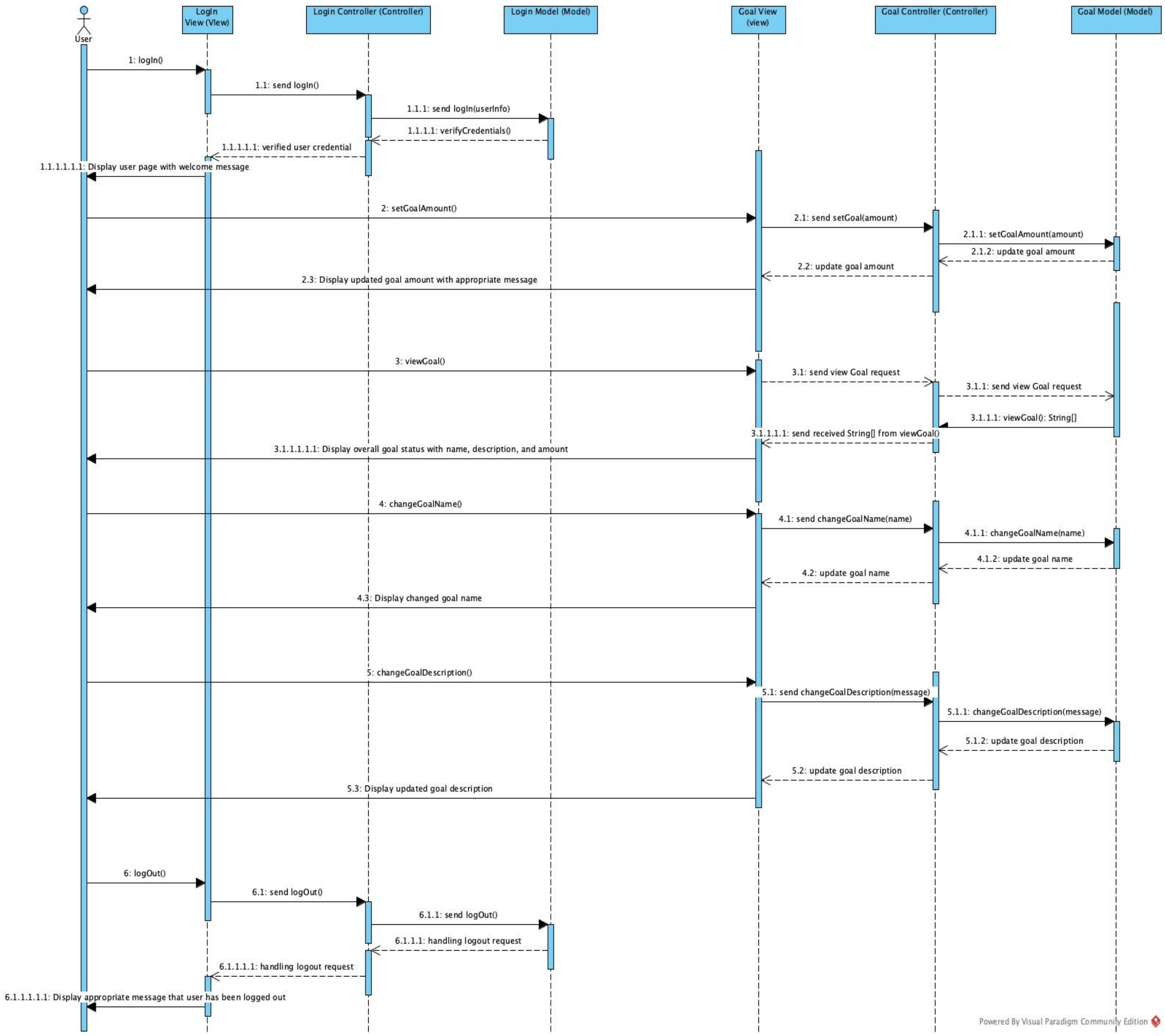


4.2 Full Sequence Diagrams

4.2.1 User Log In & Income System

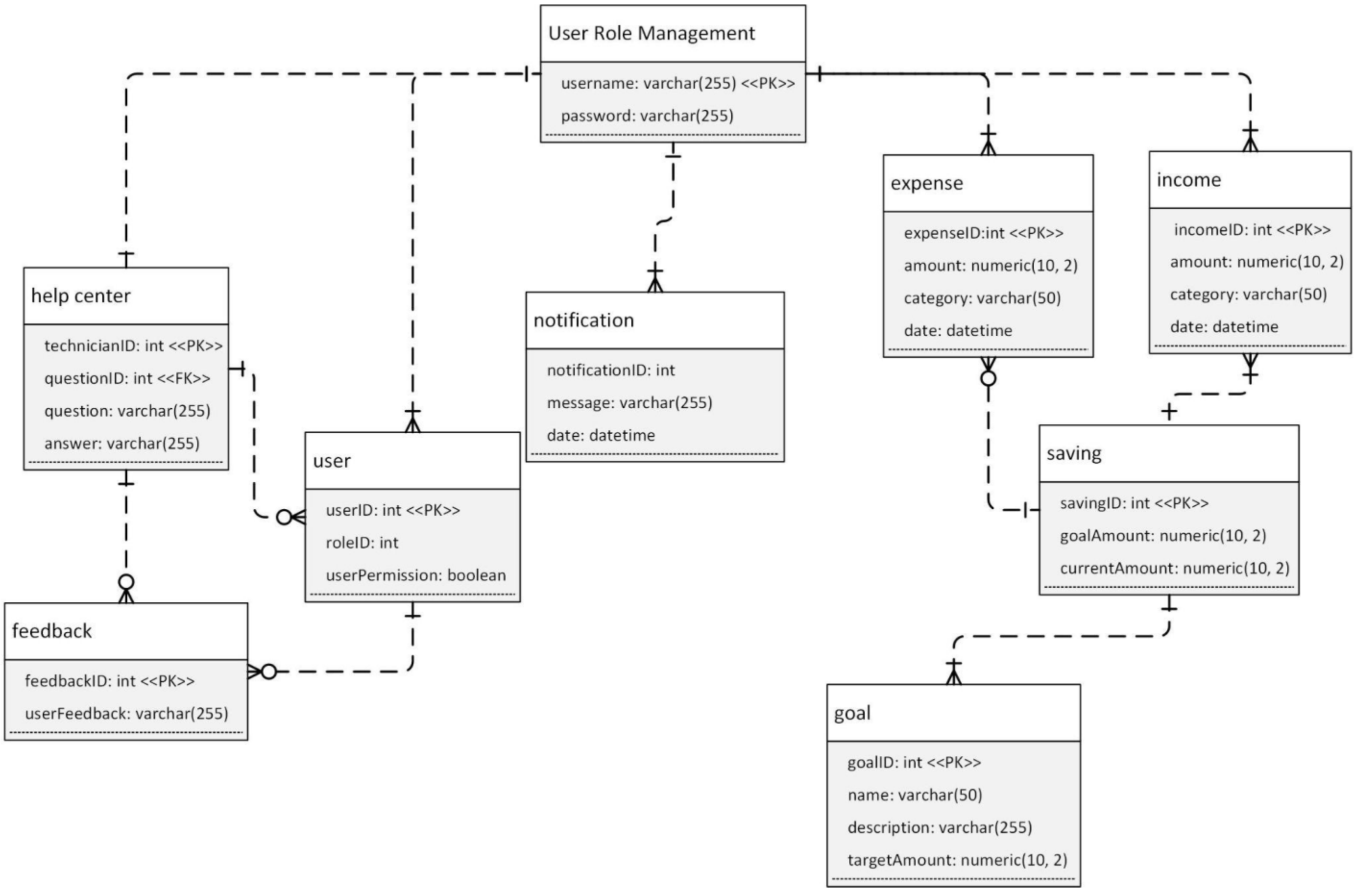


4.2.2 User Log In & Goal System



Section 5. Data Layer

5.1 Database Schema



5.2 Technology List Update

Our system will be developed as a mobile application, accessible on all smartphones and devices. The tools selected for development are as follows:

Client Side (Front-End – GUI):

- **React Native**

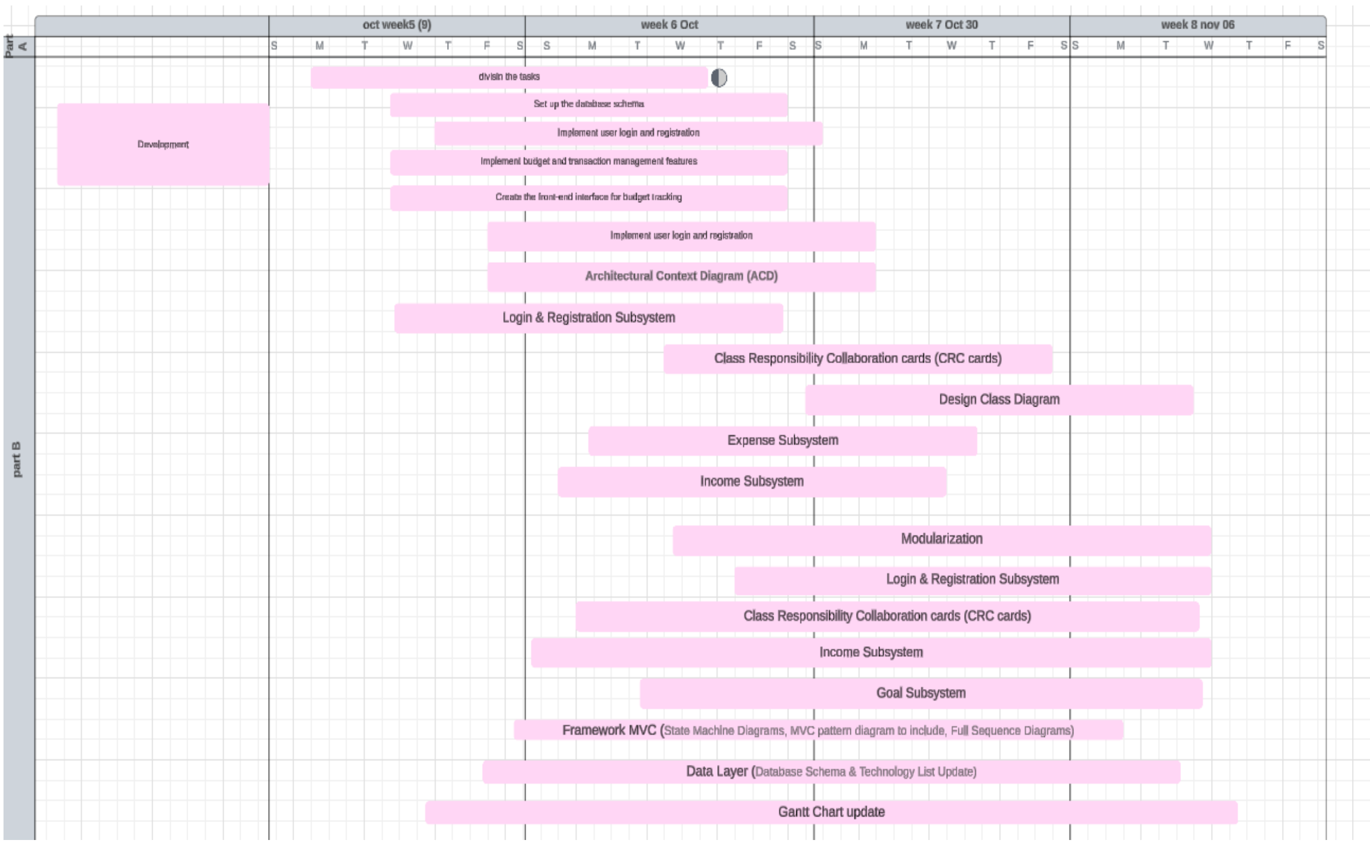
Business Logic (Middle Layer – Class Methods, etc.):

- **JavaScript/TypeScript**

Back-End (Database):

- **Firebase Firestore**
- **Firebase Authentication**

Section 6. Gantt Chart update



Part C: System Design Documents

Section 1. Corrections do Design Specifications Part B.

1.1 Corrections on Architectural context diagram(section 2.2)

- Mentioned Data Storage/Backup as subordinate system.

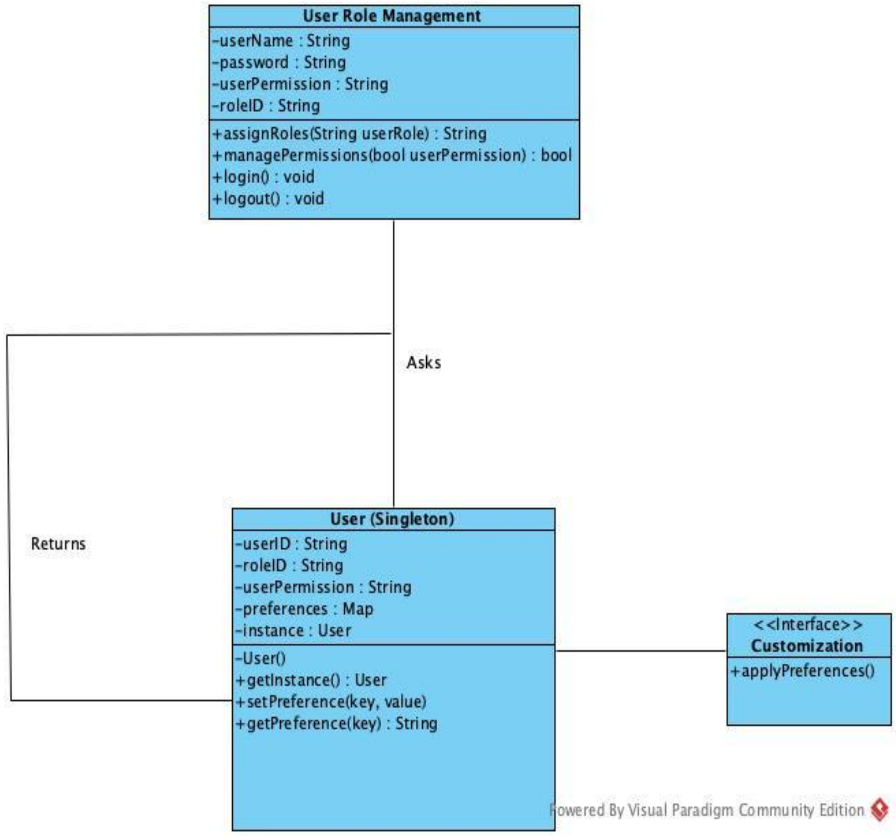
Section 4.1.1: Notification class has been removed in the Login & Registration Subsystem
Section 4.2.1: User Log In & Income System full sequence diagram has been added.
Section 4.2.2: User Log In & Goal System full sequence diagram has been updated.

Section 3.2.2: In the Expense subsystem, the Notification CRC has been updated. Goal has been removed as a collaborator.
Section 3.3.2: In the Income subsystem, the Notification CRC has been updated, and Goal has been removed as a collaborator.
Section 3.1.2: All the CRC cards are consolidated on a single page.
In 3.2.1, In Expense Subsystem; Login Interface and Registration Interface has been removed as responsibility from User Role Management crc.

Section 2. Software Design Patterns

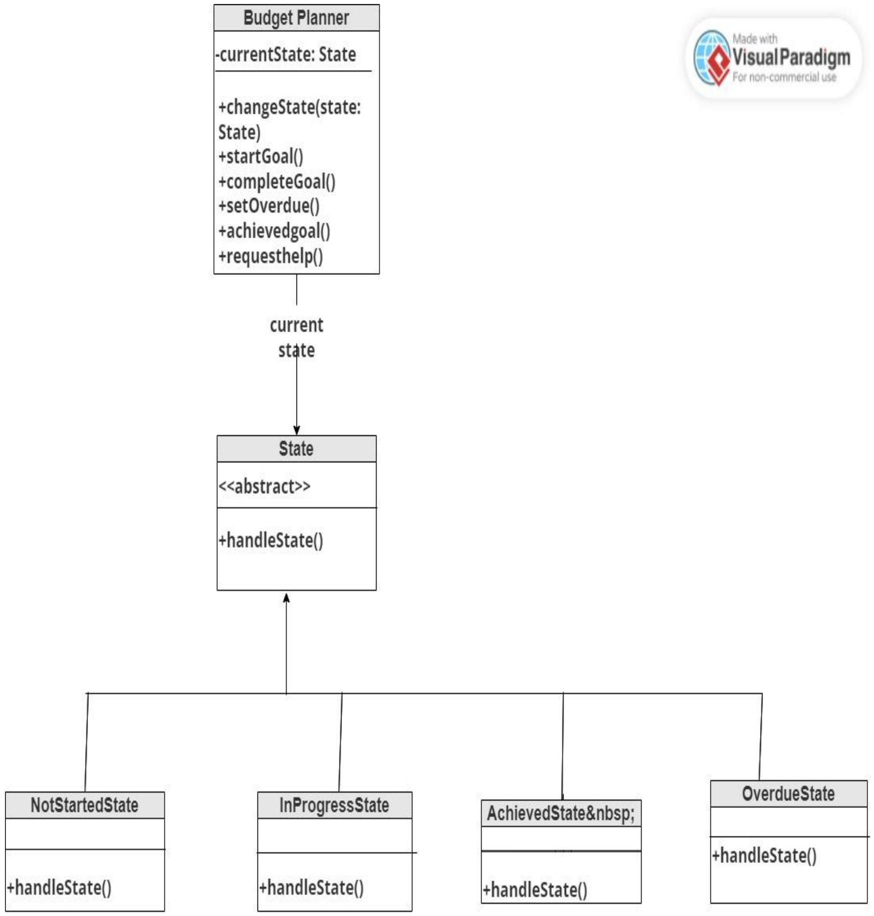
2.1 Singleton Pattern

Name:	Singleton Pattern
Problem:	The current diagram indicates that several parts of the system require access to user-specific customization preferences. And maintaining multiple user instances can lead to inconsistencies when retrieving or updating user data.
Solution:	By implementing the Singleton Pattern for the User class, we ensure that all components, including Customization, access the same User instance through a getInstance() method. Also, it enables storing user specific customization preferences in the User singleton.

<table>
<tr><td>Graph:</td><td>
</td></tr>
<tr><td>Benefits and consequences:</td><td>Benefits:
Only one instance of the User class is created
Ensures all components use the same User instance
Customization settings are stored and managed within the Singleton User instance
The Customization interface can be easily retrieve preferences using the getInstance() method
Consequences:
Customization becomes tightly coupled with the User class
It makes testing and debugging more challenging because of the global state</td></tr>
</table>

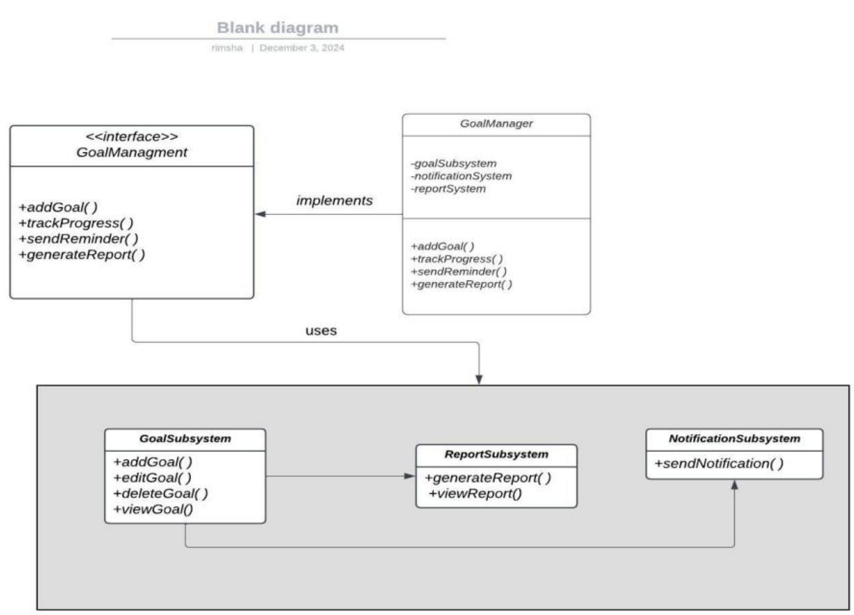
2.2 State Pattern

Name	State Pattern
Problem	The budget planner application might be managing goals in a single class, depending on complex conditionals to determine the state of a goal (eg: checking whether the goal is completed, or if it is overdue, and so on.), This approach may lead to complexity to maintain code as more states are introduces or logic changes
Solution	The state pattern can be used to separate the behavior for each goal state into distinct state classes.So that each class will handle the logic and transitions relevant to its specific state. The goal class will hold a reference to the current state and delegate actions to the appropriate state class. The steps goal state changes are NotStarted, InProgressState,AchievedState, and OverdueState

Graph	
Benefits and consequences	Benefits: • Improved Modularity: Each goal state has its own class, which encapsulates the logic for handling that state. • Easier to add new states: If new states are required (eg: paused or extendedDeadline), they can be added without changing existing code. • Clear Logic: Each state class handles specific behavior for that state, improving readability and reducing the risk of errors when managing goal states. Consequences: • Increased number of classes: Each state is encapsulated in its own class, which leads to an increase in number of classes, especially if many states are

	required ● Potential Performance Impact: Although the performance overhead is minimal, frequent state transitions of goals begin tracked could lead to slight performance issues due to object creation and state switching.

2.3 Facade

Name	Facade
Problem	Managing interactions between multiple subsystems (such as Expense, Income, Goals, and Database) directly can result in tight coupling and make the system harder to maintain or extend. This complexity can confuse users or developers trying to access core functionalities in a streamlined manner.
Solution	The **Facade Pattern** provides a unified interface through the `GoalManager` class, allowing users to perform key operations like adding goals, tracking progress, sending reminders, and generating reports without directly interacting with individual subsystems. This simplifies the process by abstracting the complexity of underlying operations.
Graph	

COMP246-408 Budget Planner

Code (Java)

GoalSubsystem.java > ...

```java
public class GoalSubsystem {
    public void addGoal() {
        System.out.println("Goal added successfully.");
    }

    public void editGoal() {
        System.out.println("Goal edited successfully.");
    }

    public void deleteGoal() {
        System.out.println("Goal deleted successfully.");
    }

    public void viewGoal() {
        System.out.println("Displaying goal details.");
    }
}

```

GoalManager.java

```java
public class GoalManager implements GoalManagement {
    private GoalSubsystem goalSubsystem;
    private ReportSubsystem reportSubsystem;
    private NotificationSubsystem notificationSubsystem;

    public GoalManager() {
        goalSubsystem = new GoalSubsystem();
        reportSubsystem = new ReportSubsystem();
        notificationSubsystem = new NotificationSubsystem();
    }

    @Override
    public void addGoal() {
        goalSubsystem.addGoal();
        notificationSubsystem.sendNotification();
    }

    @Override
    public void trackProgress() {
        goalSubsystem.viewGoal();
        System.out.println(x:"Tracking progress of the goal.");
    }

    @Override
    public void sendReminder() {
        notificationSubsystem.sendNotification();
        System.out.println(x:"Reminder sent for the goal.");
    }
```

```java
    @Override
    public void generateReport() {
        reportSubsystem.generateReport();
    }
}

```

<table>
<tr>
<td></td>
<td>
GoalManagement.java > GoalManagement
<pre>
public interface GoalManagement {
    void addGoal();
    void trackProgress();
    void sendReminder();
    void generateReport();
}
</pre>
ReportSubsystem.java > ...
<pre>
public class ReportSubsystem {
    public void generateReport() {
        System.out.println(x:"Report generated successfully.");
    }

    public void viewReport() {
        System.out.println(x:"Displaying the report.");
    }
}
</pre>
NotificationSubsystem.java > ...
<pre>
public class NotificationSubsystem {
    public void sendNotification() {
        System.out.println(x:"Notification sent successfully.");
    }
}
</pre>
</td>
</tr>
<tr>
<td>Benefits and consequences</td>
<td>
Benefits:

Simplifies the interface for interacting with subsystems.
Reduces coupling between clients and subsystems.
Centralizes access, making the system easier to maintain and extend.

Consequences:

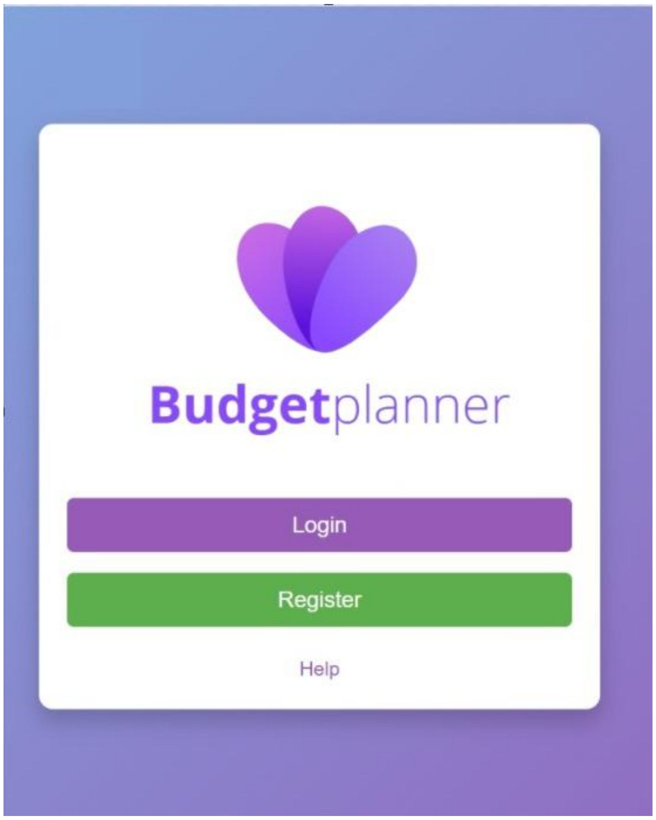
May add an additional layer of complexity if not needed.
Requires updates to the Facade class whenever the subsystems change.

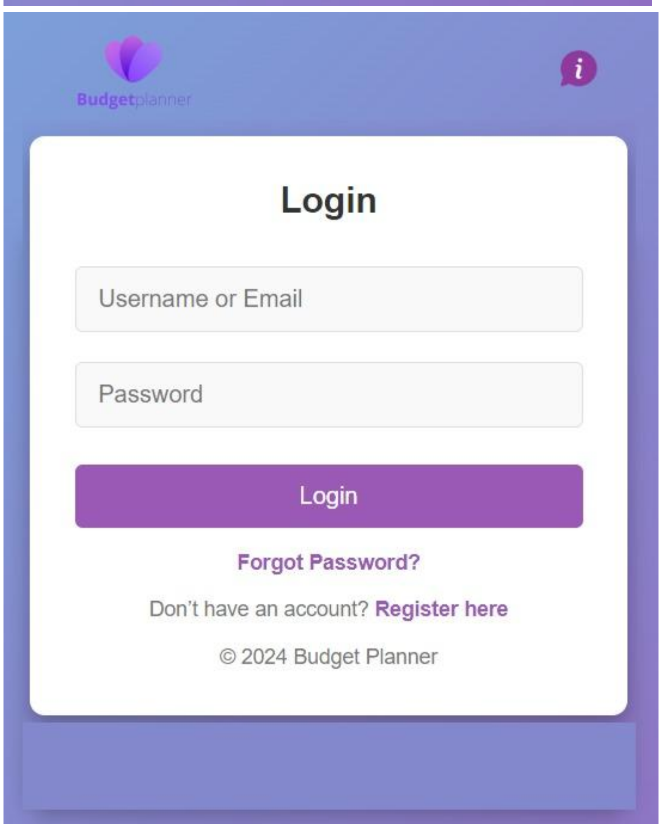
</td>
</tr>
</table>

Section 3. UI/UX design

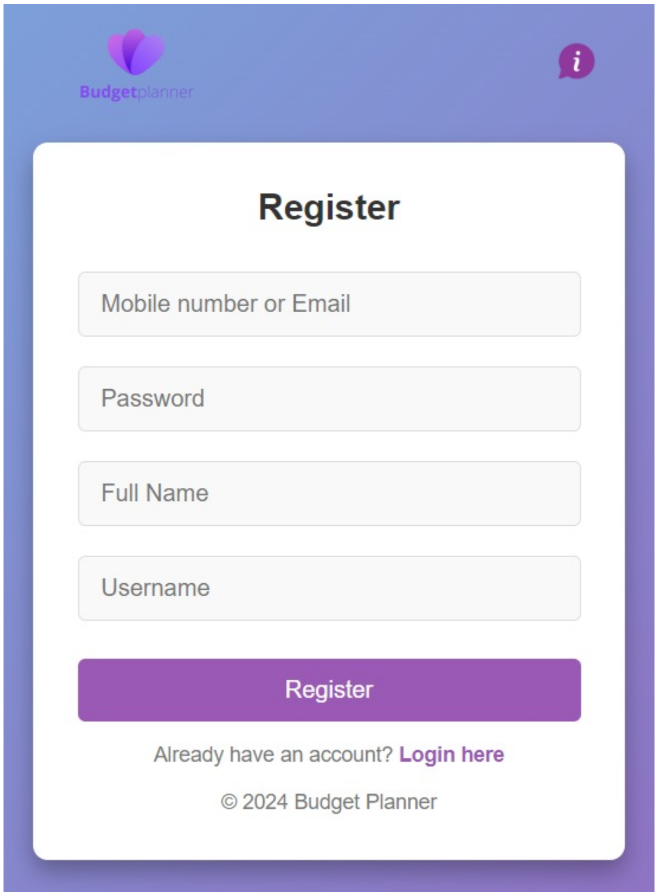
3.1 Home, Login & Registration

COMP246-408 Budget Planner

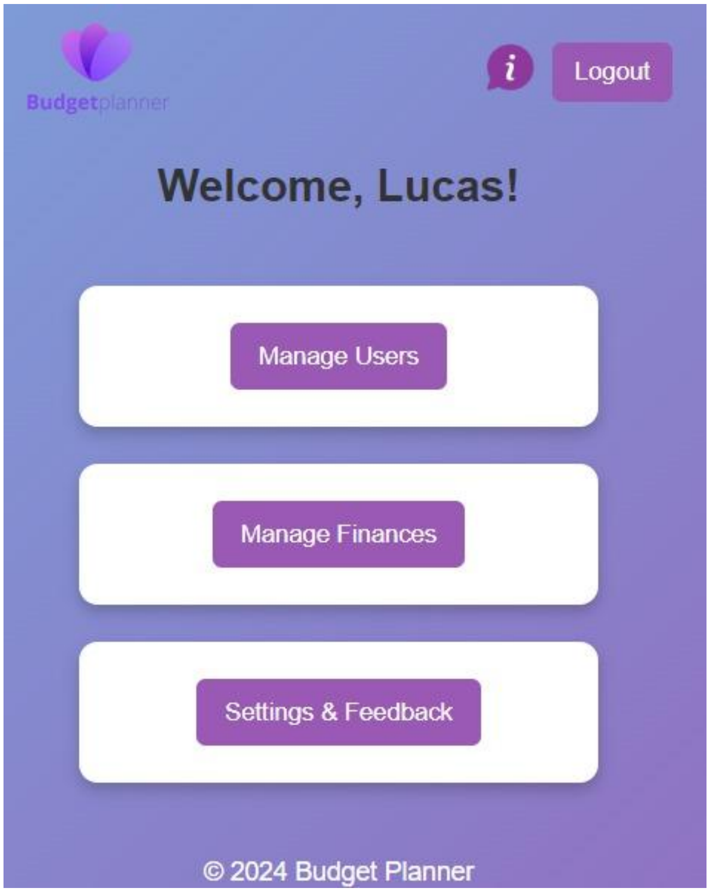




COMP246-408 Budget Planner

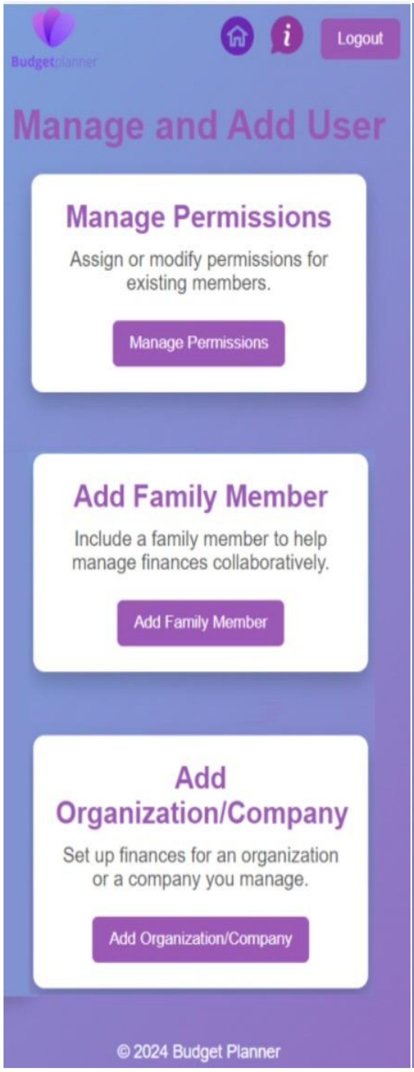


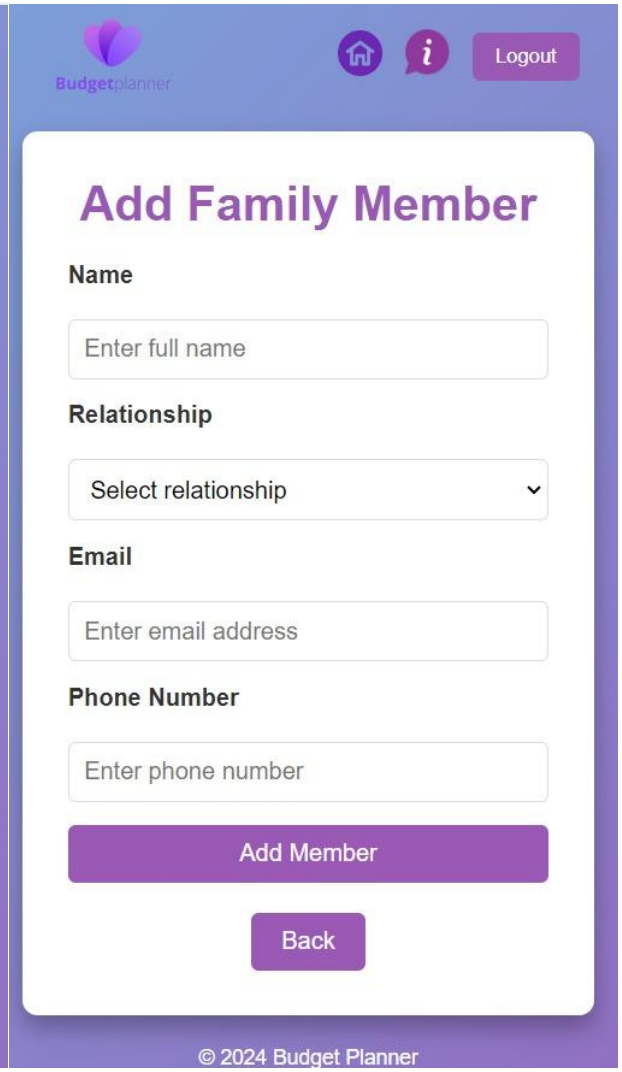
COMP246-408 Budget Planner



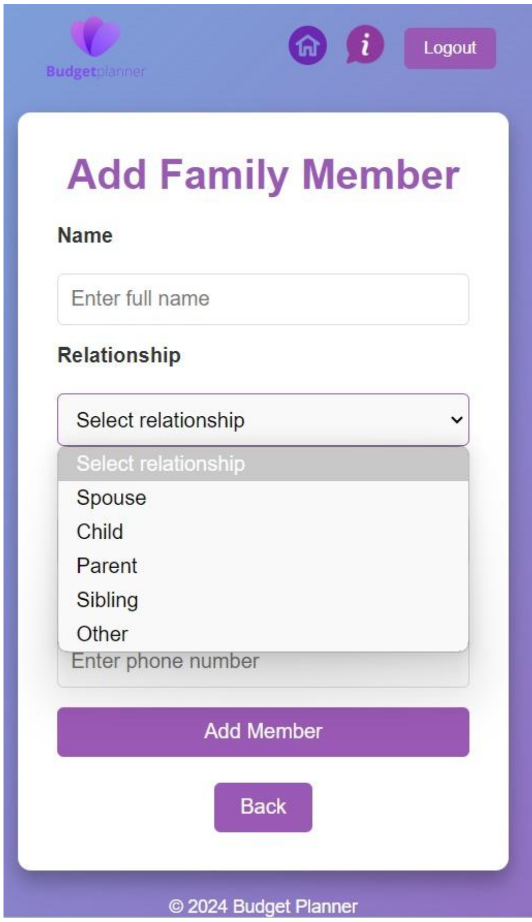
3.2 User(Manage & Add user):

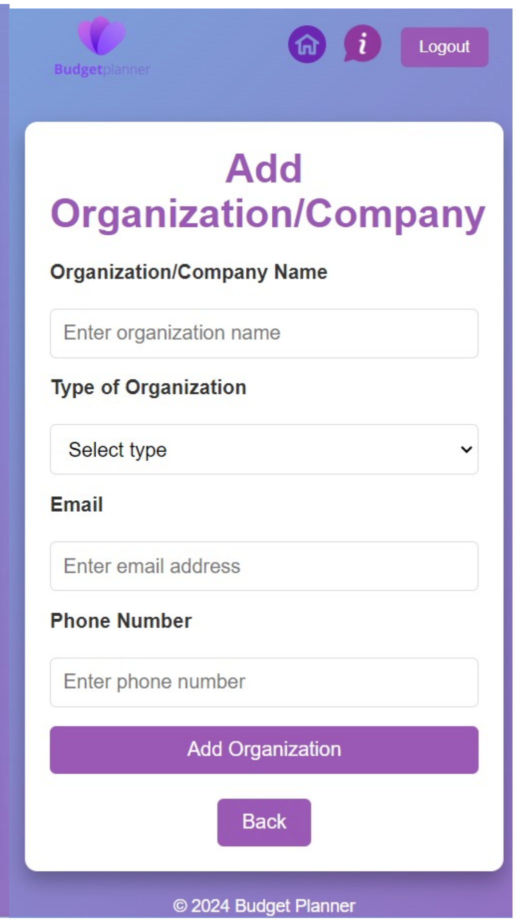
COMP246-408 Budget Planner



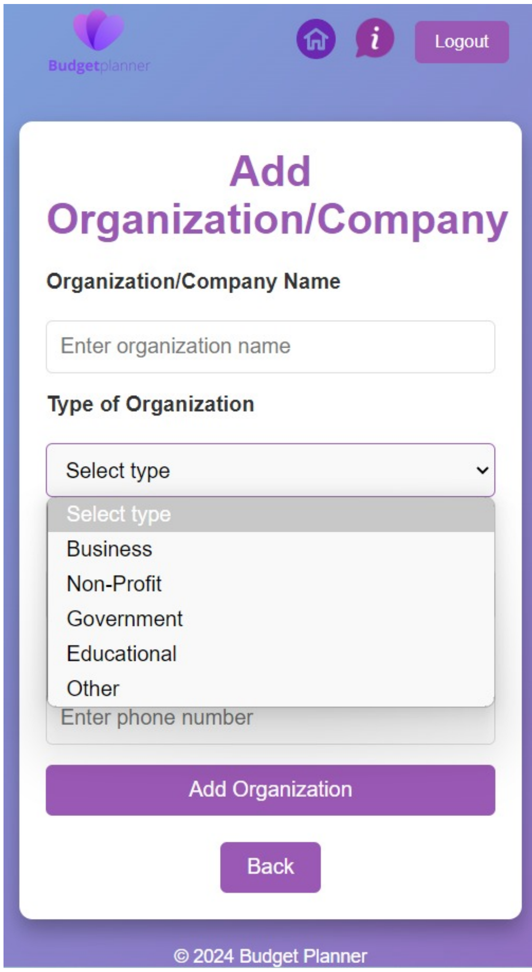


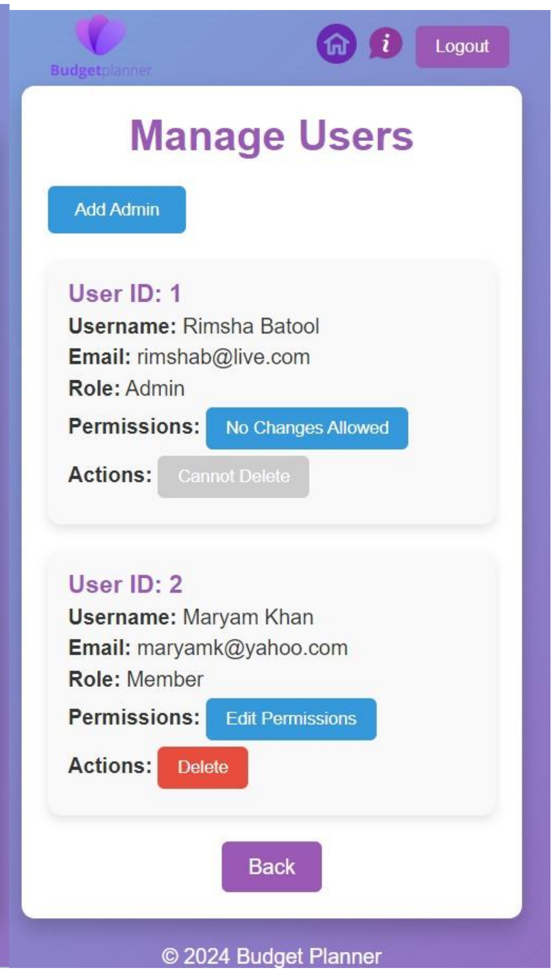
COMP246-408 Budget Planner





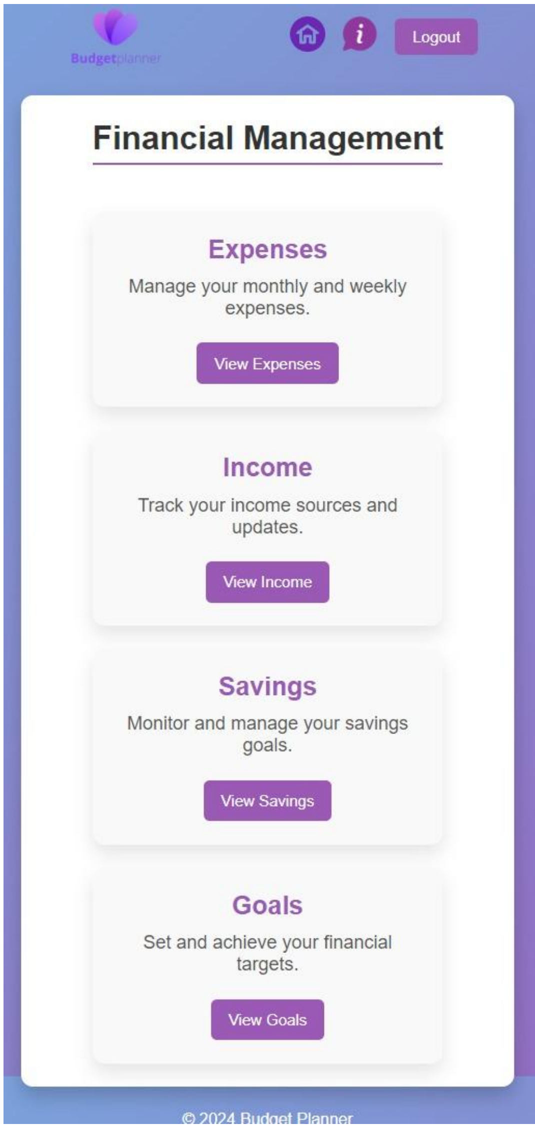
COMP246-408 Budget Planner

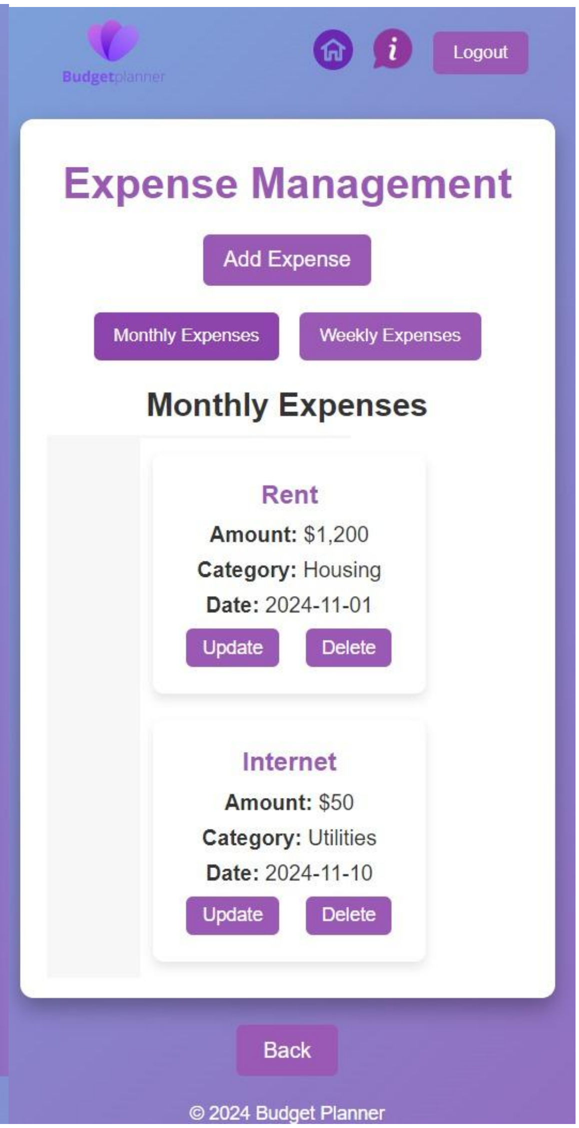




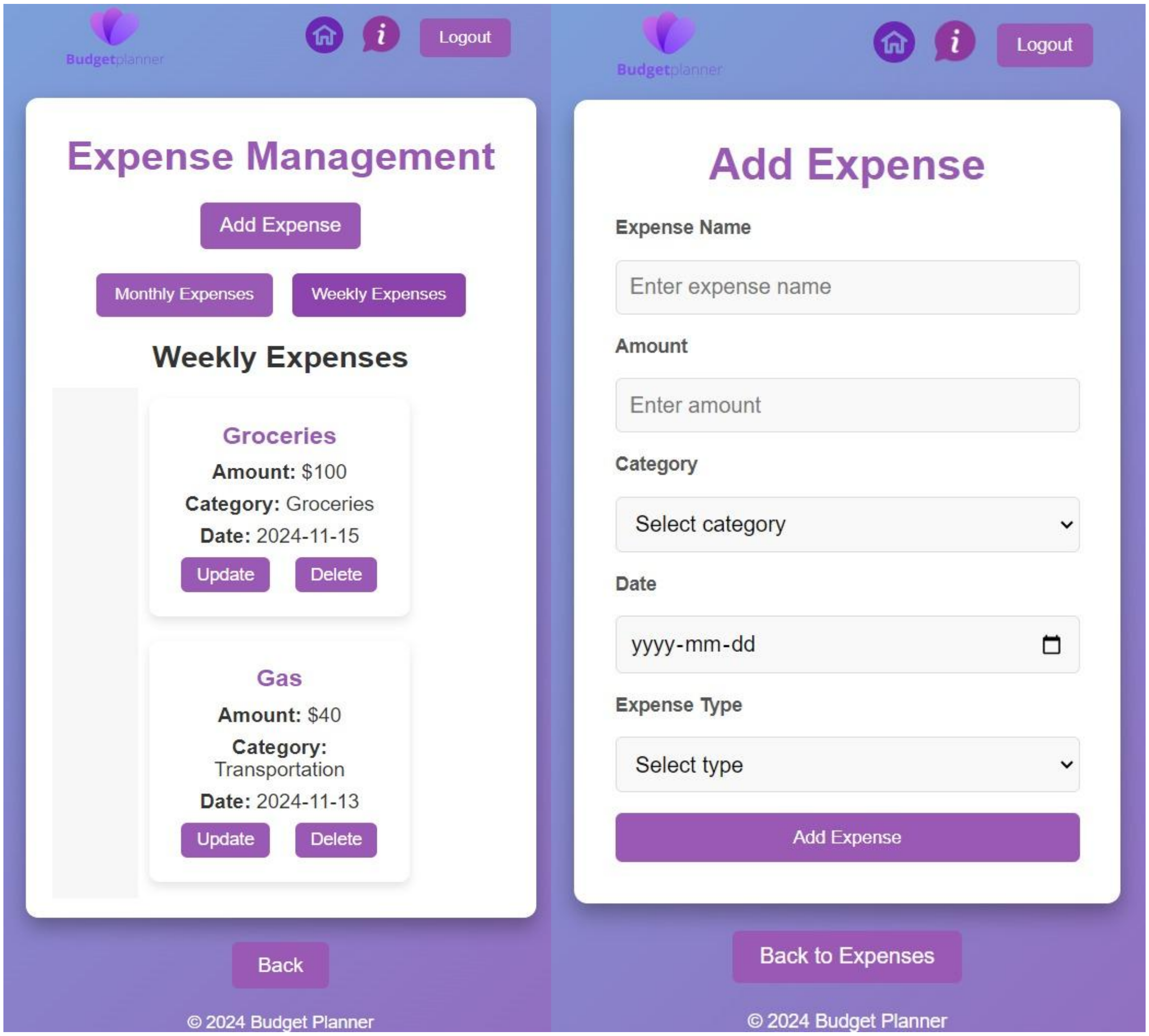
3.3 Expense(monthly and weekly):

COMP246-408 Budget Planner

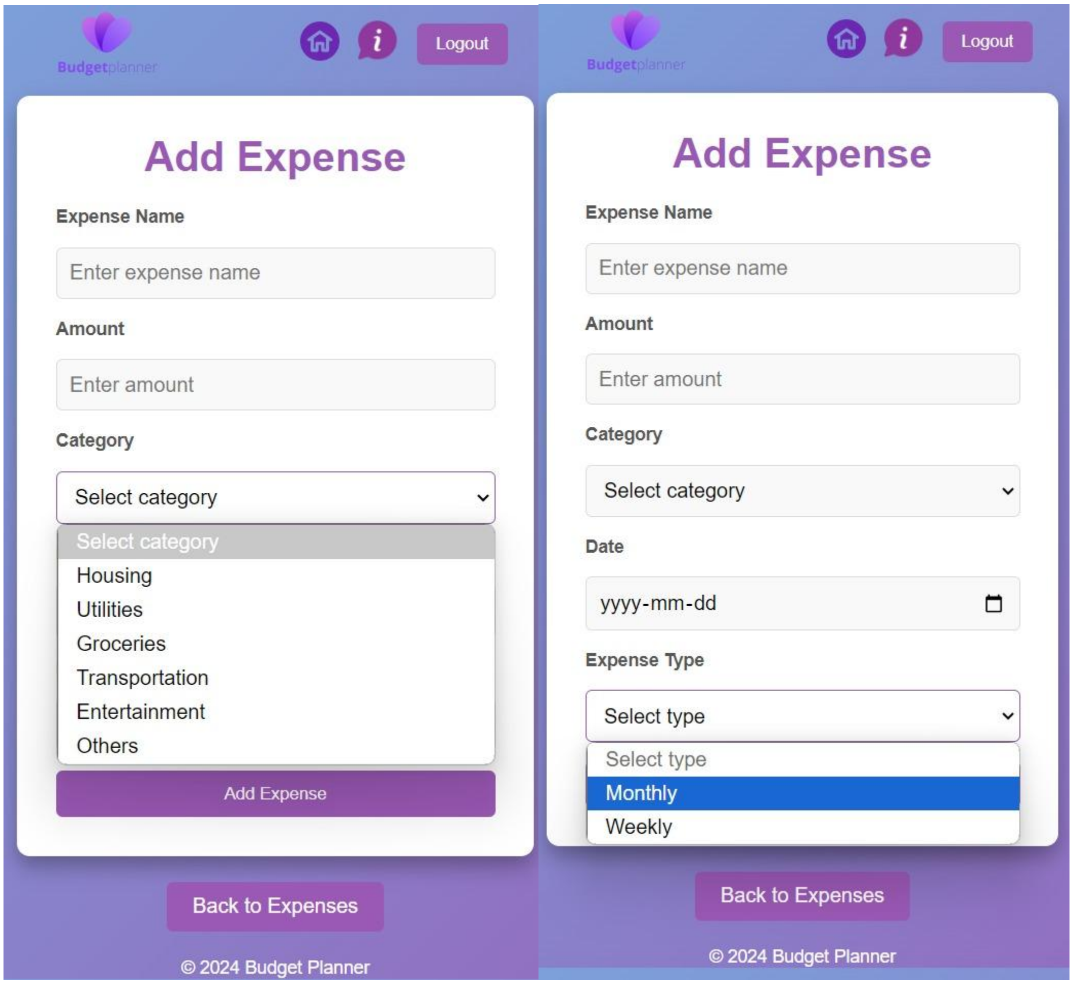




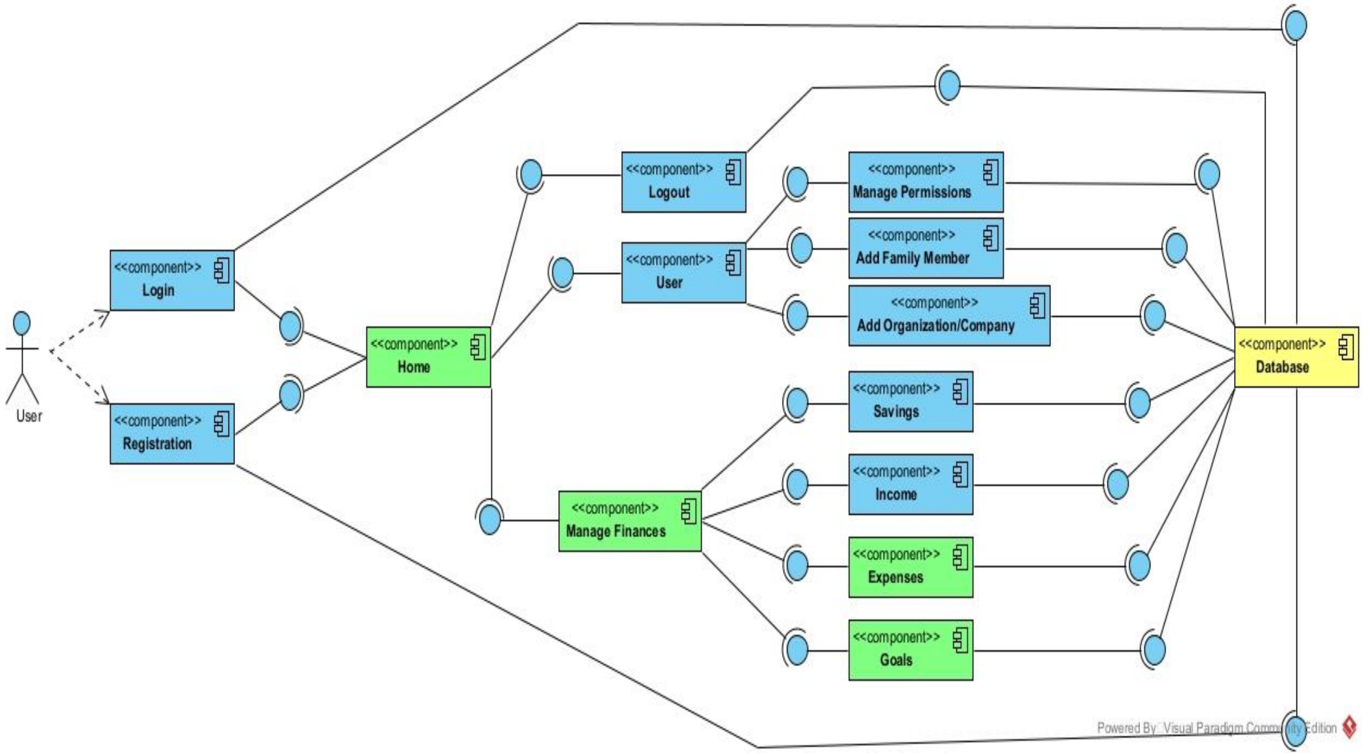
COMP246-408 Budget Planner



COMP246-408 Budget Planner



Section 4. High Level Component / Deployment Diagram



Section 5. Update the Gantt chart to include Part C Tasks(sara)