| Expense Tracker and Analyzer Dongyang Su (19808)  Ekta Awasthi (19983)  Ikedinachim Ugochukwu ()  Tashi Garg (19928) |
| --- |

Table of Contents:

[**Expense Tracker and Analyzer 1**](#_50cd1em33khd)

[Project Highlights 3](#_doe0balootki)

[Project Folder Structure 4](#_vtuhmyyilaba)

[Project Web Application Initialization 5](#_qcumw5c9di17)

[Frontend and GUI Tech Stack 6](#_yjkyk6vjub3r)

[Tech Stack 6](#_cnstrk5eewd8)

[Task Breakdown 6](#_wgag75bkx82c)

[UI/UX Design 6](#_vofbz9llvlpt)

[Frontend Development 7](#_1ithxks7061x)

[Handling Secure User Authentication 7](#_tvverrkea27k)

[1. User Registration (Sign Up) 7](#_jqx47ytmt9fu)

[2. Login/Logout 8](#_baspkbbg55aa)

[3. Update Profile 9](#_fq6ivmg0vm8b)

[Handling Session Management 10](#_9eiky1j4snnq)

[1. Session Initialization 10](#_7pdwhz7f743v)

[2. Session Validation 10](#_ga9crm33y034)

[3. Session Logout 10](#_xghnuytgwyvc)

[4. Flash Messages 10](#_jto6bpn9m2yn)

[5. Session Expiry 11](#_geoo1ix9ot6g)

[GUI Workflows and API endpoints 11](#_8600p7xq0po)

[Add Expense 11](#_pgh4wnyxw1d5)

[View Expenses 12](#_ogwgy9cq7dkw)

[Path: /view\_expenses 12](#_124tcfmjx08v)

[Monthly Expense Summary 13](#_y5d9f71iz5zt)

[View Income 13](#_tul0sv9k5tl9)

[Add Income 14](#_fh3yqh97e0v)

[Monthly Income Breakdown 15](#_nlznvlbs7aek)

[Machine learning Model 16](#_a7iyq11d3bx4)

[Predict Expenses 16](#_r8ms5m49qi09)

[Recommend Savings 17](#_d1isptmrsywv)

[Database and Storage 19](#_pzgfnuaq93el)

[Data Operations 19](#_w2o8cjkwe2m5)

[Screenshots 20](#_u9h7meywalu3)

[ER Diagram 21](#_allzwmmft28u)

[Process 22](#_uxjyqakbike1)

[Why FastAPI for our Backend Development? 23](#_vigaa0zdzqvd)

[FastAPI Requirements 23](#_22vxeo72wx2i)

[FastAPI Installation 23](#_n6gbzg4ux7ju)

## Project Highlights

| **Project Name** | Expense Tracker and Analyzer |
| --- | --- |
| **Summary** | Expense Tracker and Analyzer is a comprehensive personal finance management application designed to help users efficiently track, analyze, and manage their expenses. It features a **user-friendly GUI**, leverages **FastAPI** for high performance, and utilizes **MongoDB** for robust data storage. With advanced **machine learning algorithms**, the app offers insightful expense categorization, predictive analytics, and real-time data processing, empowering users to make informed financial decisions effortlessly. |
| **Members & Task Assignments** | **Tashi Garg**: GUI  **Ekta Awasthi**: Backend  **Ikedinachim Ugochukwu**: Database  **Dongyang Su**: ML |
| **Technical Highlights** | **Backend**: FastAPI, TensorFlow, Scikit-learn  **Frontend**:Jinja2 Templates, Bootstrap, JQuery  **Database**: MongoDB  **Machine Learning Models**: LSTM for expense prediction, MinMaxScaler for data normalization, NearestNeighbors for savings recommendation. |
| **Github Link** | [Group2 Project - Expense Tracker & Analyzer](https://github.com/tashigarg19928/CS531Project) |
| **Project Demo** | [Project\_Demo.mp4](https://drive.google.com/file/d/1WPNq3P9V7zr1H2JrZE0D1mmUx2qr2mq8/view?usp=drive_link)  [Expense Tracker & Analyzer.mp4](https://drive.google.com/file/d/1t_FkRw1tf08SVhaCmvMAl7ZT9H8m0QOV/view?usp=sharing) |
| **Presentation slides** | [CS531 Final Presentation - Group2 - Expense Tracker & Analyzer](https://docs.google.com/presentation/d/1Nd8msp_WVFi0yakHBYawiosSH32OEIz63J867RSbHkY/edit#slide=id.g320d2e51fac_2_0)  [CS531 Project Proposal - Group2 - Expense Tracker & Analyzer](https://docs.google.com/presentation/d/1bv2Cdo0NOuLTNK8XxJxI1UhKKSn6D2PybBsrtWj8LtE/edit#slide=id.g3147ecffb98_0_25) |
| **Reference** | [**Personal Finance App**](https://github.com/AliMahmudi886/Personal-Finance-App) **By Ali Mahmudi**   * Flask, SQLite, HTML, CSS, Javascript |

## 

## Project Folder Structure

├── CS531Project

│ ├── app/

│ │ ├── \_\_init\_\_.py

│ │ └── routes.py

│ ├── static/

│ │ ├── icons/

│ │ ├── profile\_images/

│ │ └── style.css

│ ├── templates/

├── .gitignore

├── analysis.py

├── app1.py

├── m\_db\_database.py

├── ml\_model.py

├── models.py

├── README.md

├── recommendations.py

└── requirements.txt

## Project Web Application Initialization

1. **Set up a virtual environment**

Create a Virtual Environment:

python -m venv venv

1. **Activate the Virtual Environment:**

On Linux/macOS

source venv/bin/activate

On Windows:

venv\Scripts\activate

1. **Install Dependencies:**

pip install -r requirements.txt

1. **Run the Application**

uvicorn app1:app --reload

App runs on [*http://127.0.0.1:8000*](http://127.0.0.1:8000)

## Frontend and GUI Tech Stack

### Tech Stack

* **Jinja2 Templates**:  
  A Python-based templating engine used for rendering dynamic HTML pages with data injected from the backend.
* **jQuery**:  
  A lightweight JavaScript library for simplifying DOM manipulation, event handling, and AJAX functionality.
* **Bootstrap**:  
  A responsive, mobile-first front-end framework for building modern web interfaces with pre-designed components and utilities.
* **Chart.js**:  
  A JavaScript library for creating interactive and visually appealing charts and graphs (e.g., bar, line, pie). Useful for data visualization.
* **DataTables.js**:  
  A jQuery plugin that enhances HTML tables with features like sorting, filtering, pagination, and dynamic data loading for better user interaction.

This extended stack supports dynamic server-side rendering, responsive UI design, interactive data visualization, and enhanced table manipulation.

### Task Breakdown

#### UI/UX Design

* Design the user interface for key components, including the main dashboard, content creation tools, and analytics pages.
* Create wireframes and prototypes to map user flows, prioritizing intuitive navigation and usability.
* Develop responsive designs to ensure seamless functionality across desktop, tablet, and mobile devices.

#### Frontend Development

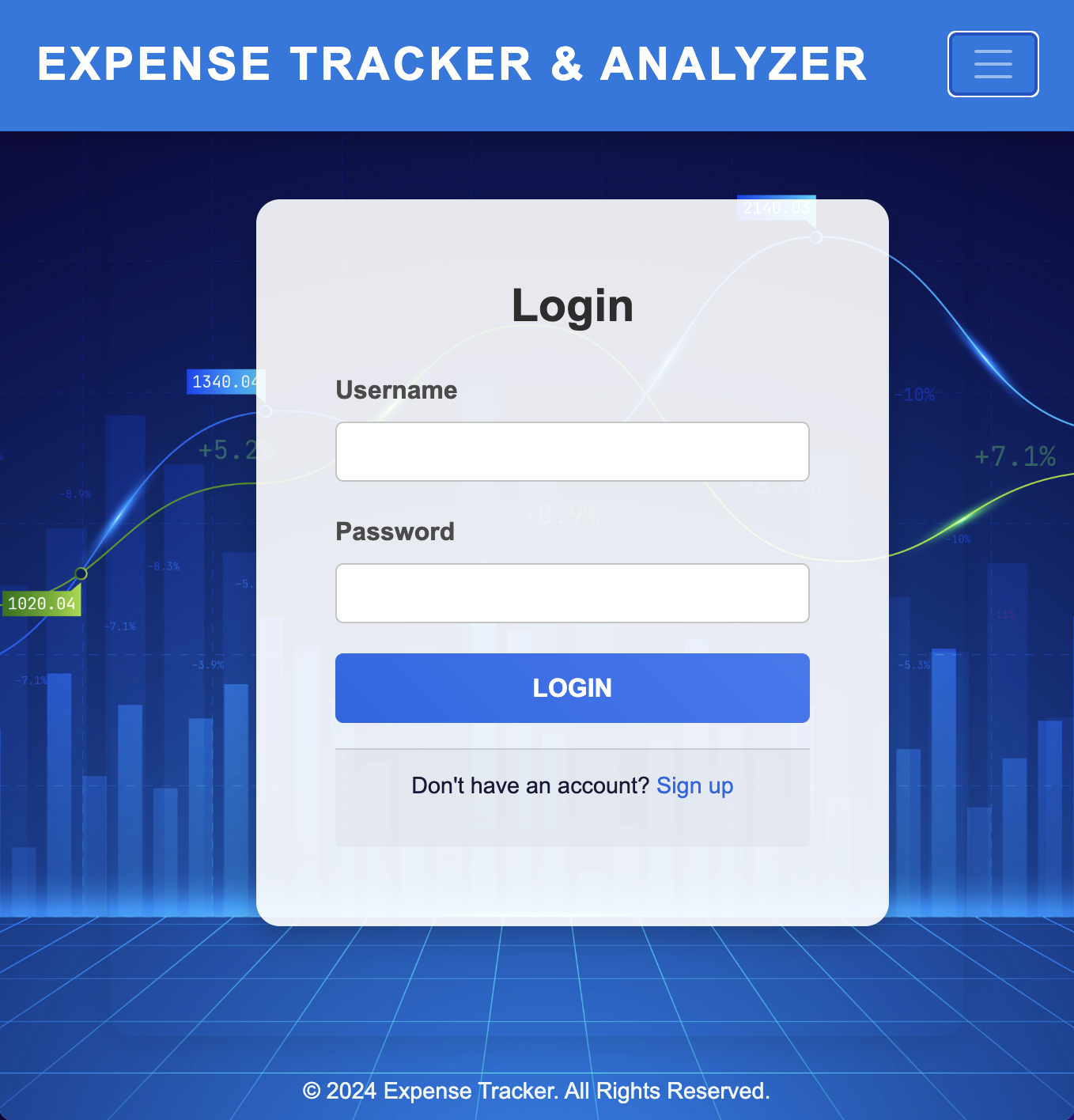
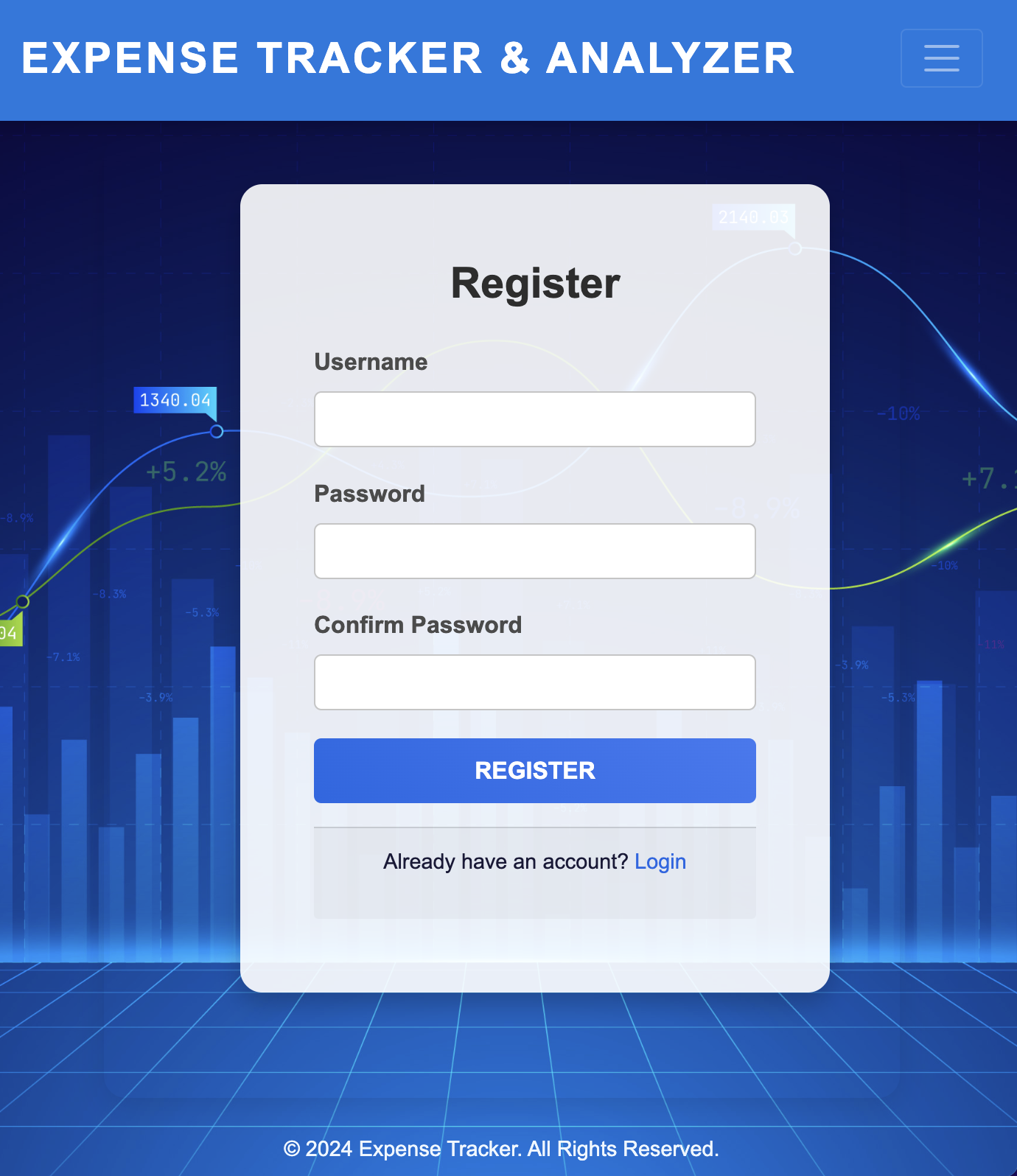
* Build the user interface based on the UI/UX designs, maintaining a cohesive and polished appearance throughout the platform.
* Implement interactive features, such as drag-and-drop content creation tools and real-time analytics visualizations.
* Integrate frontend components with backend services to enable efficient data retrieval and dynamic content display.

## Handling Secure User Authentication

A user authentication and profile management system ensures secure access and personalized user experiences. Powered by bcrypt, it ensures password encryption and security while providing features for user account management.

### 1. User Registration (Sign Up)

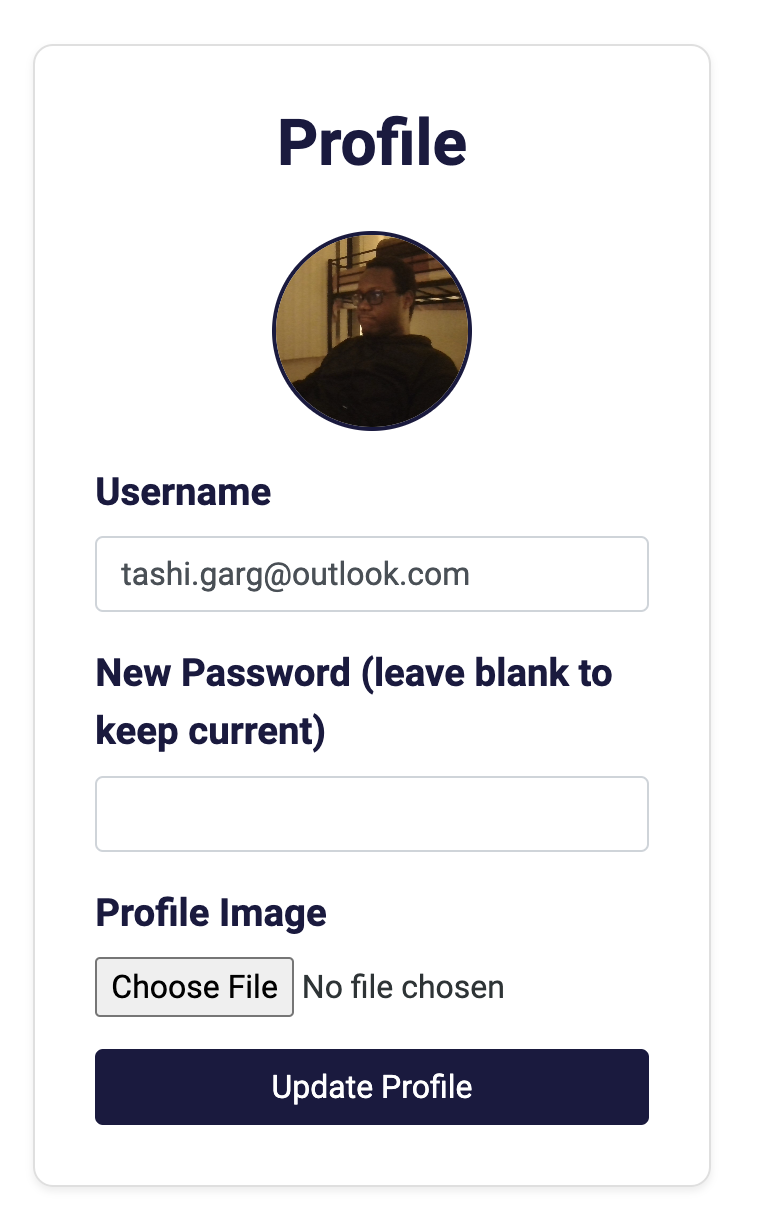
* Users provide a **username**, and **password** to create an account.
* **Password Security**:
  + Passwords are hashed using bcrypt before being stored in the database to prevent plaintext storage.
  + Hashing adds a unique salt to each password to make it resistant to brute-force or rainbow table attacks.
* **Response**:
  + On successful registration, users are redirected to the login page automatically.



### 2. Login/Logout

* **Login**:
  + Users input their registered username and password.
  + The system retrieves the hashed password from the database and uses bcrypt to verify the password.
  + Upon successful login:
    - A session is created for the user to maintain authentication.
    - Users are redirected to their dashboard
  + On failure:
    - Display an error message like "Invalid credentials".
* **Logout**:
  + Ends the user's session by clearing authentication tokens or cookies.
  + Redirects to the login or homepage.

### 3. Update Profile



* **Editable Fields**:
  + Users can update their **username**, **profile photo** or **password**.
* **Password Updates**:
  + Require the user to enter their current password to authorize the change.
  + The new password is hashed using bcrypt before saving.

## Handling Session Management

The session management in the provided code relies on **cookies** for maintaining user authentication and personalization throughout the application.

### **1. Session Initialization**

* **User Login**:
  + When a user logs in successfully (/login endpoint), a cookie named user\_id is set in the response.
  + This cookie holds the user's unique identifier (user["\_id"]) and has a validity period of **30 minutes** (max\_age=30 \* 60).

### **2. Session Validation**

* **Authenticated Access**:
  + Many endpoints, such as /home, /view\_expenses, /add\_income, etc., check the presence of the user\_id cookie to determine if the user is authenticated.
  + If the cookie is missing or invalid, the user is redirected to the login page or presented with an error.

### **3. Session Logout**

* **Logout**:
  + On logging out (/logout endpoint), the application deletes the user\_id cookie by omitting its value in the response.
  + This effectively invalidates the session for the user.

### **4. Flash Messages**

* Flash messages for user feedback (e.g., "Expense added successfully") are managed via temporary cookies.
  + The cookies are set with a short validity period (max\_age=5) and are deleted

### **5. Session Expiry**

* Since the user\_id cookie has a defined expiration (30 minutes), it provides a time-bound session.
  + After expiration, the user must log in again to generate a new session.

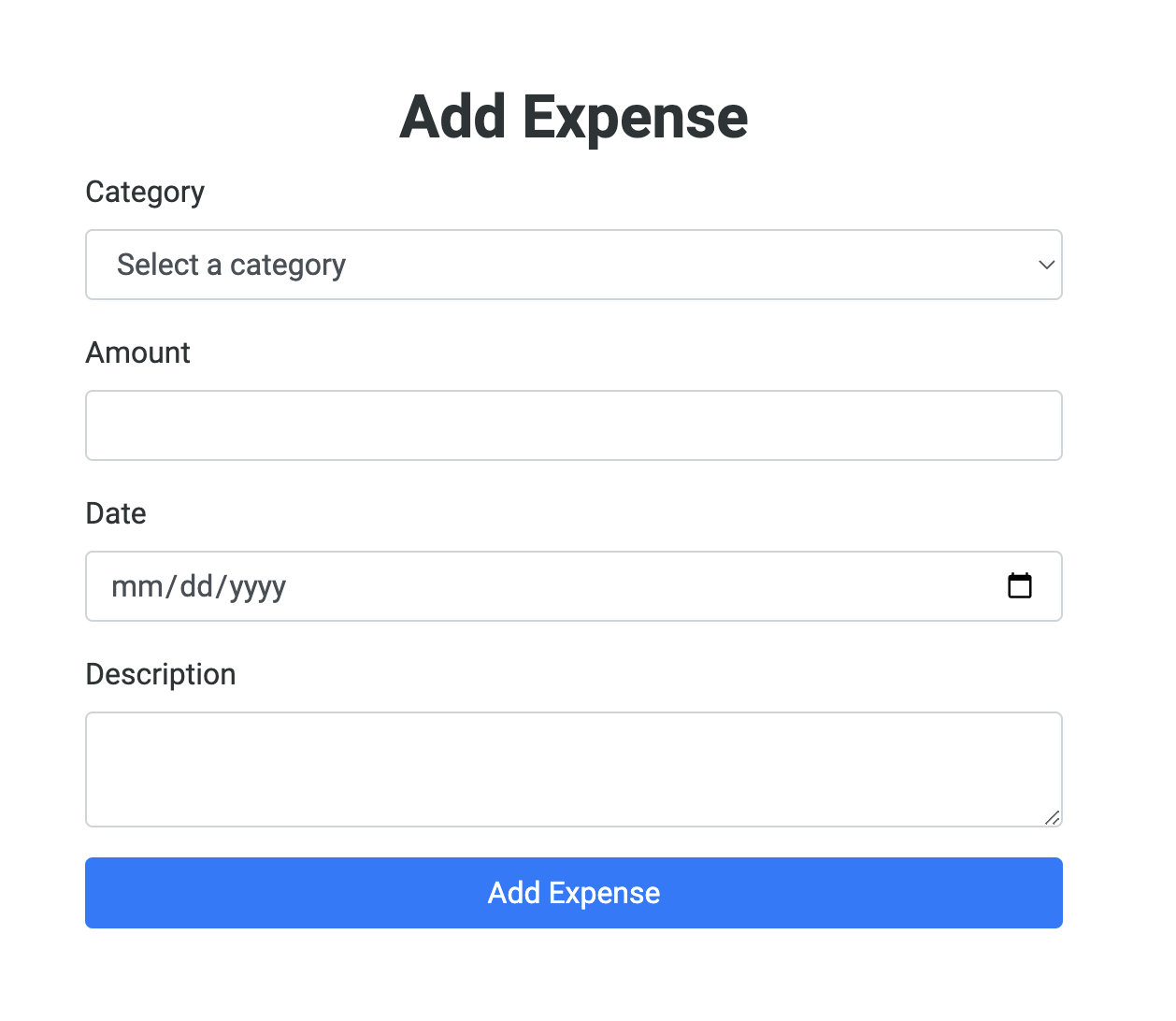
## GUI Workflows and API endpoints

### **Add Expense**

**Path**: /add\_expense

The **/add\_expense** endpoint supports both **GET** and **POST** methods, enabling users to add new expenses to their accounts.

* When accessed via a **GET request**, the endpoint renders the add\_expense.html template, displaying a form where users can input details such as the expense category, amount, description, and date.
* Upon receiving a **POST request**, the endpoint processes the submitted form data by validating all fields to ensure completeness and accuracy. It inserts the new expense into the expenses table, and the user is redirected to the homepage (/) with a flash message confirming the addition.

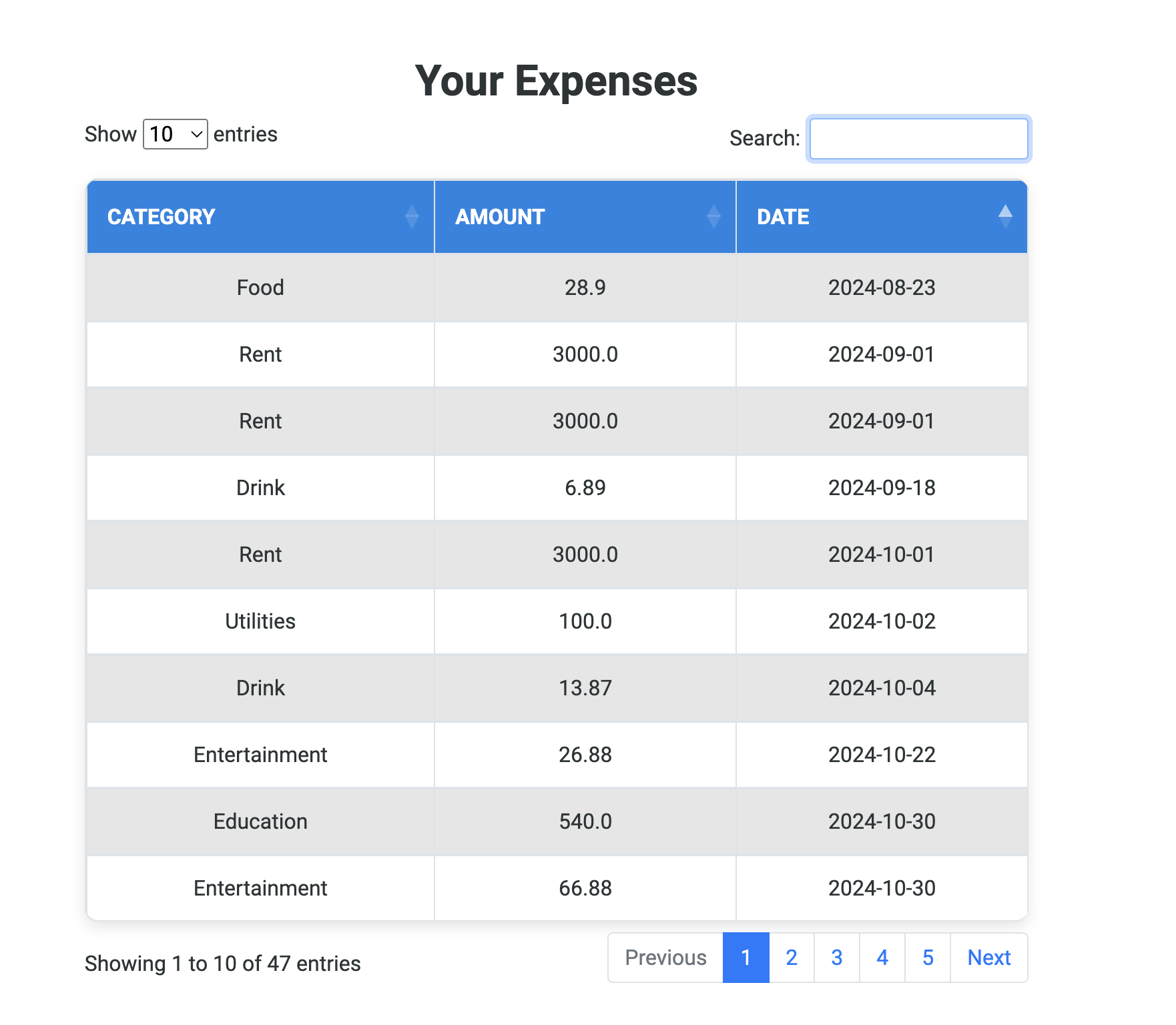


### **View Expenses**

### **Path:** /view\_expenses

The **/view\_expenses** endpoint supports the **GET method**, allowing users to view a detailed list of their recorded expenses.

* It retrieves all expense records from the expenses table that are associated with the logged-in user.
* The retrieved data is then passed to the view\_expenses.html template, where it is dynamically displayed in an organized format, such as a table or list, for easy analysis.

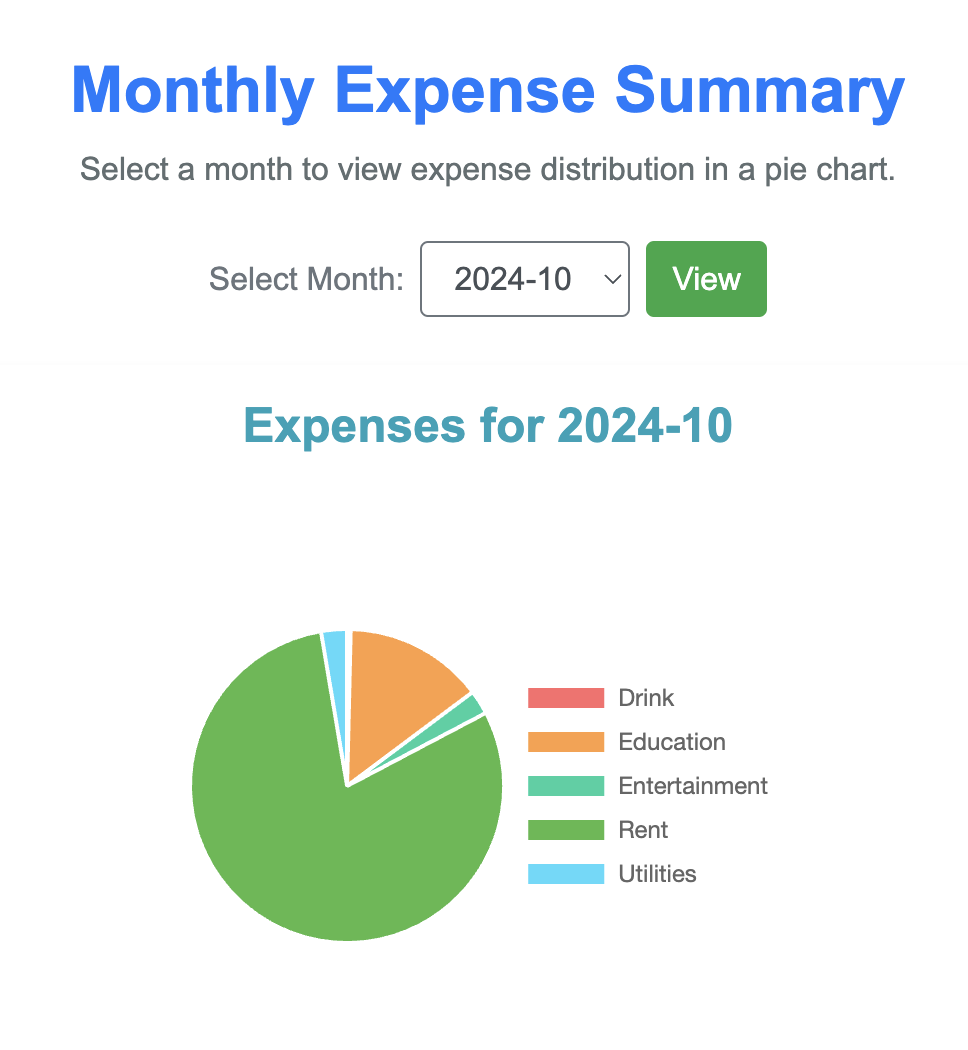


### 

### **Monthly Expense Summary**

**Path:** /expense\_summary

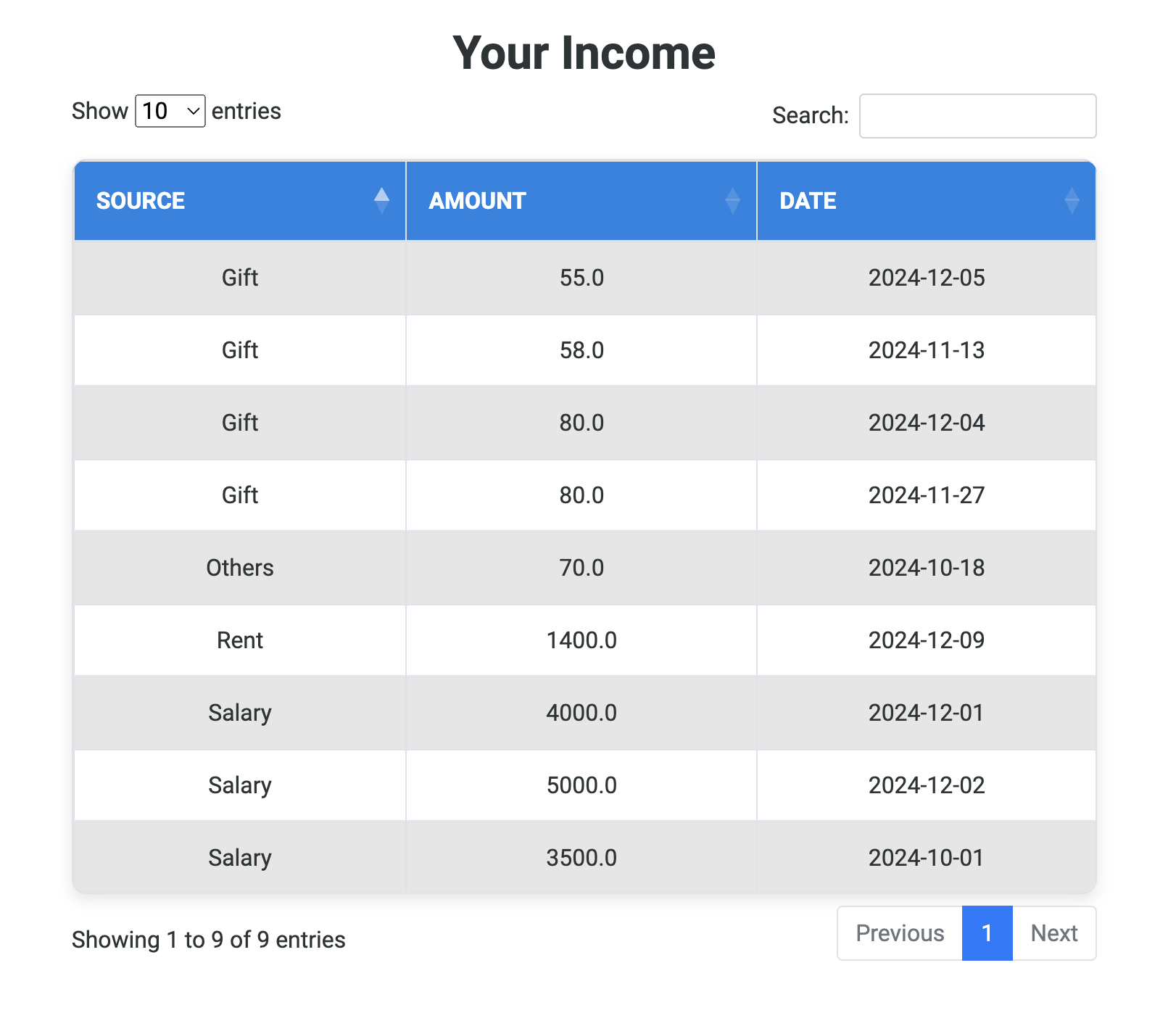
* The **/expense\_summary** endpoint uses the GET method to provide a categorized monthly summary of expenses.
* It fetches and aggregates expense data from the database by category and month, displaying it in the expense\_summary.html template.



### **View Income**

**Path:** /view\_income

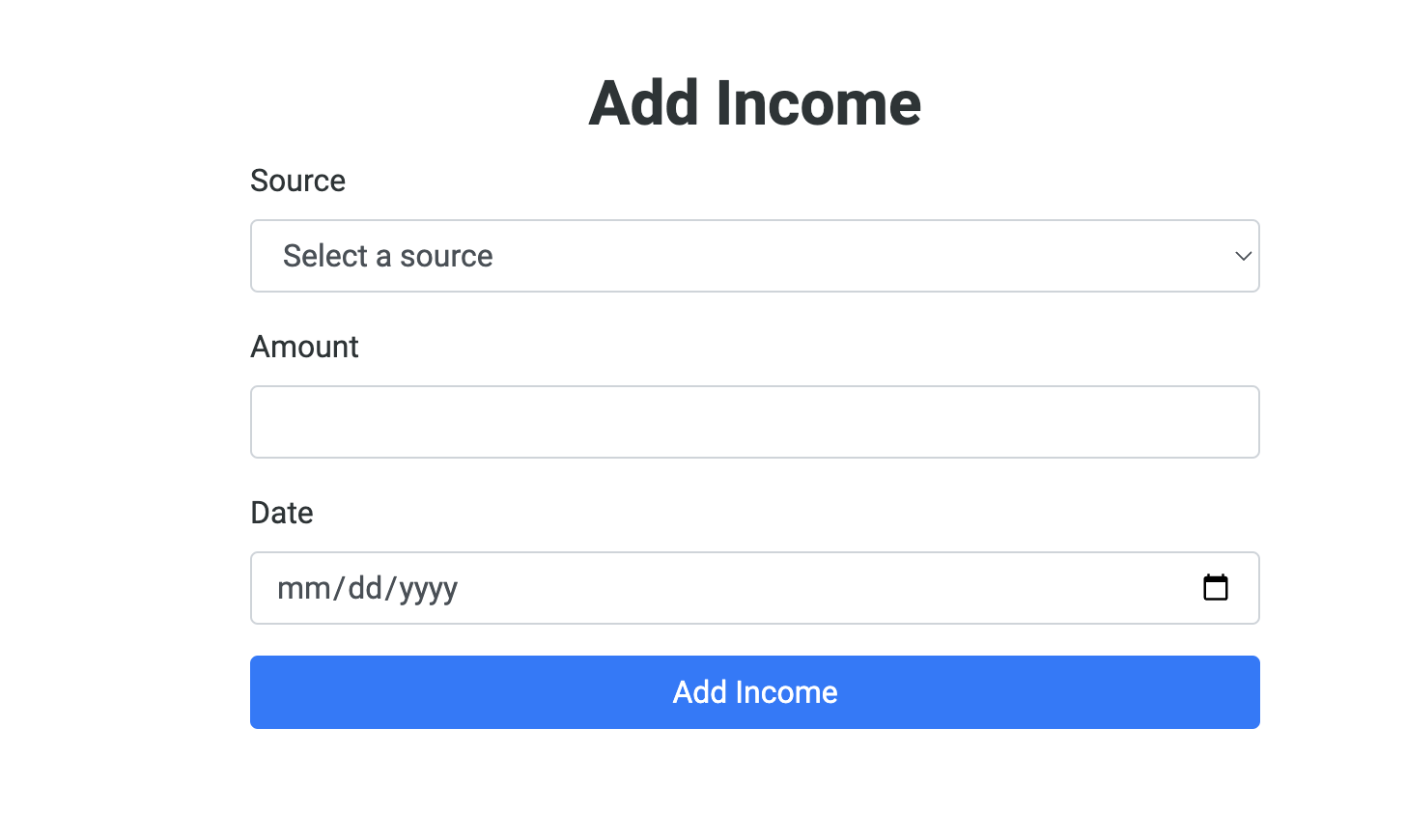
* The **/view\_income** endpoint supports the GET method to display a detailed list of all income records.
* It retrieves income data for the logged-in user and renders it in the view\_income.html template.



### **Add Income**

**Path:** /add\_income

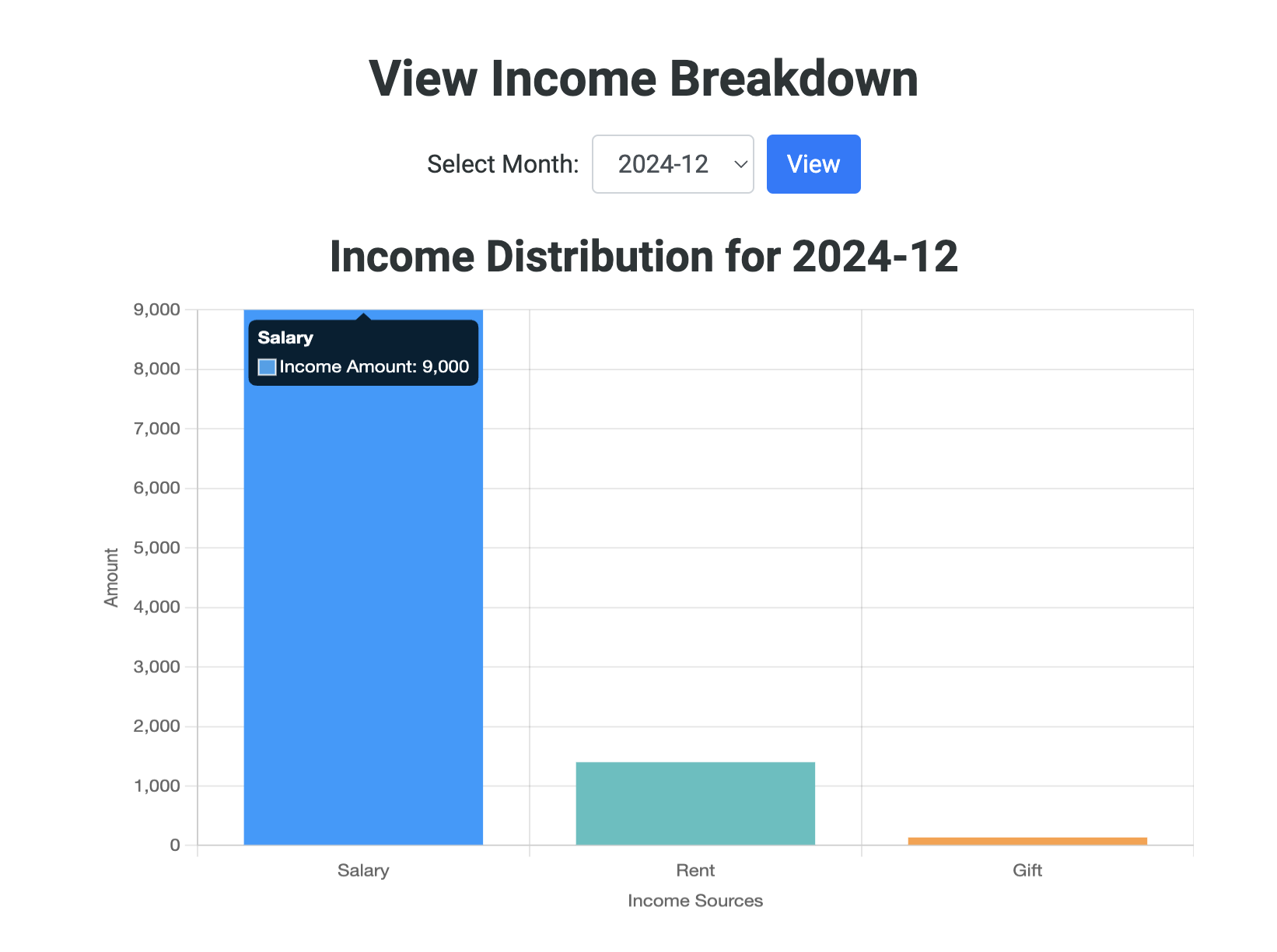
* The **/add\_income** endpoint supports both GET and POST methods to facilitate adding new income entries.
* It renders a form on GET and processes submitted data on POST to insert new records into the income table.



### **Monthly Income Breakdown**

**Path:** /income\_summary

* The **/income\_summary** endpoint uses the GET method to provide a categorized monthly breakdown of income.
* Aggregated income data is displayed in the income\_summary.html template for analysis.



## Machine learning Model

### **Predict Expenses**

**Path:** /predict\_expenses

* The **/predict\_expenses** endpoint supports the GET method to forecast future expenses using historical user data.
* It trains an LSTM model and generates predictions, which are dynamically rendered in the predict\_expenses.html template.

**Machine Learning Details:**

* Model Type: LSTM (Long Short-Term Memory)
* LSTM is a type of recurrent neural network (RNN) designed for sequence prediction tasks, making it well-suited for time series data such as user expenses. It is ideal for forecasting expenses based on historical patterns.

**Steps:**

Data Fetching:

* The endpoint fetches historical expense data for a user using the helper function ***fetch\_expense\_data(user\_id)***, which retrieves expenses from **MongoDB** and prepares the data for training by formatting the date column and ensuring the amount column contains valid numeric values.

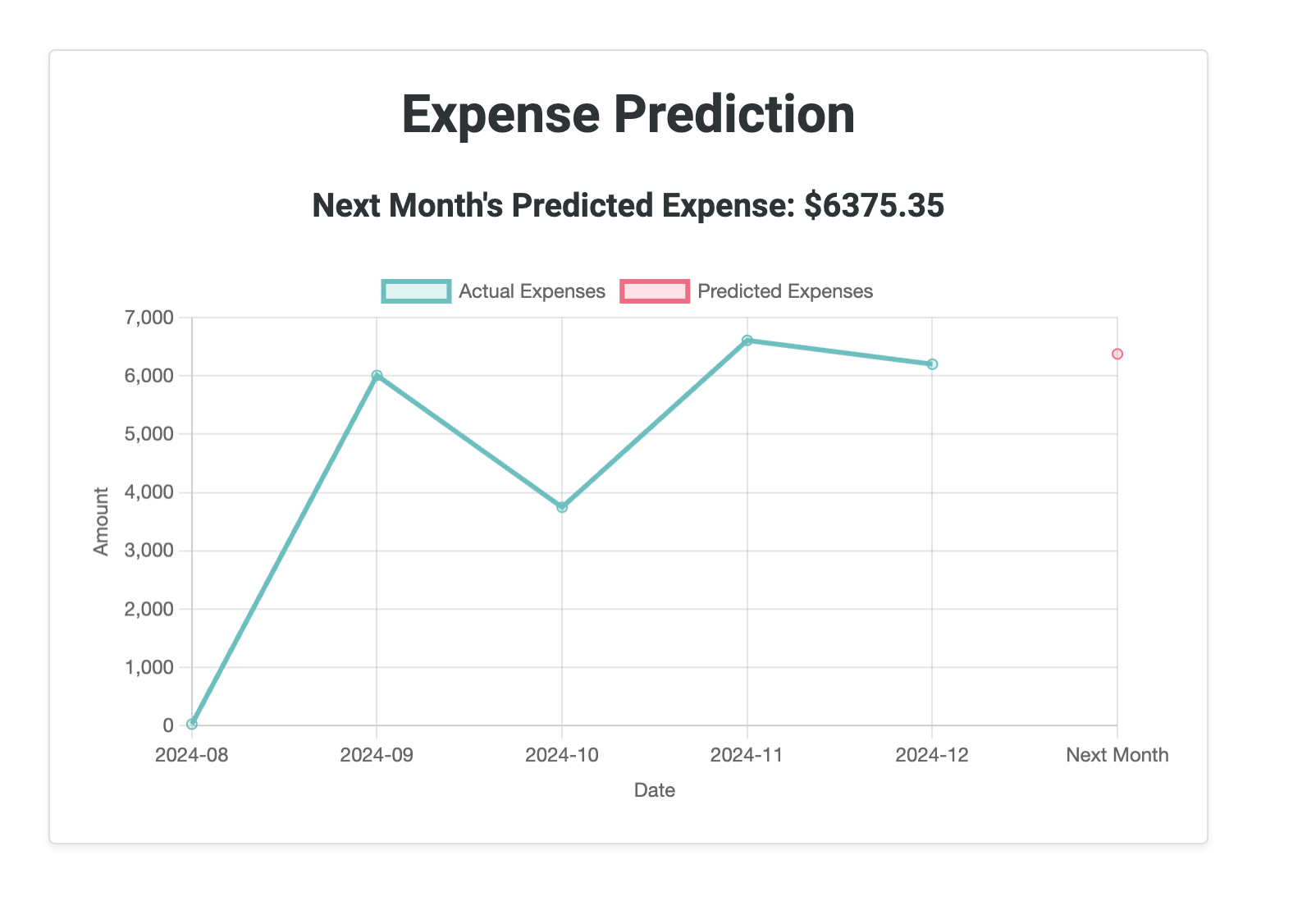
Data Preprocessing:

* The data is resampled by month-end (ME), and the total amount for each month is used for training. The **MinMaxScaler** **normalizes the data** to ensure values are in the range [0, 1], which is important for LSTM models to train effectively.

Model Training:

* The **LSTM model is trained with the preprocessed data**. The model is defined using the **Sequential API from TensorFlow**, with two LSTM layers (each with 50 units) followed by a **Dense layer**. The **mean\_squared\_error (MSE) loss function** is used for training.

Prediction:

* After the model is trained, the most recent expense amount is scaled and used to predict the next month's expenses. The prediction is then inversely transformed to return the value to the original scale (i.e., the amount). The response will include the predicted value for the next month's expenses. 

### **Recommend Savings**

**Path:** /recommend\_savings

* The **/recommend\_savings** endpoint supports the GET method to suggest a savings plan based on user expense patterns.
* The calculated savings recommendation is passed to the recommend\_savings.html template for display.

**Steps**:

Data Fetching:

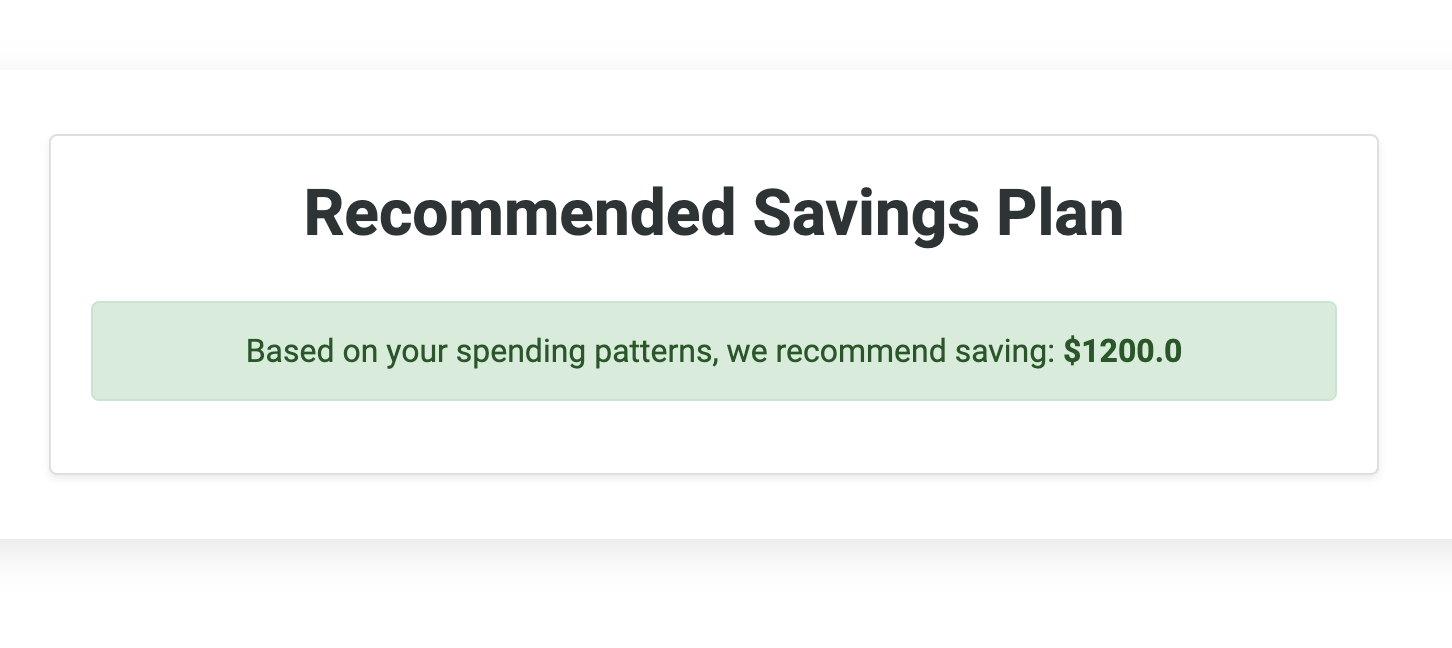
* The endpoint fetches the user’s expense data using the ***fetch\_expense\_data(user\_id)*** function, ensuring that the necessary columns (date and amount) are present and valid.

Model Training (**Nearest Neighbors**):

* The algorithm fits the **NearestNeighbors** model using the amount of expenses. The model identifies the closest match based on the most recent expense amount, making a recommendation based on similar spending patterns.

Recommendation Calculation:

* After fitting the **NearestNeighbors** model, the closest match to the last expense is identified, and the corresponding amount is returned as the savings recommendation.



## 

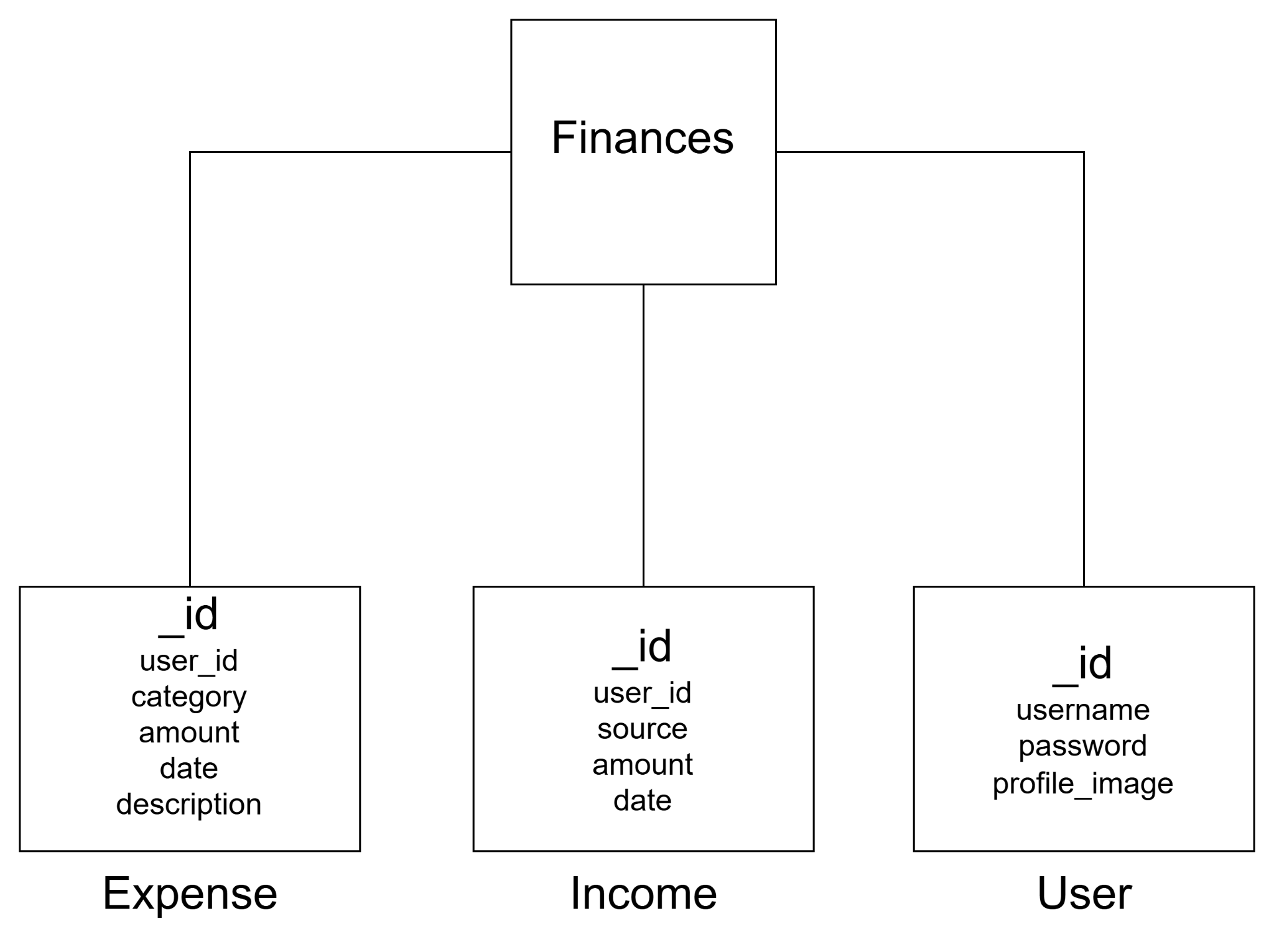
## Database and Storage

### Data Operations

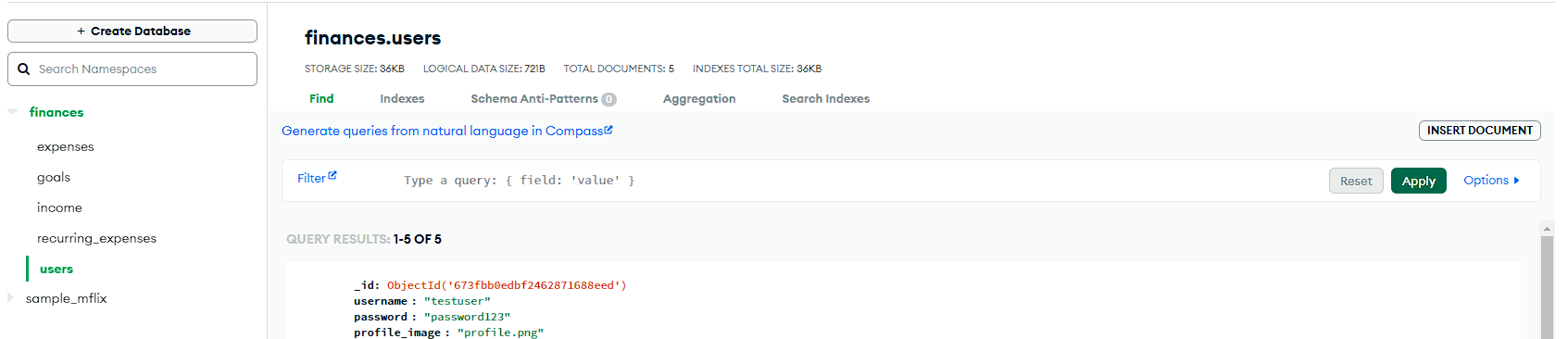
For this project, we made use of Mongo DB for data storage. Mongo DB helped us with the following operations:

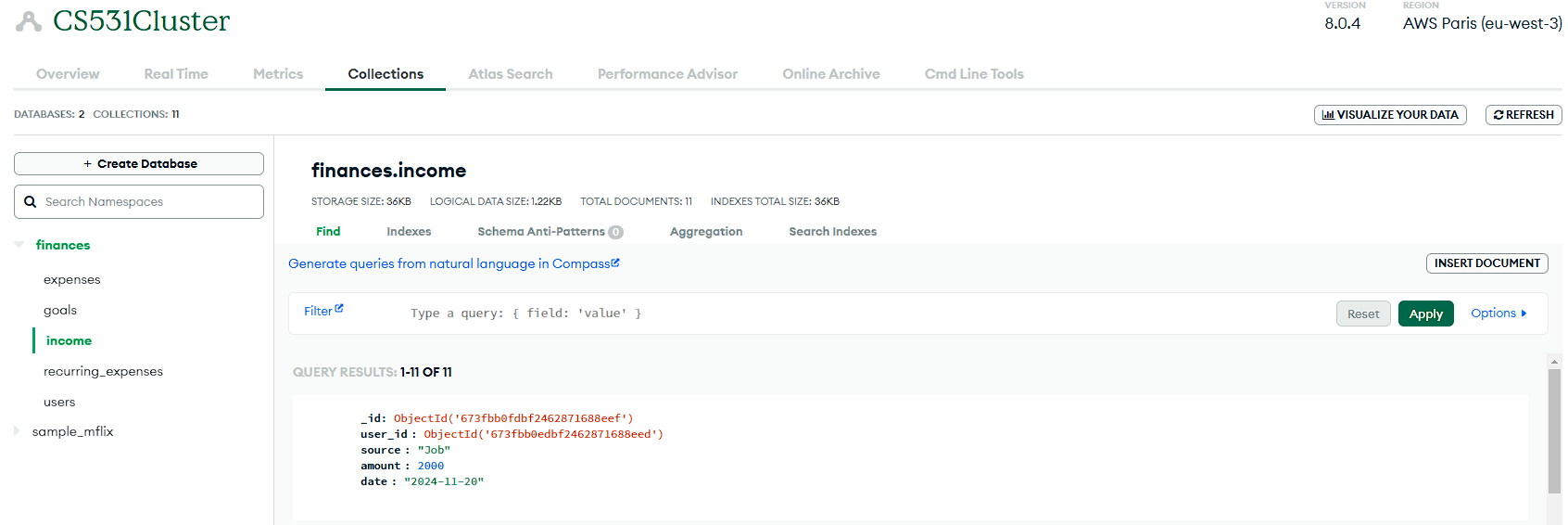
1. User Authentication and Registration
2. Create Expenses/Income/Users
3. Add Expenses/Income/Users
4. Update Expenses/Income/Users
5. Delete Expenses/Income/Users

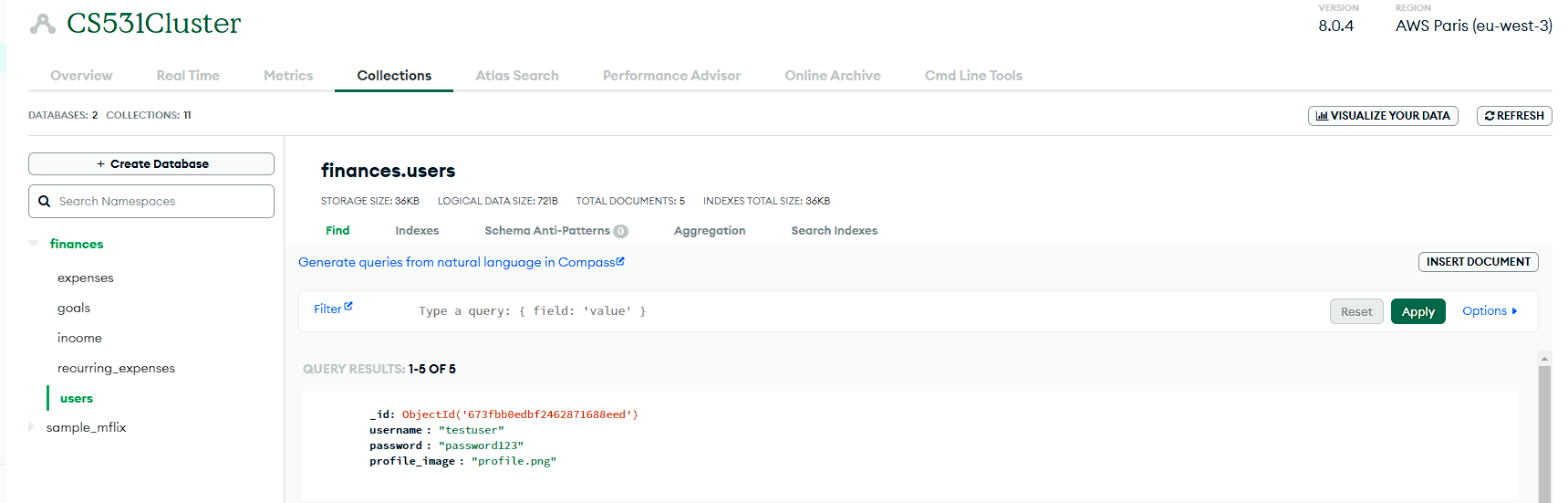
Data Tables

The diagram below shows the structure of our project’s database:

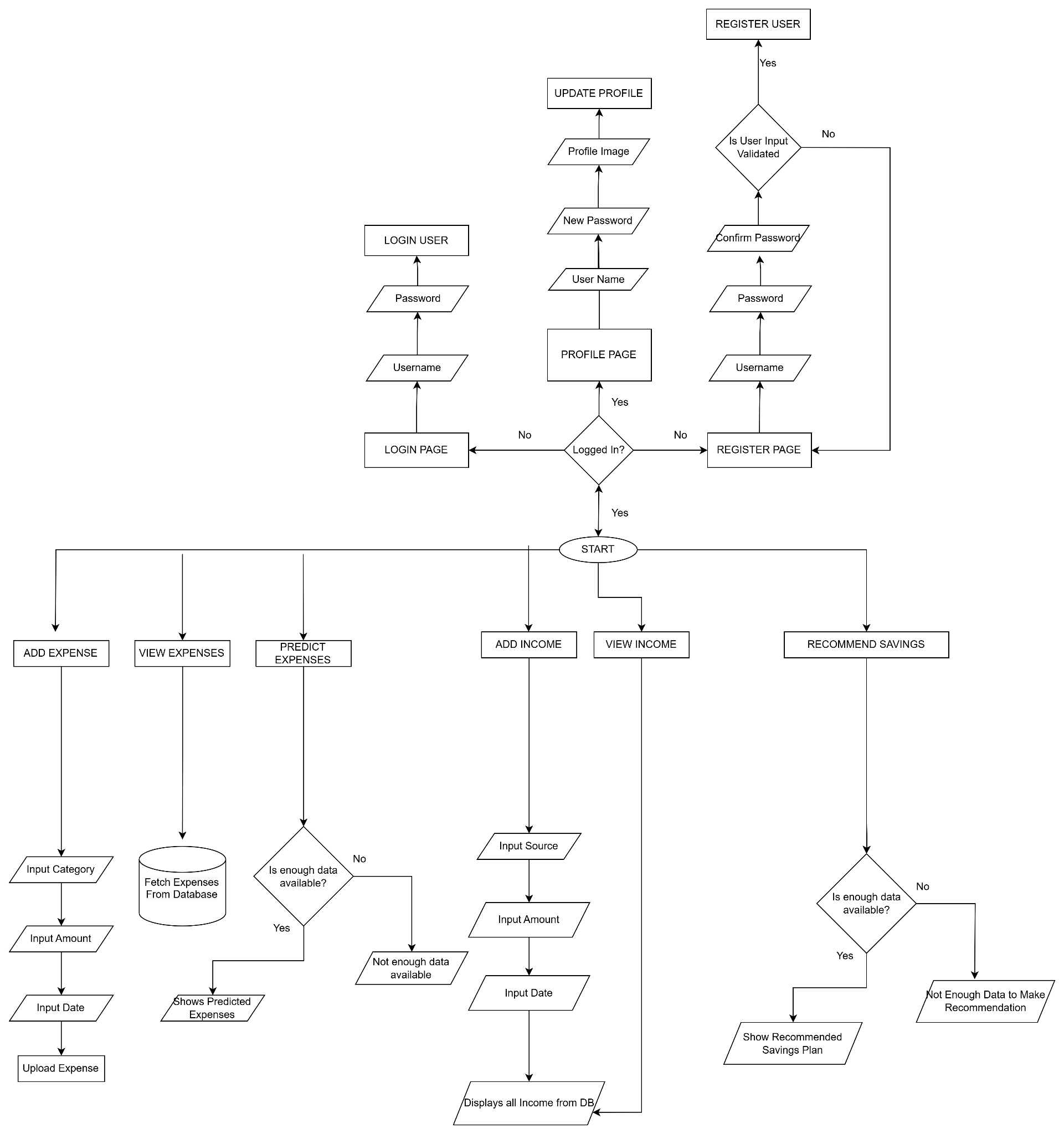
### Screenshots







### ER Diagram

****

### Process

The process of setting up Mongo DB in our project involved the following steps:

1. Installing Dependencies
2. Setting Up MongoDB Connection
3. Defining the Collection
4. Setting Up Models
5. Setting Up Routes
6. Testing the Application

More about this can be found in the [documentation](https://www.mongodb.com/developer/languages/python/python-quickstart-fastapi/) provided by the MongoDB team.

### 

## **Why FastAPI for our Backend Development?**

For our project Expense Tracker & Analyzer, we chose FastAPI as the Python web framework for the backend because it provides various powerful features that perfectly align with the needs of our application. Here are some of the factors that made FastAPI an excellent choice for our application.

* Easy to use and flexible in nature
* High Performance
* Asynchronous Programming
* Scalability and Extensibility
* Built-in security features
* Automatic data validation and documentation

Since FastAPI is the fastest framework available for building APIs and as our app involves processing and analyzing financial transactions, the speed and responsiveness of our app is extremely important for providing a seamless experience to the users. And that is one of the reasons to choose FastAPI for our backend Development.

#### **FastAPI Requirements**

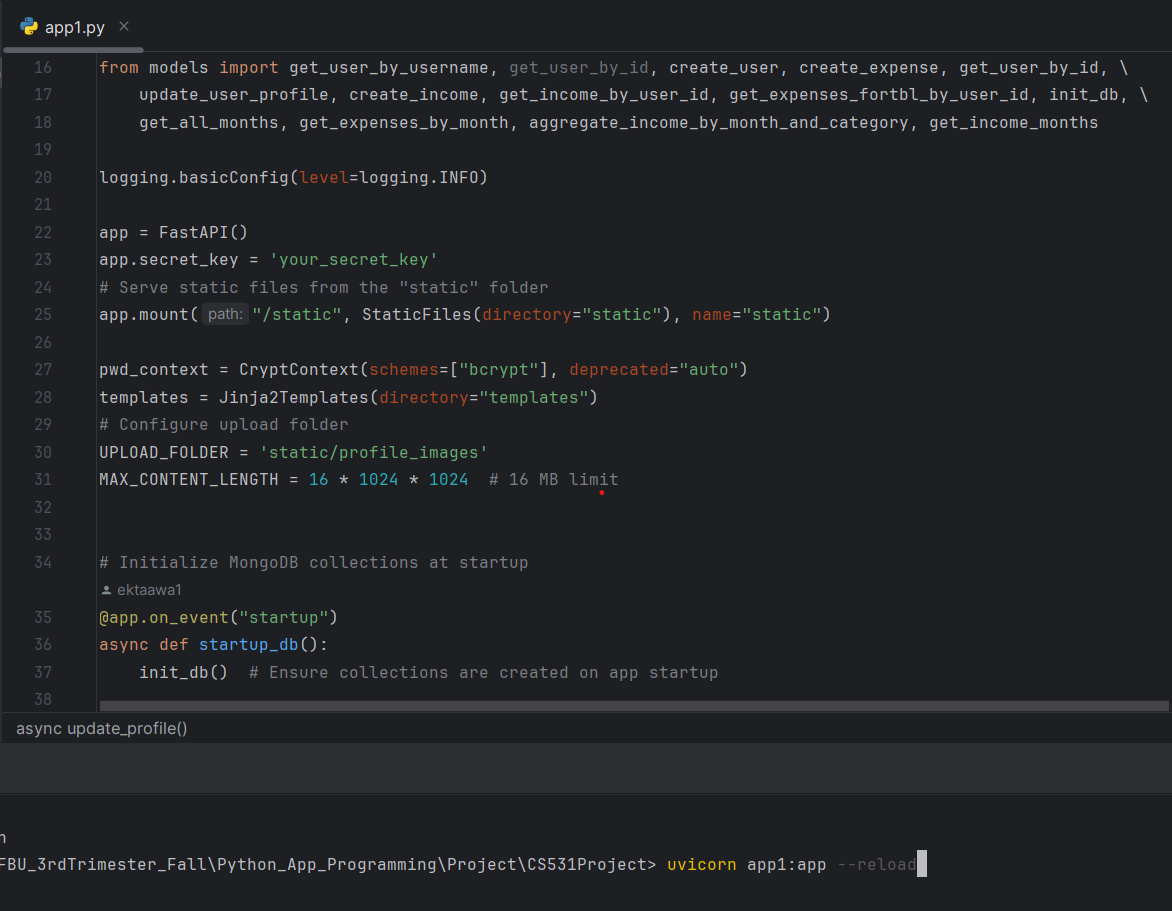
* Starlette for the web parts
* Pydantic for the data validation

#### **FastAPI Installation**

Create and activate the virtual environment and then install fastapi using the below command.

* pip install fastapi

Once the framework was installed, created a file ‘app1.py’ to create all the APIs required for our expense tracker and analyzer application. Below is the code snippet of the file.



**Add Expense API-**

