CP317- Software Engineering Project

Student Financial Planner

Group Members:	Usama Mohiuddin, Lorand Kis, Chetas Patel
Student ID:	212090340, 210629580, 200679130
Date:	2023-04-09
Semester:	Winter 2023
Professor:	Dr. Sumeet Sekhra
Course:	CP317 – Software Engineering

Table of Contents

Project Planning 3
Gantt Chart 3
Project Breakdown 4
Feasibility Analysis 5
Project Architecture 6
UML DIAGRAMS 8
Software Processes 11
Software Requirement Specification (SRS) Document 12
Project Testing 16
Project Output 19

Project Planning

The student financial will provide users with a platform to track their financial status, including their expenses and income. The website will include the following feature:

- User registration and login
- Dashboard displaying summary of financial status
    - Displays Income
    - Displays Expenses
- Expense tracking tool
- Income tracking tool
- Alerts and notifications for enlarged expenses
- Displays report for financial status

The project objective is to develop and launch a fully functional student financial tracker website withing a 12-week timeframe.

Gantt Chart

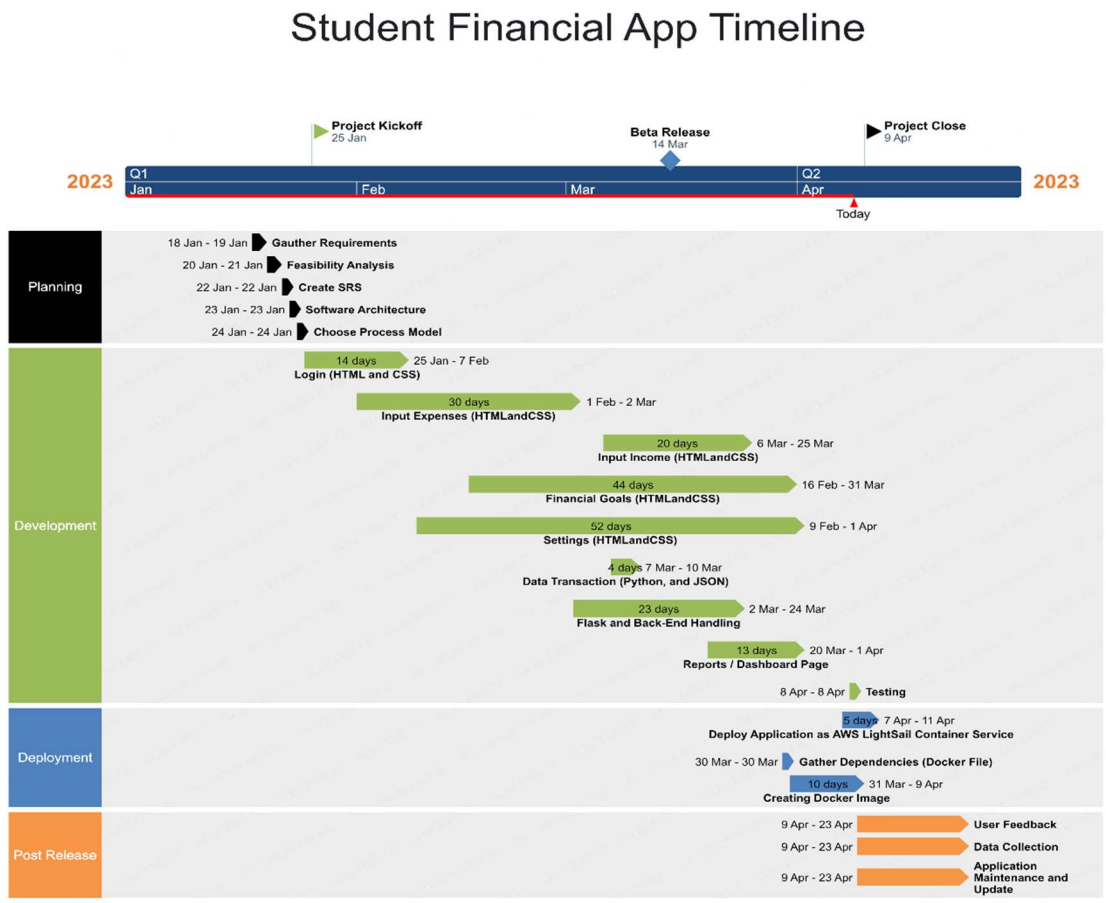

Figure 1 Gantt Chart

Project Breakdown

- **PROJECT PLANNING STAGE**
    1. Gather Requirements
    2. Feasibility Analysis
    3. Create SRS Document
    4. Software Architecture
    5. Choose Process Model
- **DEVELOPMENT STAGE**
    1. Login (HTML and CSS)
    2. Input Expenses (HTML and CSS)
    3. Input Income (HTML and CSS)
    4. Settings (HTML and CSS)
    5. Data Transaction (Python, and JSON)
    6. Flask and Back-End Handling
    7. Reports/Dashboard Page
    8. Testing
- **DEPLOYMENT STAGE**
    1. Deploy Application as WS LightSail Container Service
    2. Gather Dependences (Docker File)
    3. Creating Docker Image
- **POST RELEASE STAGE**
    1. User Feedback
    2. Data Collection
    3. Application Maintenance and Update

Feasibility Analysis

Introduction

In recent years, the number of students struggling with financial management has increased significantly. To address this issue, a Student Financial App has been proposed to help students manage their finances effectively. In this feasibility analysis, we will examine the technical, operational, economic, and schedule feasibility of the project.

Technical Feasibility

The technical feasibility of the Student Financial App is high, as there are no major technological barriers that would prevent the development of the app. The app will be built using widely available programming languages and frameworks such as Python, Flask, and HTML/CSS/JavaScript, which are easily accessible and scalable. The app will be hosted on a cloud-based server, which provides scalability, flexibility, and cost-effectiveness.

Operational Feasibility

The operational feasibility of the Student Financial App is also high, as the app will be user-friendly and easy to navigate. The app will have a simple interface, making it easy for students to input their expenses, income, and financial goals. The app will also have features such as notifications and reminders, making it easy for students to stay on top of their finances. The app will be tested with a small group of students before being rolled out to a larger audience to ensure that it meets their needs and expectations.

Economic Feasibility

The economic feasibility of the Student Financial App is moderate, as there will be initial development costs associated with the app. However, the app will be monetized by offering premium features to users, such as advanced analytics and financial planning tools. The app will also have the potential to attract sponsorships from financial institutions, which can provide a source of revenue. The app will have a freemium model, where basic features will be free, and users can upgrade to premium features for a fee.

Schedule Feasibility

The schedule feasibility of the Student Financial App is high, as the project can be completed within the proposed timeline. A Gantt chart has been created to break down the development process into manageable tasks and to monitor progress. The project team will be working on a full-time basis, which will ensure that the project is completed within the proposed timeline.

Conclusion

Overall, the Student Financial App is a feasible project that can be completed within the proposed timeline and budget. The app addresses a real need for students to manage their finances effectively, and it has the potential to generate revenue through sponsorships and premium features. With a user-

friendly interface and useful features such as notifications and reminders, the app is likely to be well-received by students. The next steps will be to develop the app and test it with a small group of users to ensure that it meets their needs and expectations.

Project Architecture

Introduction

The design of the architecture of a software application plays a crucial role in its development. The architecture determines the behavior of the software and its interaction with other systems. For this student financial app, we have chosen to use a Model-View-Controller (MVC) architecture with the Python Flask framework to manage the application's functionality. Additionally, we have chosen to deploy the application through a client-server architecture, using a containerized application generated from a Docker image. This document will provide an explanation of why these choices were made.

Application Using MVC Architecture

The MVC architecture separates the application into three main components: the model, the view, and the controller. This architecture is widely used because it helps in maintaining the code-base and making it easier to modify, test, and debug. Additionally, it provides better modularity by separating the application's user interface, data, and business logic.

In this student financial app, the model component is responsible for managing the data, such as expenses, income, and financial goals. The view component handles the user interface, including the HTML templates, CSS stylesheets, and JavaScript scripts. The controller component manages the application's logic, including processing user input and updating the model and view components.

Python Flask Framework

We have chosen the Python Flask framework to implement the MVC architecture of this application. Flask is a lightweight and modular framework that allows developers to easily implement web applications using the MVC architecture. It provides various tools for handling HTTP requests, routing, and templates. Flask's lightweight nature makes it easy to modify and scale the application, and its modular design allows developers to add only the required components.

Deployment Using Client-Server Architecture

In a client-server architecture, the application's components are distributed between the client-side and server-side. The client-side components run on the user's device, and the server-side components run on a remote server. This approach enables the client-side to only interact with the server-side components for retrieving and sending data.

We have chosen a client-server architecture for this student financial app because it provides a more scalable and maintainable solution. By separating the client-side and server-side components, we can easily scale the application as the user base grows. Additionally, the server-side components can be easily maintained and updated without having to update each individual client.

Docker Image

To deploy the application, we have chosen to use a Docker image. Docker allows developers to package an application with all its dependencies into a container that can run on any operating system. Using a Docker image for deployment simplifies the deployment process, ensures that the application runs consistently across different environments, and makes it easy to update and roll back the application.

Conclusion

In summary, we have chosen to use an MVC architecture with the Python Flask framework to implement this student financial app. This architecture provides better modularity and maintainability of the codebase. We have also chosen to use a client-server architecture to ensure the scalability and maintainability of the application. Finally, we have chosen to deploy the application using a Docker image to simplify the deployment process and ensure consistency across different environments.

UML DIAGRAMS

Model View Controller Diagram

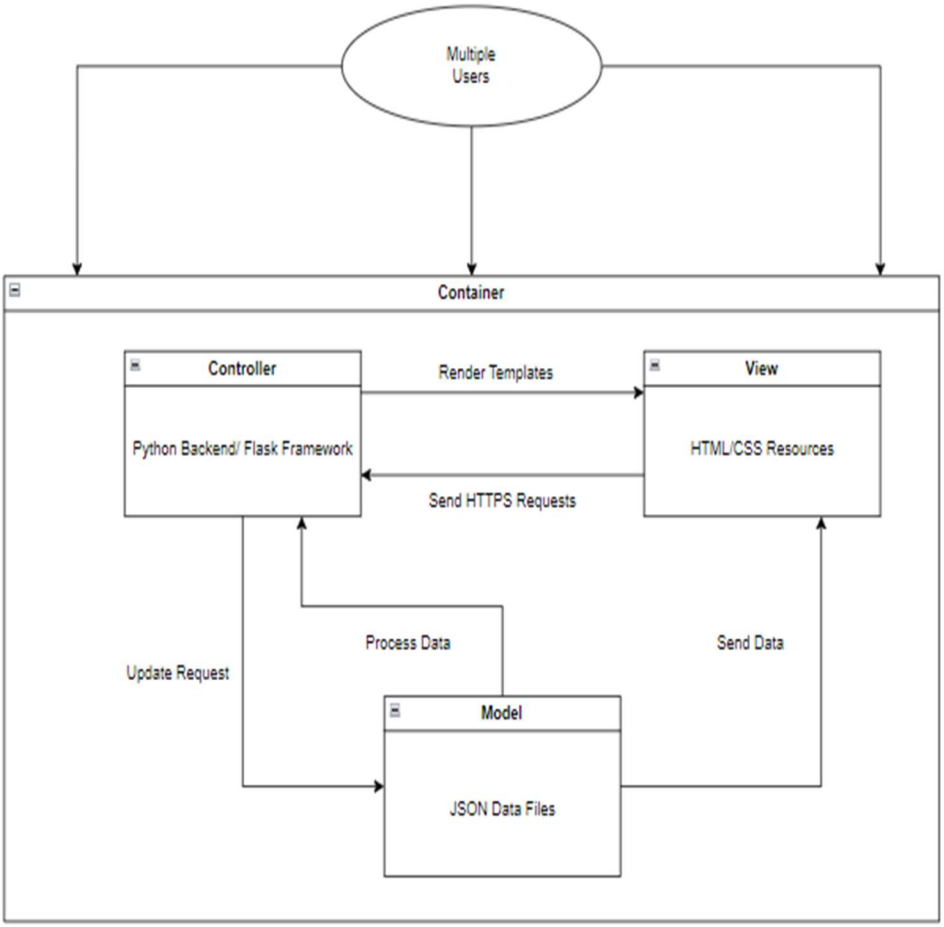

Figure 2 (MVC) Model- View - Controller Architecture Design

Use Case Diagram

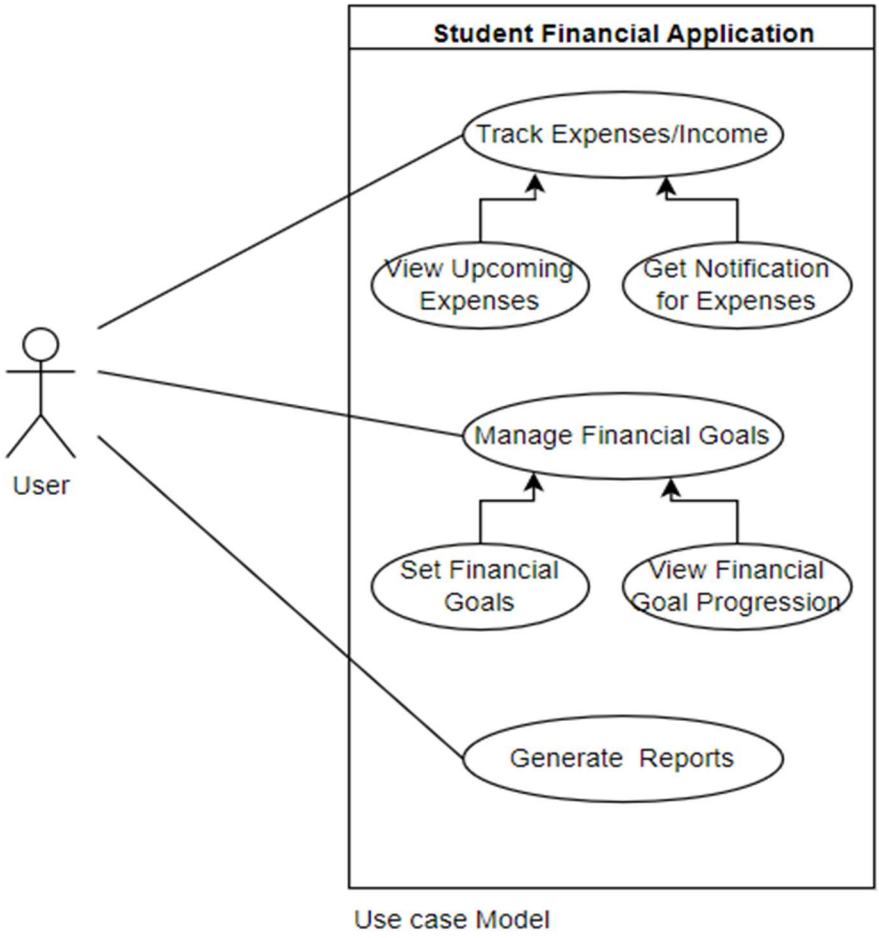

Figure 3 Use Case Diagram

Component Diagram

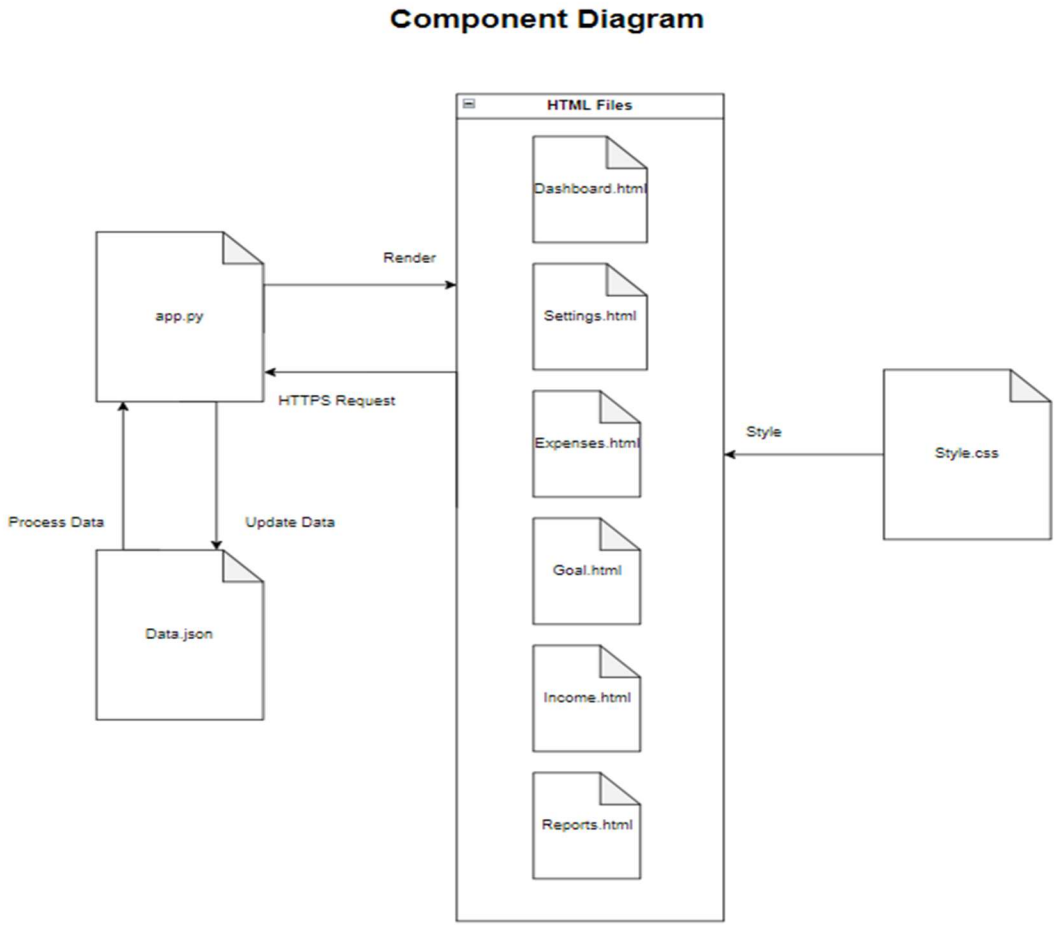

Figure 4 Component Diagram

Software Processes

We used the Scrum model for our student financial application. Scrum is an Agile framework that allows teams to work collaboratively, efficiently, and flexibly on complex projects. It's particularly useful for projects with evolving requirements, where change is inevitable, and requirements may change frequently. Here are some reasons why we have chosen the Scrum model for our student financial application:

1. **Flexibility:** Scrum is a flexible framework that allows us to adjust to changing requirements and project needs. It's ideal for projects where the requirements are not completely clear or may change over time.

2. **Incremental delivery:** Scrum emphasizes the delivery of working software in increments, providing early and continuous feedback to stakeholders. This allows us to quickly adjust and adapt the software to meet the changing needs of our users.

3. **Collaboration:** Scrum encourages collaboration and open communication between team members and stakeholders. This leads to a shared understanding of the project's goals and requirements, which is critical to the project's success.

4. **Continuous improvement:** Scrum encourages continuous improvement through retrospectives and feedback. This allows us to identify areas for improvement and make changes to the project as needed.

5. **Predictability:** Scrum provides a framework for estimating and tracking progress, allowing us to have a high degree of predictability and transparency throughout the project.

In summary, the Scrum model is ideal for our student financial application because it provides the flexibility, collaboration, and continuous improvement needed for a complex software project. It also allows us to deliver working software in increments, providing early and continuous feedback to stakeholders, which is essential for meeting the needs of our users.

Software Requirement Specification (SRS) Document

1. Introduction

The Student Financial Application is a web application designed to help students manage their finances effectively. This SRS document provides an overview of the requirements and features of the application.

1.1 Purpose

The purpose of this SRS document is to describe the functional and non-functional requirements of the Student Financial application.

1.2 Scope

The Student Financial application will allow students to track their expenses, create a personalized budget, set financial goals, and receive personalized recommendations to manage their finances effectively. The application will be user-friendly and customizable, allowing students to choose their preferred language, currency, and expense categories.

1.3 Definitions, Acronyms, and Abbreviations

SRS: Software Requirements Specification App: Application

2. Overall Description

The Student Financial application is a web app designed to help students manage their finances effectively. The application is user-friendly and customizable, allowing students to set financial goals, track their expenses, and create a personalized budget.

2.1 Product Perspective

The Student Financial application is a standalone web app that will be accessible from any device with an internet connection.

2.2 Product Features

The Student Financial application will have the following features:

2.2.1 Expense Tracking

The application will allow students to track their expenses by category, such as food, transportation, entertainment, and school supplies. Students can easily input their expenses and categorize them in the app and view their spending history by category and over time.

2.2.2 Budget Creation

The application will allow students to create a personalized budget based on their income and expenses. Students can set monthly spending limits for each category, and the app will alert them when they are close to reaching their limits.

2.2.3 Financial Goal

Setting The application will allow students to set financial goals, such as saving for a trip or paying off student loans. Students can track their progress towards their goals and receive personalized recommendations for how to reach them faster.

2.2.4 Expense Analysis

The application will provide students with detailed analysis of their expenses and help them identify areas where they can save money. For example, the app can show students how much they spend on eating out each month and provide suggestions for how to cut down on that expense.

2.2.5 Customization

The application will allow students to customize their experience by choosing their preferred currency, language, and categories for expenses.

2.3 User Characteristics

The Student Financial application is designed for students of all ages who want to manage their finances effectively.

2.4 Constraints

The Student Financial application will require an internet connection to function correctly.

3. Functional Requirements

The functional requirements of the Student Financial application are as follows:

3.1 Expense Tracking

The application should allow students to:

- Add expenses and categorize them by type (e.g., food, transportation, entertainment, school supplies)
- Edit, delete, and view their expenses by category and over time
- View their total expenses for a given period (e.g., month, week, year)

3.2 Budget Creation

The application should allow students to:

- Create a personalized budget based on their income and expenses
- Set monthly spending limits for each category
- Receive alerts when they are close to reaching their limits
- Adjust their budget based on their changing financial situation

3.3 Financial Goal Setting

The application should allow students to:

- Set financial goals, such as saving for a trip or paying off student loans
- Track their progress towards their goals
- Receive personalized recommendations for how to reach their goals faster

3.4 Expense Analysis (continued)

The application should allow students to:

- Analyze their expenses by category and over time
- Identify areas where they can save money by viewing expense trends and patterns
- Provide suggestions for how to cut down on specific expenses
- View expense reports and charts to track their financial progress

3.5 Customization

The application should allow students to:

- Choose their preferred language and currency
- Customize their expense categories to match their personal spending habits
- Set up notification preferences for alerts and reminders

4. Non-functional Requirements The non-functional requirements of the Student Financial application are as follows:

4.1 Performance

The application should be fast and responsive, with minimal loading times and no significant delays when adding or viewing expenses.

4.2 Usability

The application should be user-friendly and easy to navigate, with intuitive interfaces and clear instructions.

4.3 Security

The application should protect users' personal and financial information by using secure login procedures and encryption protocols.

4.4 Compatibility

The application should be compatible with different web browsers and operating systems to ensure access from a variety of devices.

5. Design Constraints

The design of the Student Financial application should be simple and intuitive, with clear visual representations of data and a consistent layout throughout the app.

6. Future Enhancements

The following features could be added in future versions of the Student Financial application:

- Integration with bank accounts and credit cards for automatic expense tracking
- Machine learning algorithms to provide personalized financial advice and suggestions
- Social features to enable students to share financial tips and advice with their peers
- Integration with external resources such as financial blogs, news, and educational content.

7. Conclusion

The Student Financial application is designed to help students manage their finances effectively by providing easy-to-use tools for expense tracking, budget creation, financial goal setting, and expense analysis. With its user-friendly and customizable features, the application aims to help students achieve financial independence and stability.

Project Testing

Inputs, Outputs, Test Cases

1. Test Suite for Login Functionality

Input:

- Valid email and password
- Invalid email or password

Output:

- Successful login and redirection to dashboard.
- Error message indicating invalid credentials.

Test Cases:

- Enter valid email and password, click login button - expected output: successful login and redirection to dashboard.
- Enter invalid email, valid password, click login button - expected output: error message indicating invalid credentials.
- Enter valid email, invalid password, click login button - expected output: error message indicating invalid credentials.
- Leave email and password fields blank, click login button - expected output: error message indicating invalid credentials.

2. Test Suite for Adding an Expense

Input:

- Expense name
- Expense category (Optional)
- Expense amount
- Date of expense
- Recurring (Optional)

Output:

- Successful addition of expense and display on expenses page.
- Error message indicating invalid input or failure to add expense.

Test Cases:

- Enter valid expense name, expense amount, and date - expected output: successful addition of expense and display on expenses page
- Enter invalid expense name (e.g., blank), valid expense amount, and valid date - expected output: error message indicating invalid input or failure to add expense

- Enter valid expense name, invalid expense amount (e.g., non-numeric), and valid date - expected output: error message indicating invalid input or failure to add expense

3. **Enter valid expense name, valid expense amount, and invalid date (e.g. in the future) - expected output: error message indicating invalid input or failure to add expense**
4. **Test Suite for Adding Income**

Input:

- Income name
- Income amount
- Date of income
- Recurring (N/A, Weekly, Bi-Weekly, Monthly, Annually)

Output:

- Successful addition of income and display on income page.
- Error message indicating invalid input or failure to add income.

Test Cases:

- Enter valid income name, income amount, and date - expected output: successful addition of income and display on income page.
- Enter invalid income name (e.g., blank), valid income amount, and valid date - expected output: error message indicating invalid input or failure to add income.
- Enter valid income name, invalid income amount (e.g., non-numeric), and valid date - expected output: error message indicating invalid input or failure to add income.
- Enter valid income name, valid income amount, invalid date (e.g., in the future), and recurring set to N/A - expected output: error message indicating invalid input or failure to add income.

5. **Test Suite for Setting Financial Goals**

Input:

- Goal name
- Goal amount
- Target date

Output:

- Successful addition of goal and display on goals page.
- Error message indicating invalid input or failure to add goal.

Test Cases:

- Enter valid goal name, goal amount, and target date - expected output: successful addition of goal and display on goals page.
- Enter invalid goal name (e.g., blank), valid goal amount, and valid target date - expected output: error message indicating invalid input or failure to add goal.
- Enter valid goal name, invalid goal amount (e.g., non-numeric), and valid target date - expected output: error message indicating invalid input or failure to add goal.

- Enter valid goal name, valid goal amount, invalid target date (e.g., in the past) - expected output: error message indicating invalid input or failure to add goal.

6. Test Suite for Generating Reports

Input:

- Date range for report (Start Date and End date)

Output:

- Successful report generated and display on report page.
- Error message indicating invalid input or failure to generate report.

Test Cases:

- Enter valid start date and end date - expected output: successful report generated and displayed to the page.

Enter invalid start date and/or end date (e.g., blank, or end date is before start date) - expected output: error message indicating invalid input or failure to generate report

Project Output

The following will be produced as part of the Student Financial application website project:

DASHBOARD PAGE

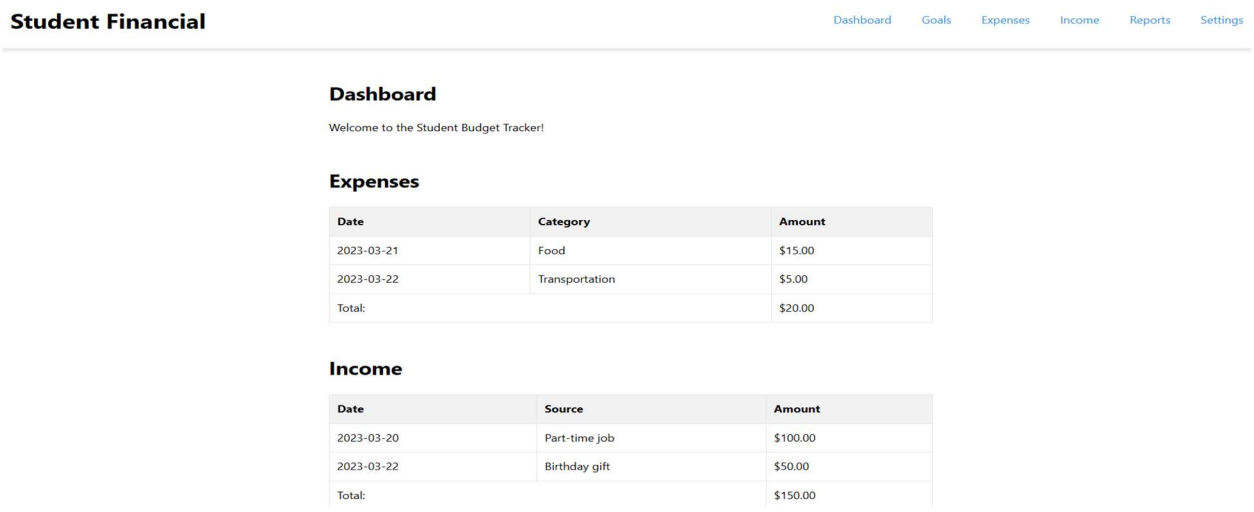

Figure 5 Dashboard

DESCRIPTION: this webpage is where the students can have a general overview of the changes, they have made on the website. In this website they can see a general overview of the changes they made in their income and financial Expense.

GOALS PAGE

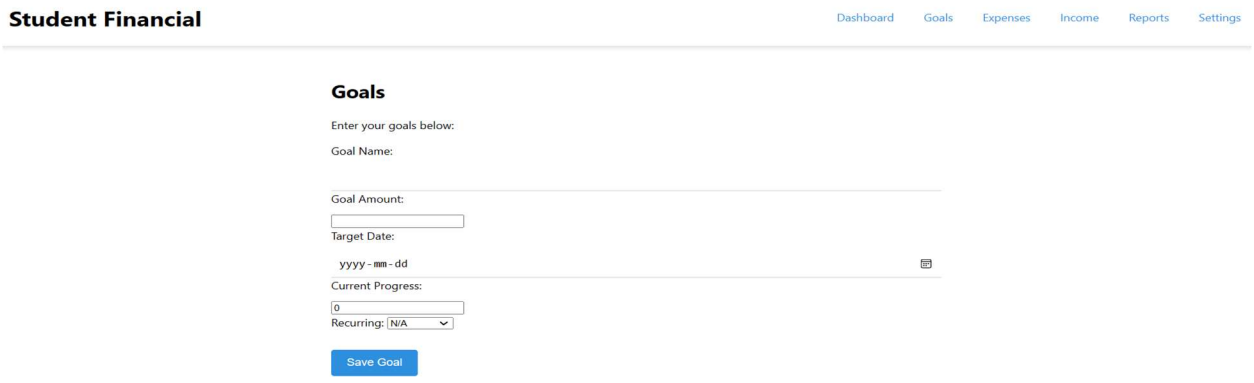

Figure 6 Goals

DESCRIPTION: In the goals webpage, students can set future goal amounts and assign target goal dates. This webpage also tracks the current progress so students can see how much they have progressed in their financial goals.

"The Rest of the Webpages are Similar in perspective, they are tools for students that they can use for their financial assistance in maintaining incomes and expenses."

EXPENSES PAGE

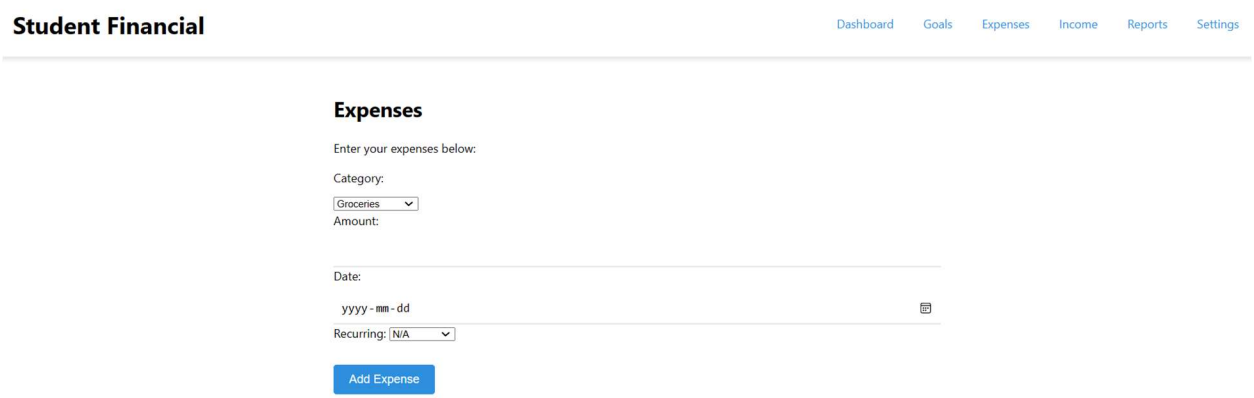

Figure 7 Expenses

INCOME PAGE

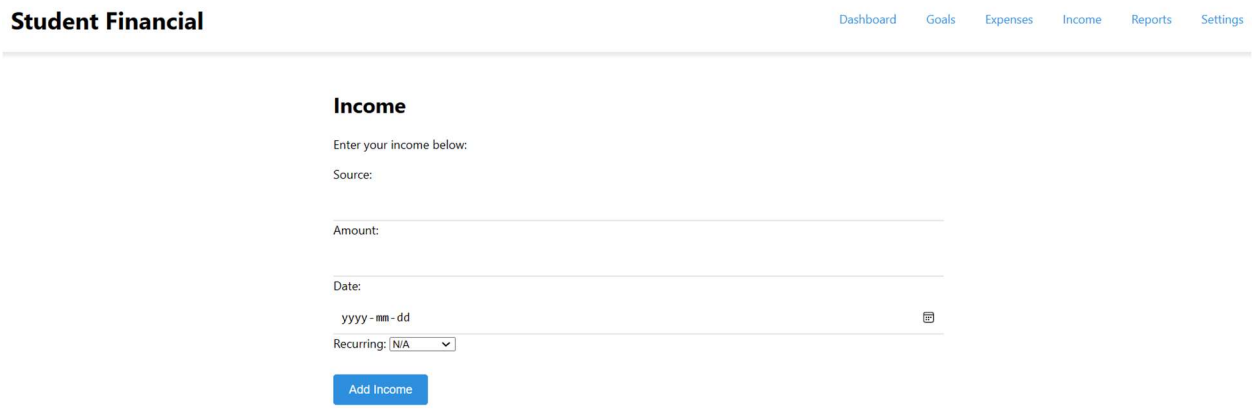

Figure 8 Income

REPORTS PAGE

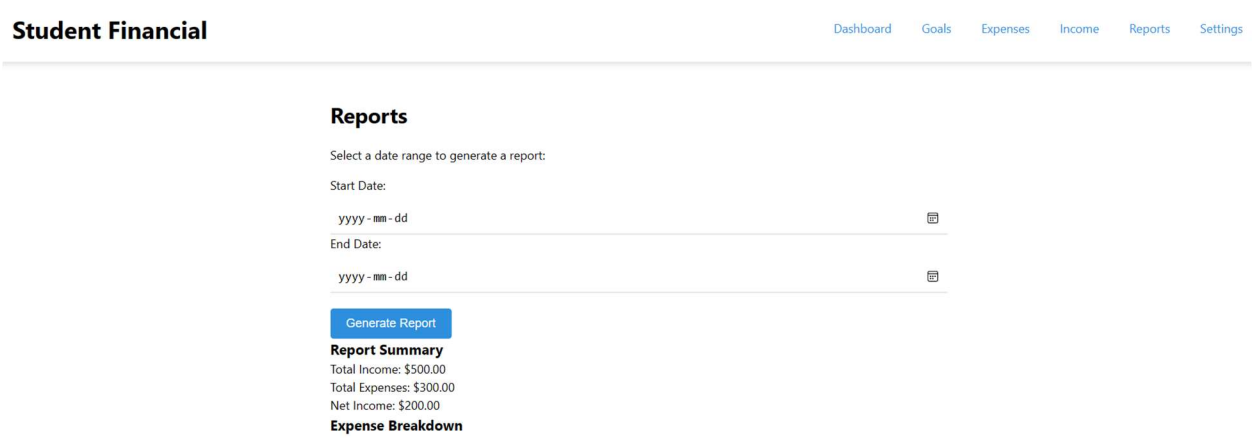

Figure 9 Reports

SETTINGS PAGE

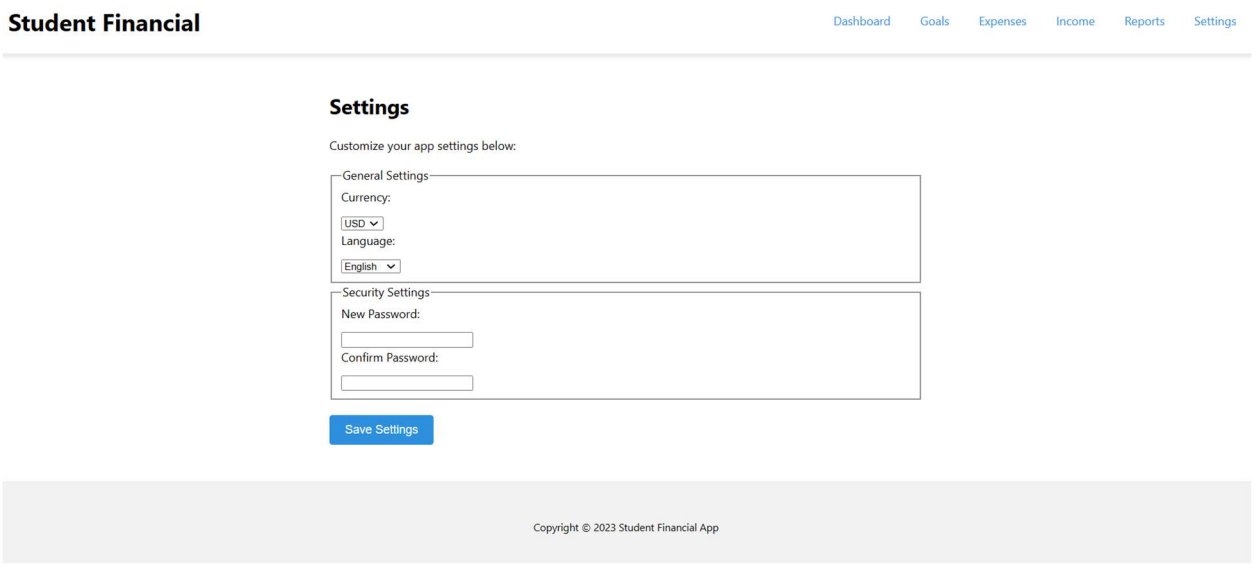

Figure 10 Settings