Project Report

Personal Expense Tracker

Created by Senihers-group 5

Bhavya Choudhary
Riya Jha
Yutika Singh
Shristi Tiwari
Mahi Agarwal

Under the mentorship of

Karuna Kukreja
Purvansh Shah

Introduction

This project aims to develop a user-friendly application for expense tracking and financial management. The application will streamline the process for users to record their expenses, track spending patterns over time, visualize data through charts and graphs, identify cost-cutting opportunities, and improve financial decision-making. To enhance user engagement, the application will gamify the experience by tracking user activity, maintaining a streak for continuous expense tracking, and sending notifications via email regarding expenses, incomes, and current balances.

Goals and Objectives

1. **Efficient Expense and Income Tracking:** Enable users to easily record and track their expenses and income, providing a clear overview of their financial transactions.
2. **Data Visualization for Analysis:** Implement visualizations such as charts and graphs to help users understand their spending patterns, identify trends, and make informed financial decisions.
3. **User-Friendly Income Management:** Offer a user-friendly interface for managing income sources, including the ability to add, view, and delete income records conveniently.
4. **Streamlined Expense Management:** Provide efficient tools for managing expenses, including adding new expenses, deleting or updating existing records, and maintaining a streak for continuous tracking.
5. **Personalized Financial Suggestions:** Offer personalized suggestions and recommendations based on user data, helping users allocate funds effectively and optimize their financial management strategies.
6. **Email Notifications for Updates:** Implement email notifications to keep users informed about their financial status, including balance updates, expense alerts, and important financial insights.

Flowchart

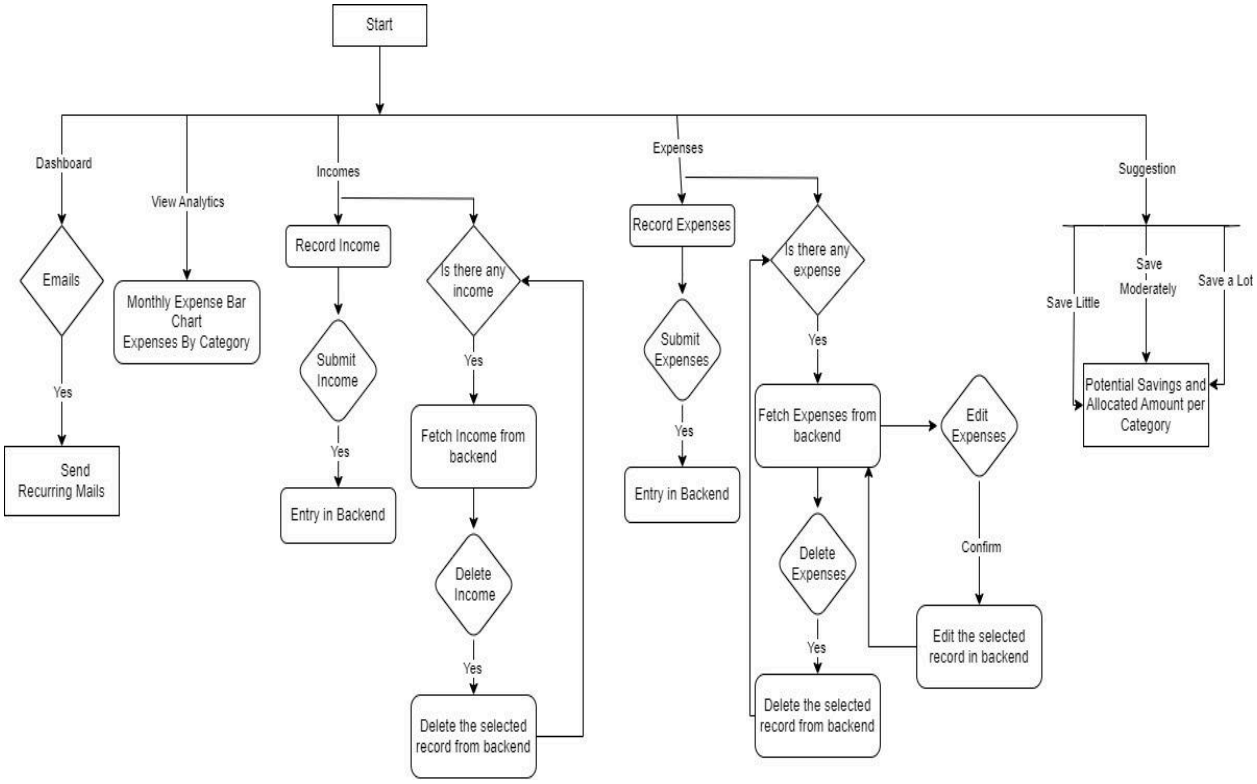


Fig1: Flowchart

ER Diagram

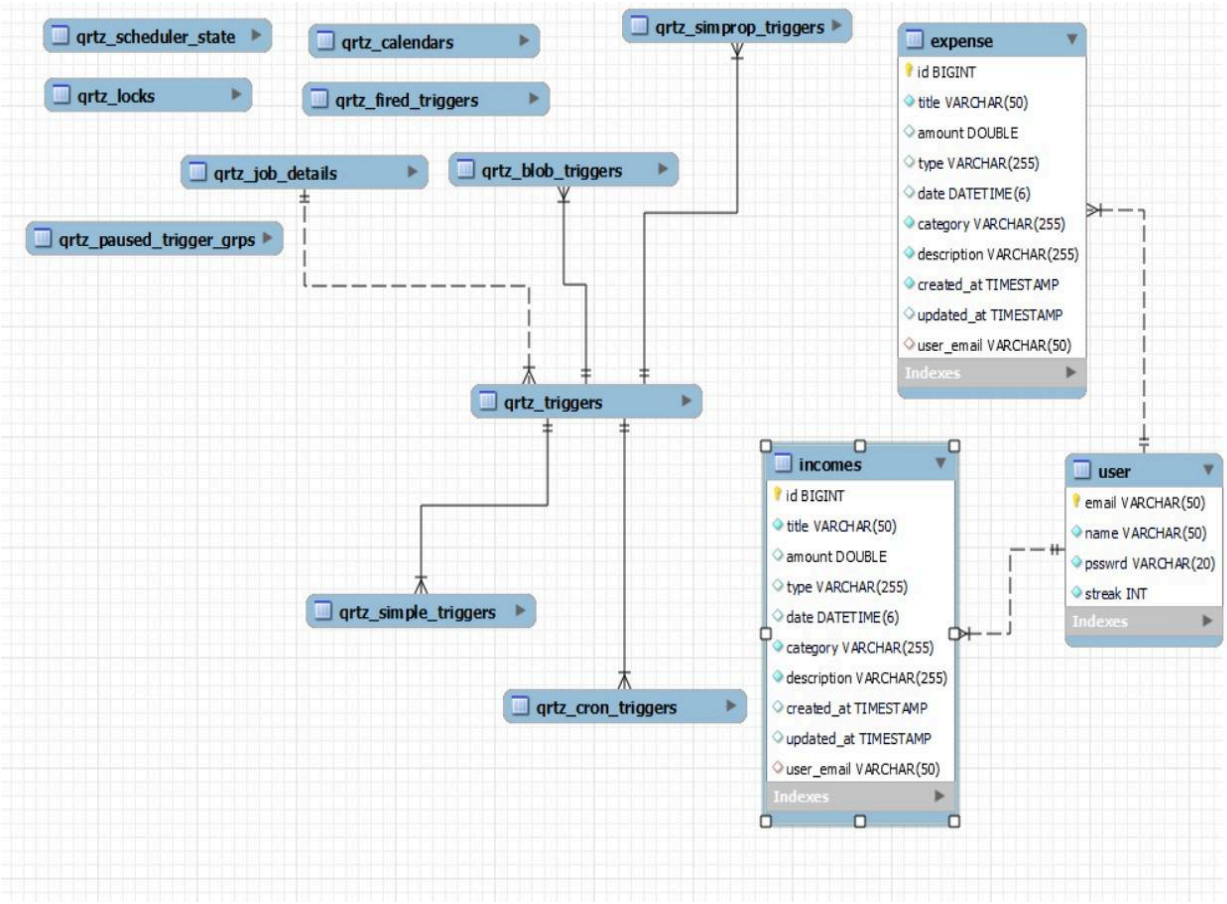


Fig2: ER Diagram

Tech stacks used

1. Frontend:
    - React.js
    - Styled Component
    - Chart.js
2. Backend:
    - Java
    - Spring Boot
    - Quartz - Job Scheduler
    - Spring Mail
3. Database
    - SQL

Features of the project

1. **Record Income and Expenses**
    - **Adding Incomes and Expenses:** This section of the income page presents users with a user-friendly form for conveniently adding new income and expense details efficiently and accurately.
    - **Income and Expense History:** On this section of the page, users can access their income and expense history, neatly sorted by date for easy reference. Additionally, the page offers a delete feature, empowering users to manage and delete their transaction records. An additional feature of the update is available on the expense page.
    - **Streak Maintenance:** We have implemented a streak feature to the expense page to track the number of continuous days a user has recorded his/her expenses. This functionality is managed through three APIs. The process works as follows:
        - **Storing Current Streak:** The application maintains a record of the user's current streak, which represents the number of consecutive days with expenses.
        - **Storing Last Added Date:** Additionally, the application stores the date when the user added the last expense.
        - **Streak Maintenance Logic:** When a user adds a new expense, the application checks the current date and compares it with the last added date of an expense. If the difference between the current date and the last

added date is 1 day, the streak is incremented by 1, else the streak is reset to 0. This updated streak value is then passed to the send_streak_to_backend API to update the streak on the backend.

```
// Check if the addition is continuous
const currentDate = new Date();
const timeDifference = currentDate - lastAddedDate;
const daysDifference = timeDifference / (1000 * 60 * 60 * 24);
const isContinuous = daysDifference === 1;

// Reset the streak if the addition is not continuous
if (!isContinuous) {
    await sendStreakToBackend(0); // Reset streak
} else {
    await sendStreakToBackend(streak + 1); // Increment streak
}
```

- **API Integration:** These streak maintenance operations are facilitated through three APIs: one for updating and storing the current streak, another for updating and storing the last added date, and a third API called send_streak_to_backend for updating the streak on the backend.

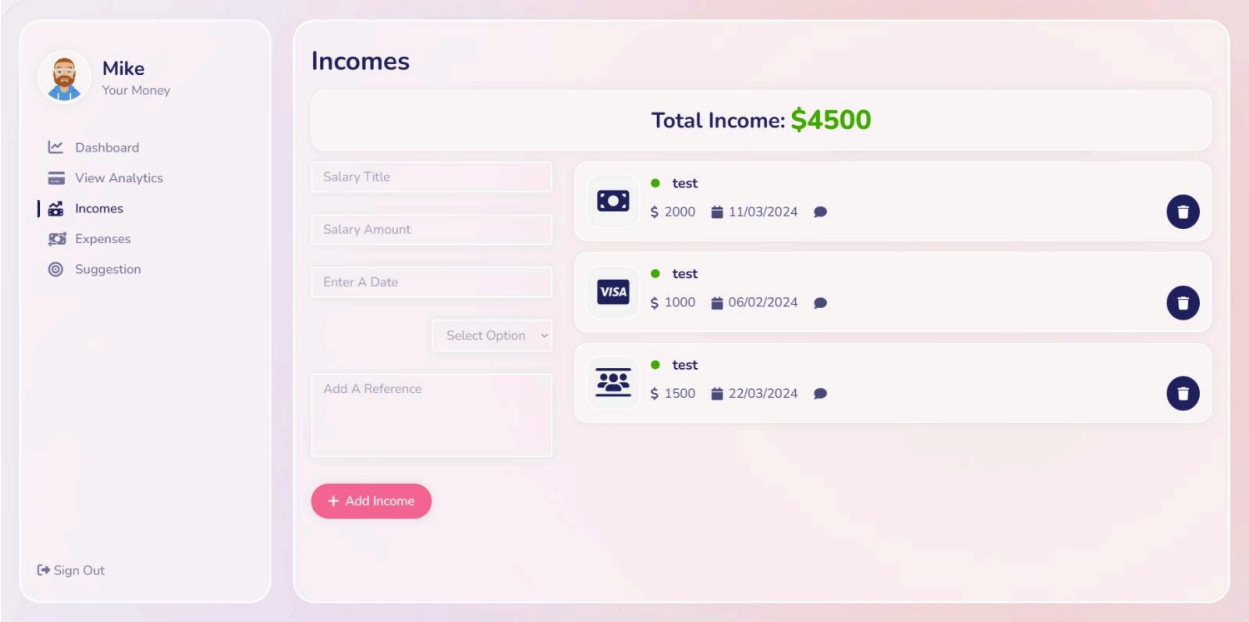


Fig3: Incomes Page

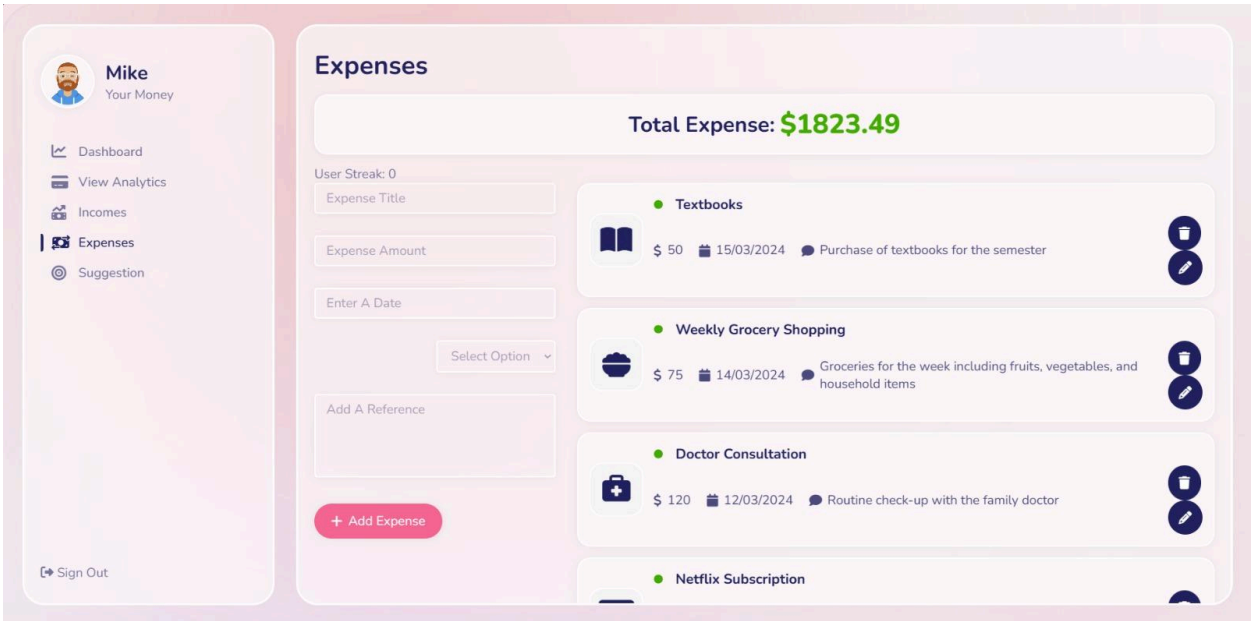


Fig4: Expenses Page

2. **Get a visual representation of your expenses**
    - **Monthly Expenses Bar Chart:** Powered by ReactJS and Chart.js, this feature generates an interactive bar chart on the analytics page, dynamically illustrating users' monthly expenses. Data fetched from an SQL database via Spring Boot enables the chart to provide a comprehensive overview of spending habits over time. With each bar representing a month, users can easily track fluctuations and patterns in their expenditures, empowering them to make informed financial decisions and adjust their budget as needed.

Fig5: View Analytics Page Monthly Expenses Bar Chart

- **Expenses by Category Pie Chart:** Leveraging the capabilities of ReactJS and Chart.js, this feature presents users with a visually informative pie chart showcasing their monthly expenses categorized by type. By fetching data from a SQL database via Spring Boot, the chart offers a clear understanding of how spending is distributed across different categories. This insight enables users to conduct detailed financial analysis, identify areas of overspending or underutilization, and plan accordingly to optimize their budget and achieve their financial goals effectively.

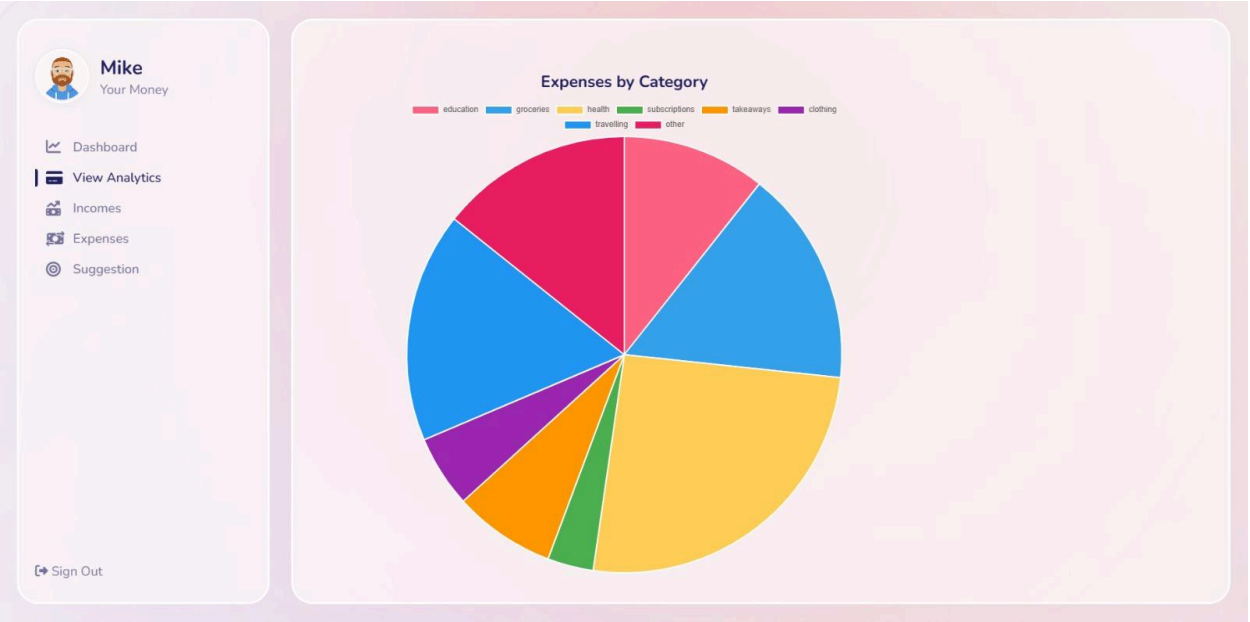


Fig6: View Analytics Page Expenses By Category

3. Get monthly updates on Expense, Income and Balance via mail

Users can effortlessly monitor their financial status by checking the checkbox on the dashboard, wherein they can opt to receive monthly financial updates on their mail about their spending habits which includes their expenses, income, and balance. This feature is implemented using an open-source Java library Quartz Job Scheduler. This feature includes the following major parts :

- **EmailJobSchedulerController:** This is the Primary API (Rest API) for scheduling email jobs.
- **Job:** This is the inference to be implemented by classes that represent a job in Quartz. It has a single method execute() where we write the work that needs to be performed by the job.
- **JobDetail:** This represents an instance of a job containing additional data in the form of a JobDataMap that is passed to the job when it is executed. Every EmailJob is identified by a JobKey that consists of a name and a group (name is unique within a group).
- **JobBuilder:** It is a fluent builder-style API to construct JobDetail instances.
- **Trigger:** This defines the schedule at which a given job will be executed.
- **TriggerBuilder:** It is used to instantiate Triggers in the application.

- **Frontend Part:**

Checkbox checked by the user to receive monthly emails

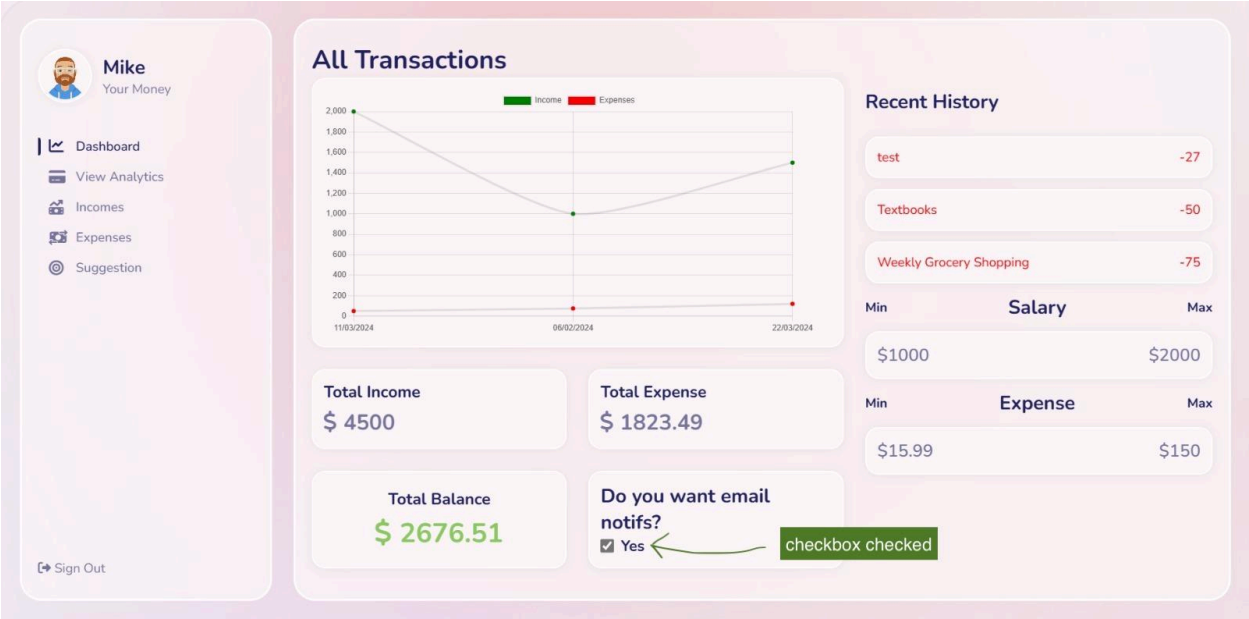


Fig7: Checkbox checked on the dashboard

Recurring Emails received by the user (every second :*for testing*) on his/her email id

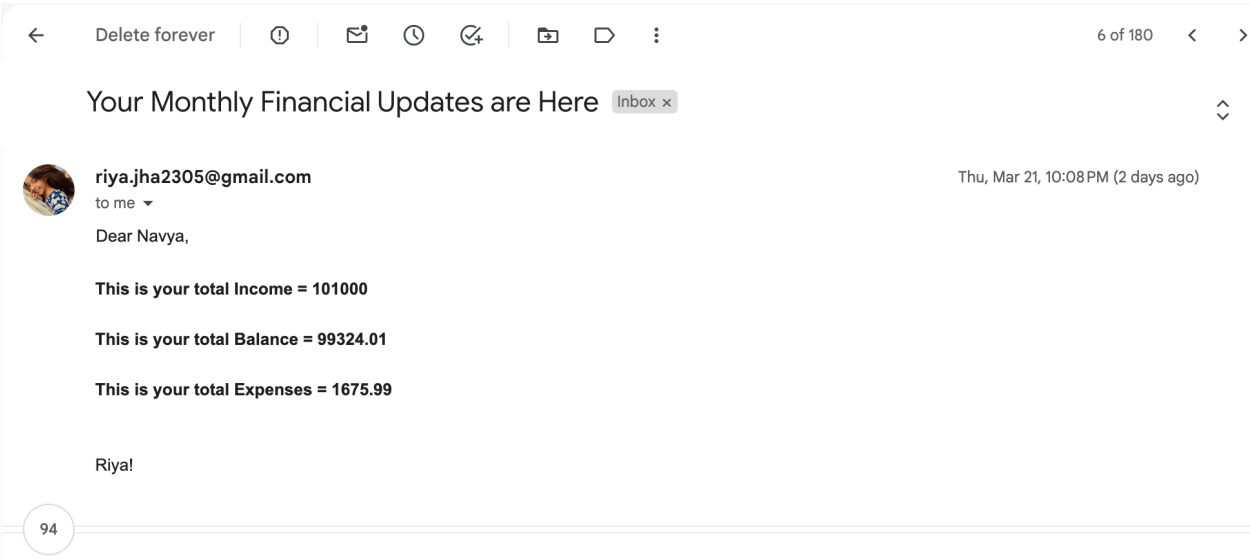
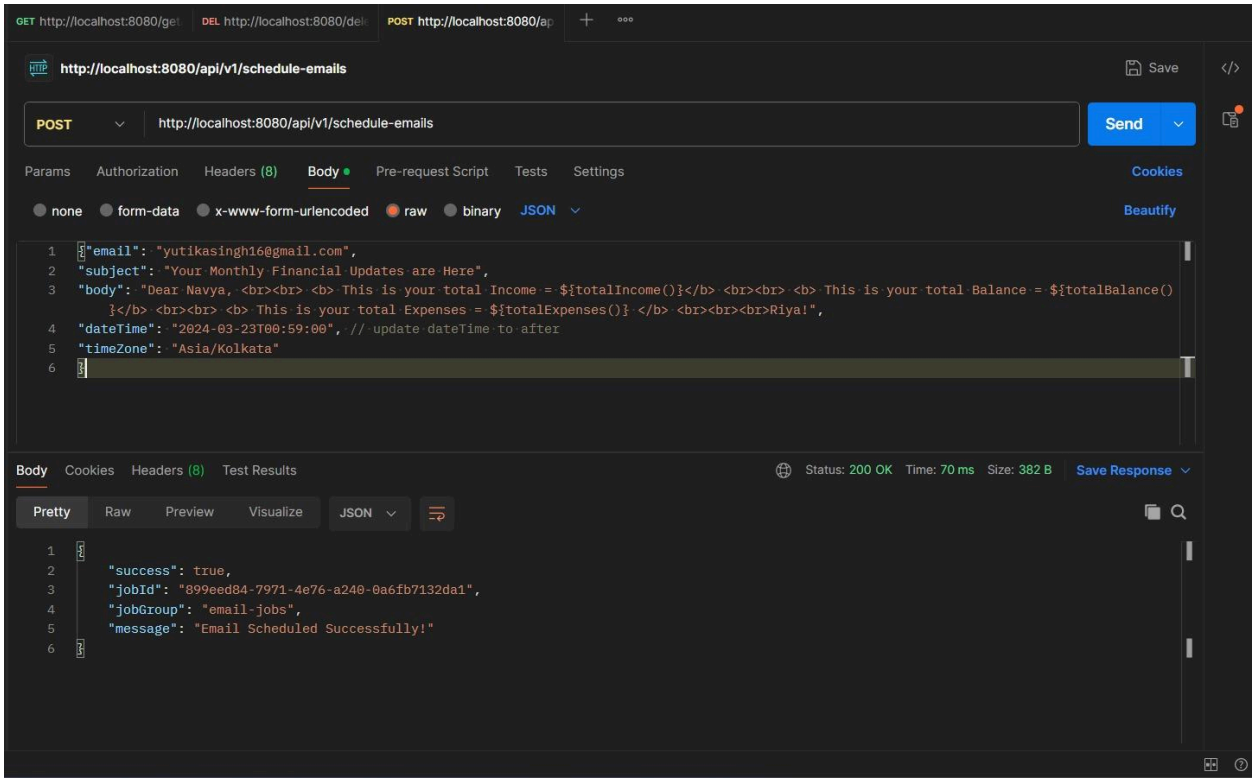


Fig8: Email received by the user

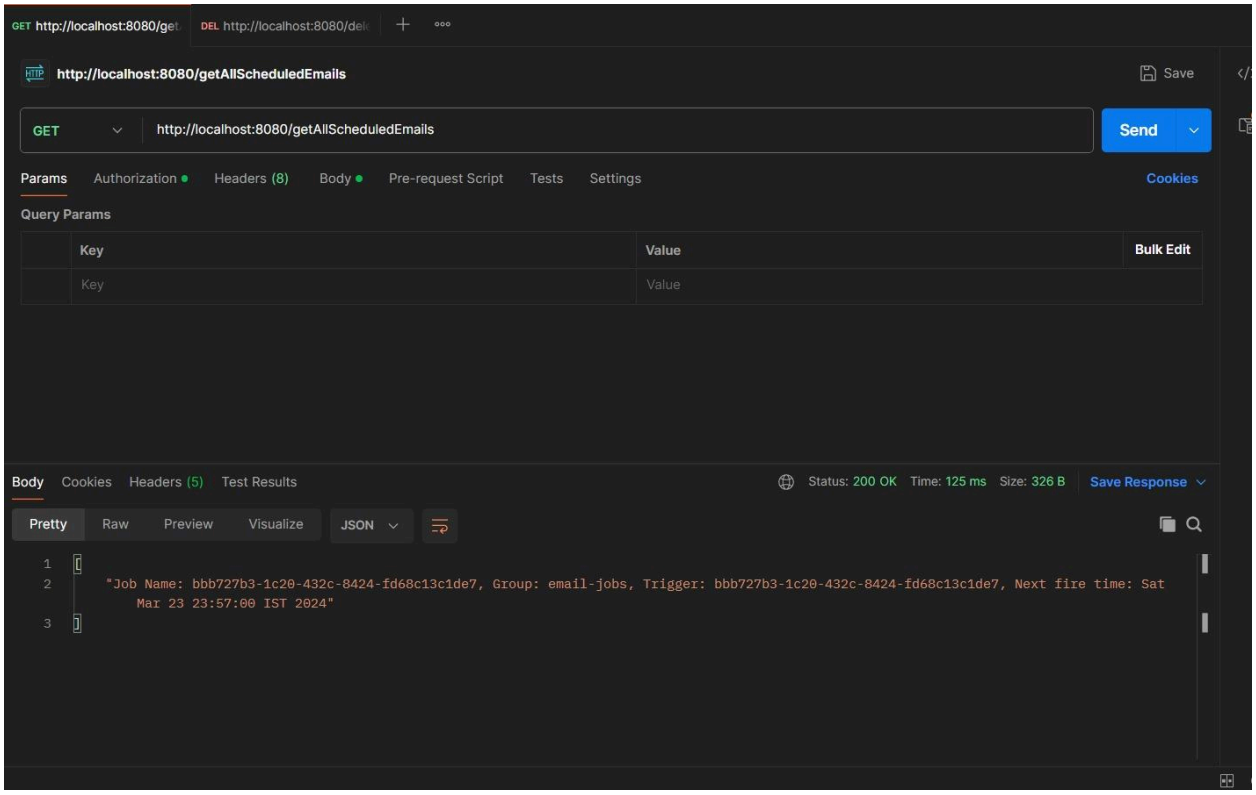
- **Backend Part:**

The backend working of the feature is displayed via postman results

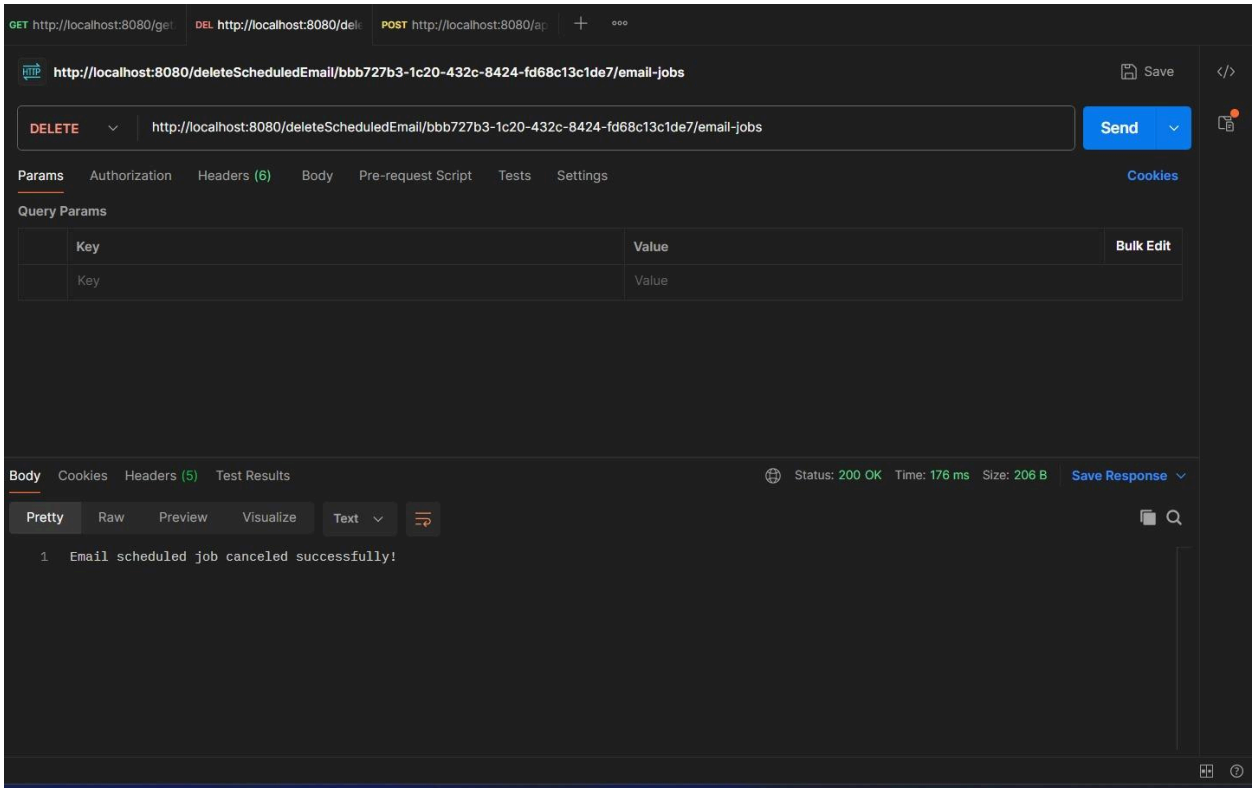
POST: http://localhost:8080/api/v1/schedule-emails



GET: http://localhost:8080/getAllScheduleEmails



DELETE: http://localhost:8080/deleteScheduledEmail/{jobName}/{jobGroup}



4. Spending Allocation Suggestions

The suggestion page helps users make smart decisions by giving them three choices for how much to save: less, moderate, or high, with moderate set as the default choice. Users can see a pie chart that shows them how much they should spend in different categories. Alongside, potential savings suggestions are provided for each category.

- **Backend Service:** Utilizing Java, the backend service processes user expenses and generates estimates based on the selected allocation level.
- **Algorithmic Approach:**
  - **Mapping of categories with their priority levels:** The *ExpenseService* prioritizes spending categories based on importance.

    Map<String, Integer> categoryPriorities maps:
    - essential categories like education, groceries and health with priority 1
    - less essential categories like traveling and clothing with priority 2

- the least essential categories like subscriptions and takeaways with priority 3

```
Map<String, Integer> categoryPriorities = new HashMap<>();
categoryPriorities.put(key:"education", value:1);
categoryPriorities.put(key:"groceries", value:1);
categoryPriorities.put(key:"health", value:1);
categoryPriorities.put(key:"travelling", value:2);
categoryPriorities.put(key:"clothing", value:2);
categoryPriorities.put(key:"subscriptions", value:3);
categoryPriorities.put(key:"takeaways", value:3);
categoryPriorities.put(key:"other", value:3);
```

- **User provided with saving level options:** Users are given 3 saving levels options:
  - Save a Little
  - Save Moderately
  - Saves a Lot

  To adjust expenses, the service calculates modified amounts for each expense based on its category's priority and the user's chosen saving levels. It retrieves each expense's priority from the *categoryPriorities* map and calculates the adjustment percentage.

- **Calculation of adjustment percentage (%):** The adjustment percentage depends on the category priority and the chosen spending level.
  - For priority 1 categories, the *adjustment % = 0%*
  - For priority 2 categories, the *adjustment % = spending level * 5%*
  - For priority 3 categories, the *adjustment % = spending level * 10%*

- **Calculation of Final Modified Expense the user should spend:** *ModifiedExpense = Amount * (1 - adjustment %)*

- **Visualization:** The frontend interface showcases the allocation suggestions through an intuitive pie chart, providing users with a clear visualization of their recommended spending allocations and potential savings in each category.

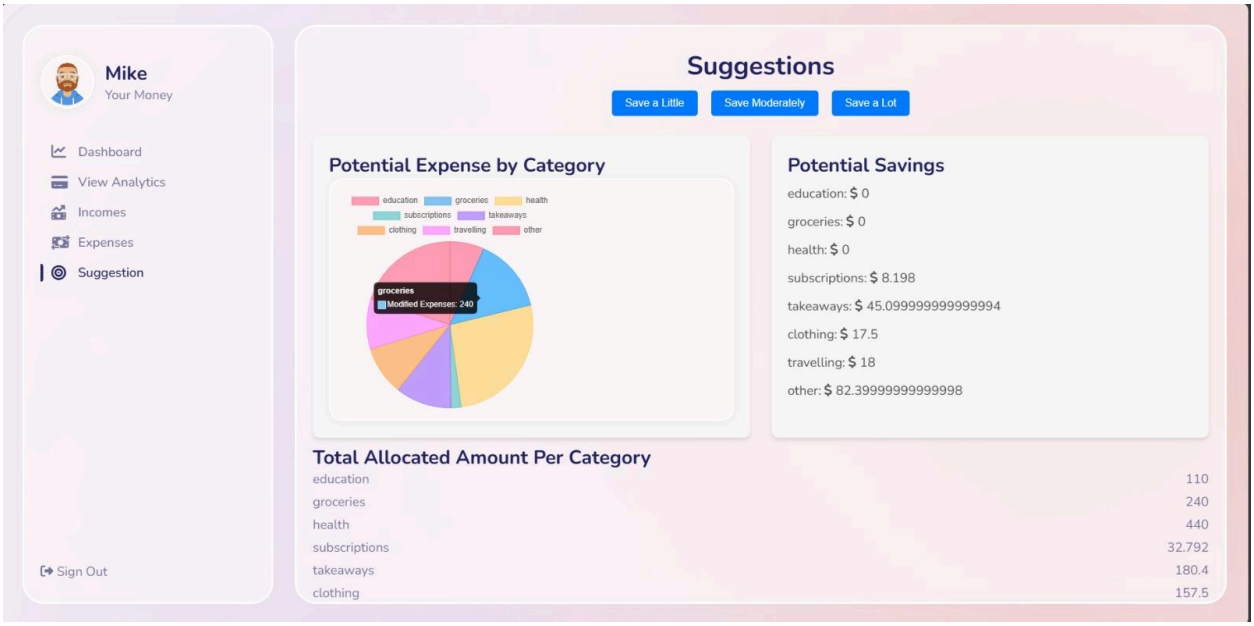


Fig9: Suggestions Page

Challenges we faced

1. Integrating React.js frontend with Spring Boot backend faced restrictions, hindering data exchange. To resolve, we implemented CORS handling in context.js, enabling cross-origin requests by configuring allowed origins, methods, and headers.
2. We had problems transferring mappings from front-end to backend so we resolved it by changing the intended port to the port on which the backend was working.
3. We had trouble running the database code on LINUX since SQL is case sensitive in LINUX.
4. We had issues with the latest version (3.2.4) of springboot-starter-parent which was not supported by the Quartz - Job Scheduler, so we downgraded the version to 2.5.6.
5. We initially faced compatibility issues with Quartz scheduler when migrating to Jakarta. This may be due to the fact that Quartz hasn't fully adapted to the namespace changes. So, instead we used javax to resolve this issue.

Desis Course learnings utilized in this project

Technical Skills

1. Object Oriented Programming
2. Data Structures & Algorithms
3. Problem Solving & Competitive Coding Skills
4. Database Management System (elective)
5. Java Programming Language (elective)

Soft Skills

1. Collaboration & Team Building
2. Communication Skills

What we learned from this project

Through this project, we learned the following skills:

1. Integration of Java SpringBoot with Node and React.
2. Connecting frontend and backend using API calls.
3. Using MySQL to store user expenses and integrating it with Java SpringBoot.
4. Graphical representation of data by fetching it from MySQL database.
5. Implementation of Email Job Scheduling using Quartz open source library.

Future scope

In the future, we would like to improve certain features in our project which we couldn't implement due to lack of time or knowledge.

1. Implementation of AI model to directly update expenses by scanning any bill using a camera or pdf scanning.
2. Authentication and maintenance of user streak to encourage users to regularly update the expenses.
3. Implementation of AI model to provide more customized suggestions regarding savings and investments.

References

1. Frontend - link1 (Basics of React.js), link2 (Implementation of Charts.js)
2. Backend - link (Spring Boot Setup), link (Basic CRUD in Java), link1, link2 (Quartz-job scheduler)
3. Database - link1 (Integration of MySQL workbench with the project), link2 (Basics of MySQL)
4. Stackoverflow and Chatgpt for error-resolving

Acknowledgements

We would like to express our heartfelt gratitude to the DESIS Ascend Educare 2023 team for giving us the wonderful opportunity to put the knowledge and skills that we have gained from the program to test through this project. We are highly thankful to our mentors Ms.Karuna Kukreja and Mr.Purvansh Shah for taking out time from their busy schedules and guiding us at every step.