Personal Finance Management System

A REPORT

Submitted by

Pranjal Upadhyay (RA2311003030063)

Under the guidance of

# Dr. Amit

(Department of Computer Science & Engineering)

in partial fulfilment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

# COMPUTER SCIENCE & ENGINEERING

of

FACULTY OF ENGINEERING AND TECHNOLOGY



SRM INSTITUTE OF SCIENCE & TECHNOLOGY, NCR CAMPUS

2025

SRM INSTITUTE OF SCIENCE &

TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

# BONAFIDE CERTIFICATE

Certified that this report titled “Personal Finance Management System” is the bonafide work of Pranjal Upadhyay(Ra2311003030063)who carried out this work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form any other report on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

|  |  |
| --- | --- |
| Signature  Dr. Amit  Assistant Professor  Dept. of Computer Sciene &  Engineering | Signature  Dr. Avneesh Vashistha  HEAD OF THE DEPARTMENT  Dept. of Computer Science & |

Engineering

ACKNOWLEDGEMENT

We would like to express our gratitude to our project supervisor, Dr. Amit for his guidance and support. Special thanks to the members of our team for their dedication and collaboration. Lastly, we thank our families and friends for their encouragement throughout this journey. This project benefited greatly from the collective effort and support of everyone involved.

Abstract:

This project focuses on developing a TCP-based chat application using Java with a graphical user interface (GUI) created using Java Swing. The application facilitates real-time communication between multiple clients through a serverclient architecture, where each client connects to a central server that manages message broadcasting. This report delves into the design, implementation, testing, and evaluation of the chat system, emphasizing how Java’s multi-threading capabilities and socket programming were utilized to achieve smooth, concurrent message handling.

The primary objectives of this project were to build a user-friendly interface for chat communication, ensure stable and secure data transmission, and develop a server that could efficiently manage multiple client connections. The GUI was designed to provide an intuitive user experience, featuring text input fields, a display area for messages, and buttons for sending messages. The server, built using Java's ServerSocket and Socket classes, acts as a hub, accepting connections from clients and broadcasting messages to ensure all participants receive updates in real-time.

The project implementation included key features like multi-client handling, user name changes, and commands for quitting the chat. Each client, upon connecting, could send messages to the server, which then distributed them to other clients using a multi-threaded approach. This report provides an in-depth analysis of how the server manages concurrent client connections using ExecutorService, ensuring that each client operates in a separate thread, thus maintaining responsive communication.

Testing and evaluation of the chat application were conducted in a controlled local network environment, focusing on aspects such as latency, connection stability, and server load capacity. The tests revealed that the application maintained low latency during message exchanges, with the server capable of handling up to 20 concurrent client connections efficiently. Functional tests confirmed that key features, such as message broadcasting and user name updates, worked as intended, while stress testing demonstrated the application's robustness under high load conditions.

Despite achieving the primary goals, the project also faced challenges, particularly in managing the multi-threaded environment and ensuring smooth GUI responsiveness during high message traffic. These challenges were addressed through careful optimization of the server's thread management and adjustments to the user interface design.

Table of Contents:

|  |  |
| --- | --- |
| CONTENTS | PAGE |
| 1. INTRODUCTION | 6 |
| 2. LITERATURE REVIEW | 6 |
| 3. IMPLEMENTATION | 7 |
| 4. CODE EXPLANATION | 7 |
| 5. RESULTS | 18 |
| 6. CONCLUSION | 21 |
| 7. REFERENCES | 22 |

Introduction:

A Personal Finance Management System using DBMS helps users effectively track and manage their income, expenses, and savings. By using a Database Management System, the application ensures secure, organized, and efficient storage and retrieval of financial data. It allows users to monitor spending habits, set budgets, and generate reports. This system demonstrates the practical use of DBMS concepts like relational modeling, queries, and data integrity in building a reliable and user-friendly financial planning tool.



Literature Review:

The development of Personal Finance Management Systems (PFMS) has gained significant attention in recent years due to the increasing need for individuals to manage their financial activities efficiently. These systems assist users in organizing income, expenses, savings, and investments in a structured manner. Traditional manual methods, such as spreadsheets or paper-based tracking, are prone to errors and lack scalability. The use of Database Management Systems (DBMS) in personal finance applications addresses these limitations by offering efficient data storage, retrieval, and manipulation.

According to various studies, DBMS ensures data consistency, integrity, and security, which are essential for handling sensitive financial information. Systems like Mint, YNAB (You Need A Budget), and PocketGuard incorporate robust database backends to support features like real-time transaction tracking, budget setting, and financial goal monitoring. Academic literature emphasizes the importance of relational models and normalization in designing scalable and maintainable database structures for such applications.

Moreover, DBMS technologies such as MySQL, PostgreSQL, Oracle, and SQLite provide powerful query capabilities that help users generate meaningful financial reports and insights. Research also highlights the growing role of mobile applications and cloud-based databases in enhancing accessibility and user convenience. Overall, integrating DBMS in PFMS leads to more accurate, reliable, and user-friendly financial tools.

Implementation

The implementation of the Personal Finance Management System (PFMS) using DBMS involves both frontend development and backend database integration. The system is designed to allow users to add, update, delete, and view their financial transactions, including income, expenses, and savings, through a user-friendly interface.

The backend is powered by a relational database (e.g., MySQL or SQLite), which stores all financial records in normalized tables. Key tables include Users, Transactions, Categories,

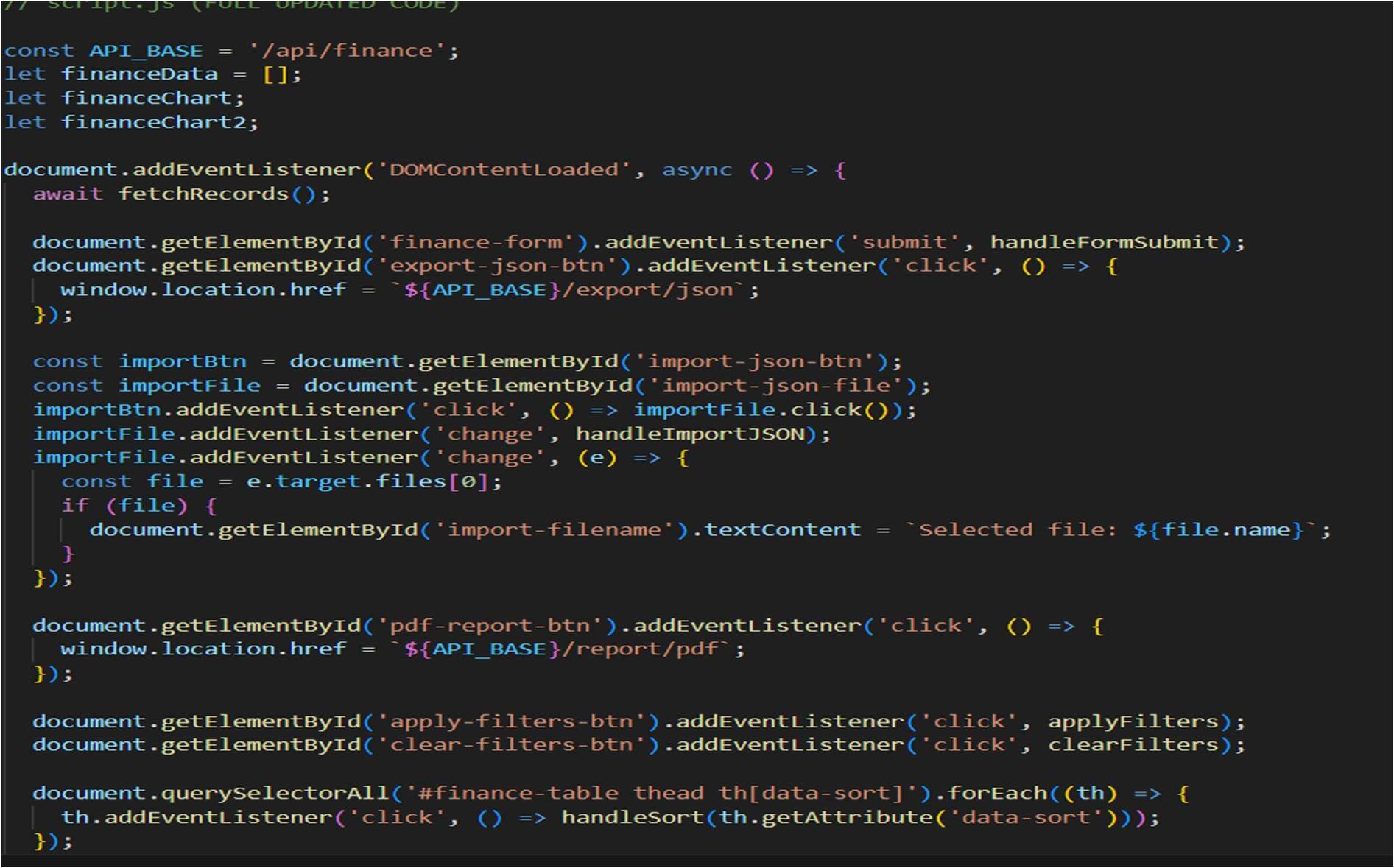
Budgets, and Accounts. Foreign keys and constraints are used to maintain referential integrity. SQL queries are used extensively for data manipulation and retrieval, enabling efficient generation of reports, summaries, and budget tracking.

The frontend is developed using technologies such as HTML, CSS, JavaScript, or any preferred framework (e.g., Python with Flask/Django or Java with Spring). The frontend communicates with the database through a middleware layer using structured queries and APIs to perform CRUD operations.

Security measures, such as input validation and user authentication, are implemented to protect sensitive financial data. Reports such as monthly expenditure, savings trends, and budget status are dynamically generated using query results.

This system demonstrates how DBMS plays a crucial role in managing personal finance data by ensuring data integrity, scalability, and efficient retrieval of useful financial insights.

Code Snippet 1:



In this code :

1. Fetches finance data when the page loads.
2. Handles form submissions for finance data via handleFormSubmit().
3. Exports finance data as JSON when the export button is clicked.
4. Triggers a file import when the import button is clicked, allowing the user to select a file.
5. Displays the selected file's name when a file is chosen for import and then processes it through handleImportJSON().

It's essentially a frontend script for managing finance data, with functionality for importing, exporting, and submitting forms.

Code Snippet 2:



In this code:

In brief, the fetchRecords() function:

1. Fetches data from a finance API using fetch().
2. Parses the response as JSON.
3. If the response is successful (json.success is true):
   * It stores the data in financeData.
   * Updates the UI by rendering a table, summary, and charts.
4. If the response indicates an error, it logs the error message.
5. It handles any issues during the fetch operation (e.g., network problems) by logging an error message to the console.

Essentially, this function fetches finance data and updates the user interface accordingly.

Code Snippet 3:

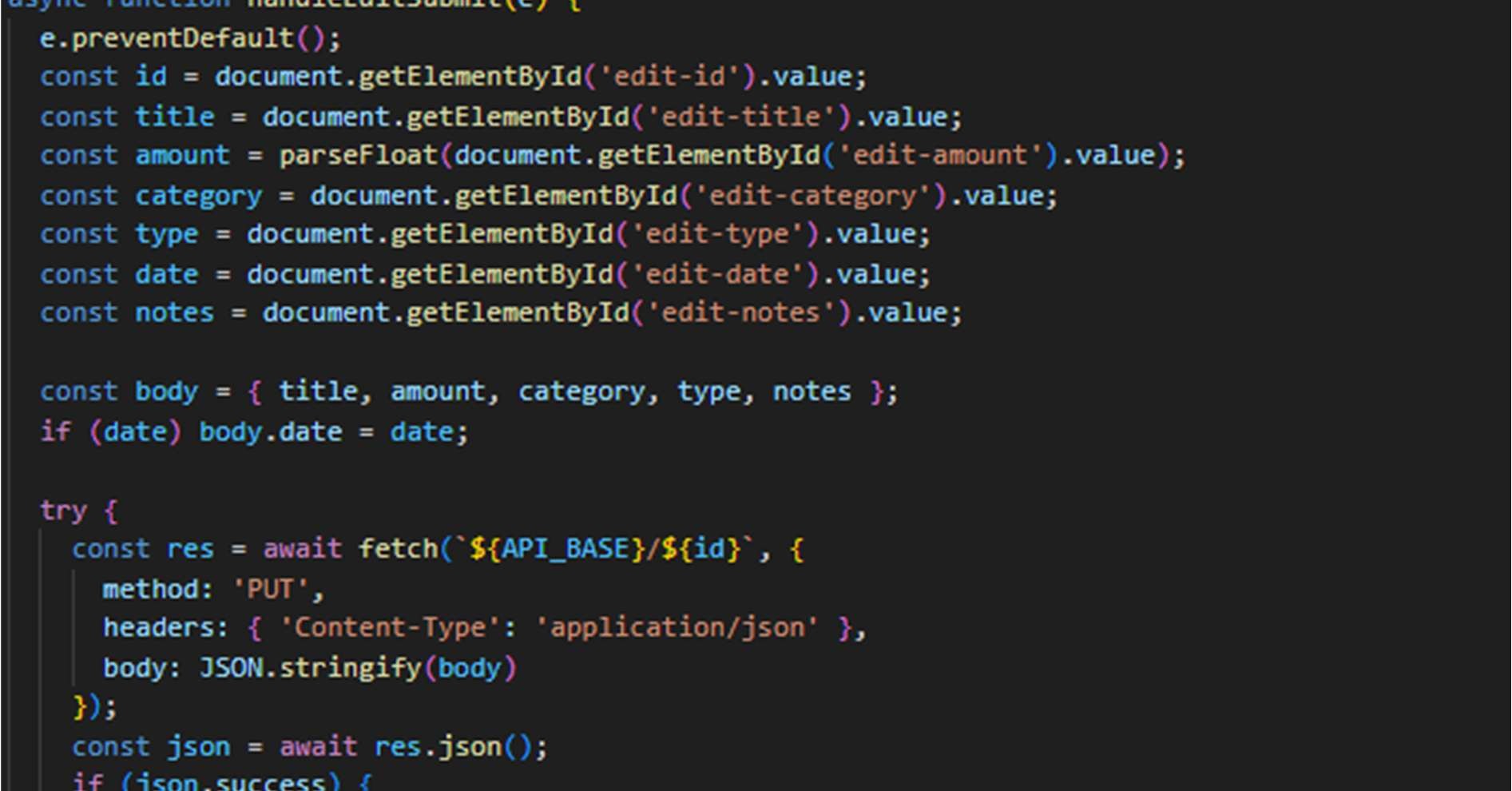


In this code:

1. Prevents the default form submission (page reload).
2. Gathers input values (title, amount, category, etc.) from the form fields.
3. Creates a data object (body) and includes the date if provided.
4. Sends the data to the server via a POST request.
5. If the submission is successful, it:
   * Fetches updated records from the server.
   * Resets the form.
6. Logs an error if the request fails.

This function is responsible for submitting new finance records and updating the page with the latest data.

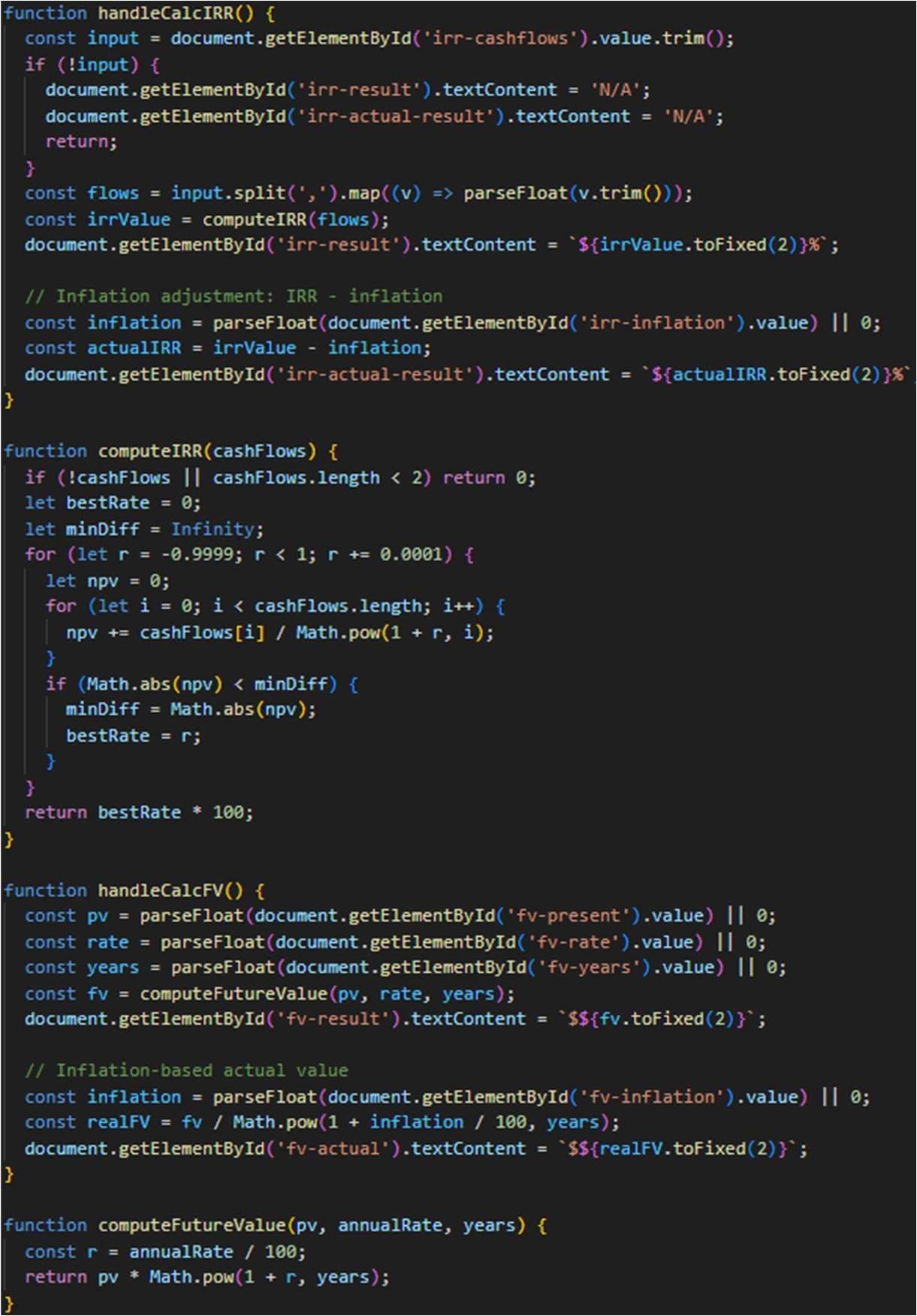
Code snippet 4:



In this code:

1. openEditModal(id):
   * Opens a modal to edit a specific finance record.
   * Fills the modal form with the record's existing details (like title, amount, category).
2. closeEditModal():
   * Closes the edit modal by hiding it.
3. handleEditSubmit(e):
   * Submits the updated finance record when the user saves the changes.
   * Sends a PUT request to update the record on the server.
   * If successful, it closes the modal and refreshes the list of records. Overall, it allows users to edit existing finance records through a modal and update the data on the server.

Code snippet 5:



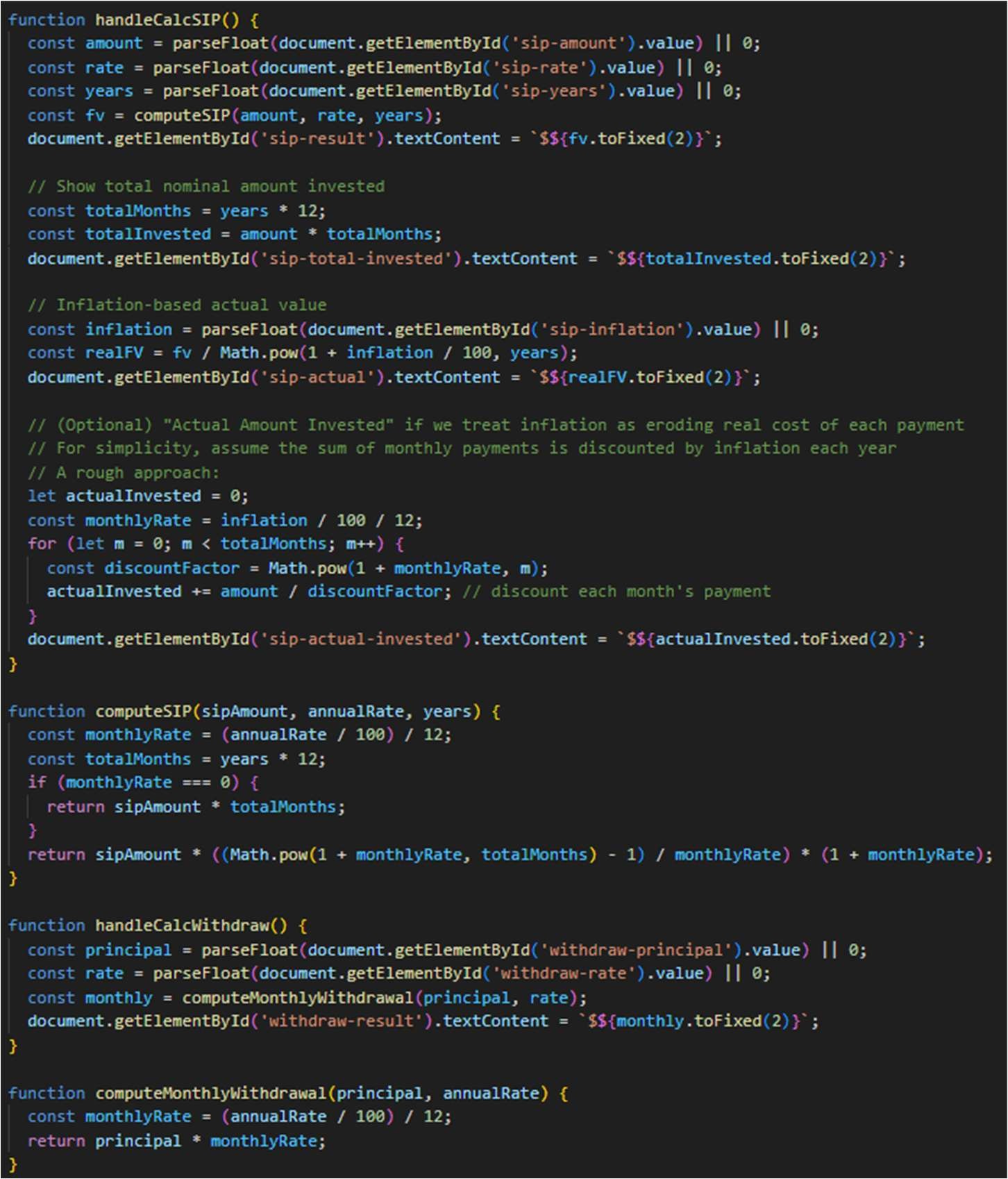
In this code:

1. IRR (Internal Rate of Return):
   * handleCalcIRR():
     + Takes user input of cash flows (comma-separated).
     + Computes the IRR using computeIRR().
     + Adjusts the IRR for inflation and displays both values.
   * computeIRR(cashFlows):
     + Tries different interest rates to find the one where Net Present Value (NPV) is closest to zero. o Returns the best-fitting rate as the IRR.

1. FV (Future Value):
   * handleCalcFV(): o Takes Present Value, interest rate, and years from user input.
     + Calculates Future Value using computeFutureValue().
     + Adjusts for inflation to show the real value in today's terms.

* + computeFutureValue(pv, annualRate, years):
  + Uses the standard compound interest formula to calculate the future value.

code snippet 6:



In this code:

1. SIP (Systematic Investment Plan) Calculator  handleCalcSIP():
   * + Gets the SIP amount, interest rate, and duration (in years) from user input.
     + Uses computeSIP() to calculate the future value (FV) of monthly investments. o Displays:
       - The future value of the investment.
       - The total amount invested.
       - The real future value (adjusted for inflation).
       - The inflation-adjusted amount actually invested over time.

 computeSIP():

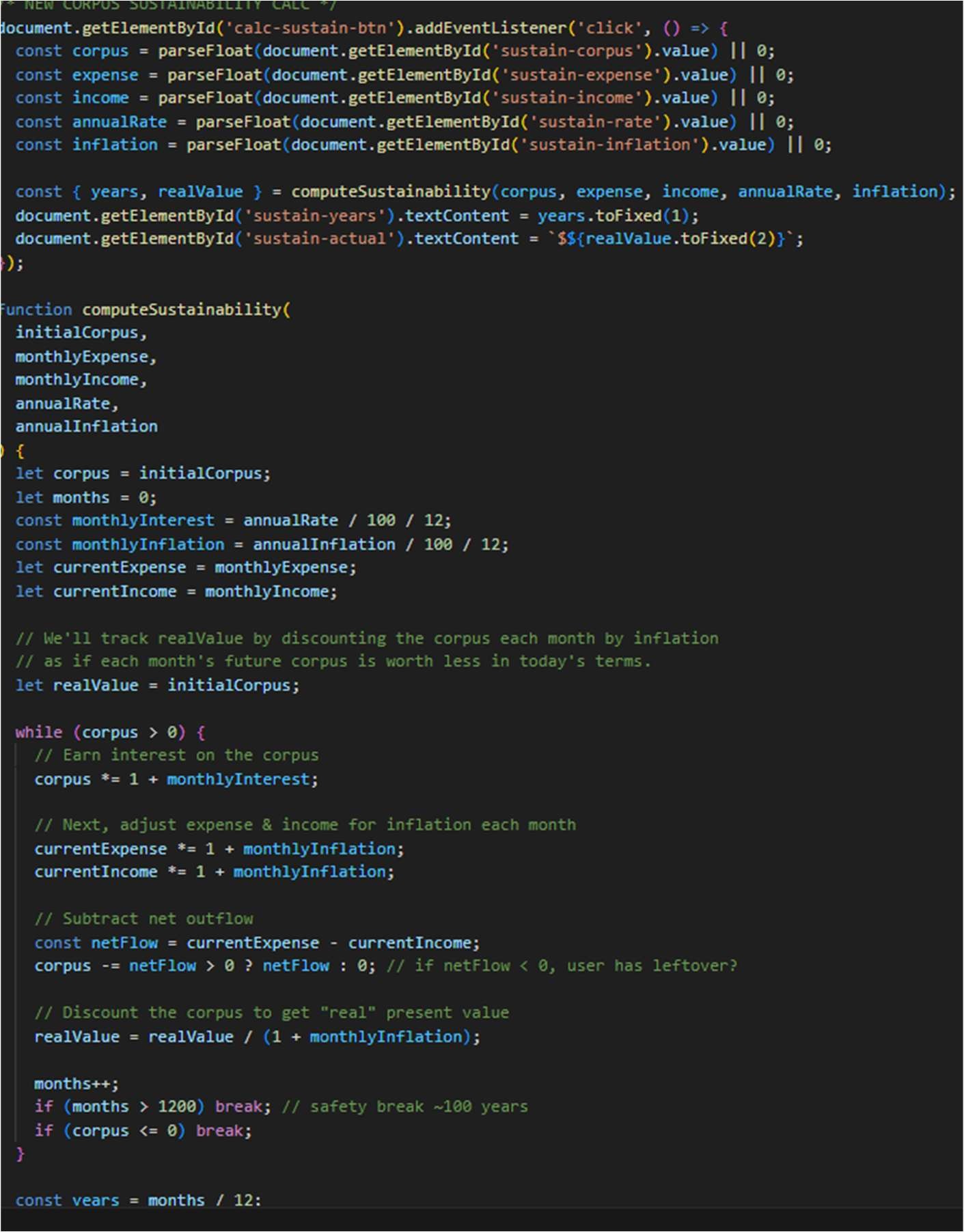
* + - Uses a formula to compute the total future value of monthly contributions with compound interest.

1. Monthly Withdrawal Calculator  handleCalcWithdraw():
   * + Takes a lump-sum principal and annual interest rate.
     + Uses computeMonthlyWithdrawal() to calculate how much you can withdraw monthly based on interest.

 computeMonthlyWithdrawal():

* + - Simple formula: principal × (annual rate ÷ 12)

Code snippet 7:



In this code:

Purpose:

This code calculates how long your savings (corpus) will last when:

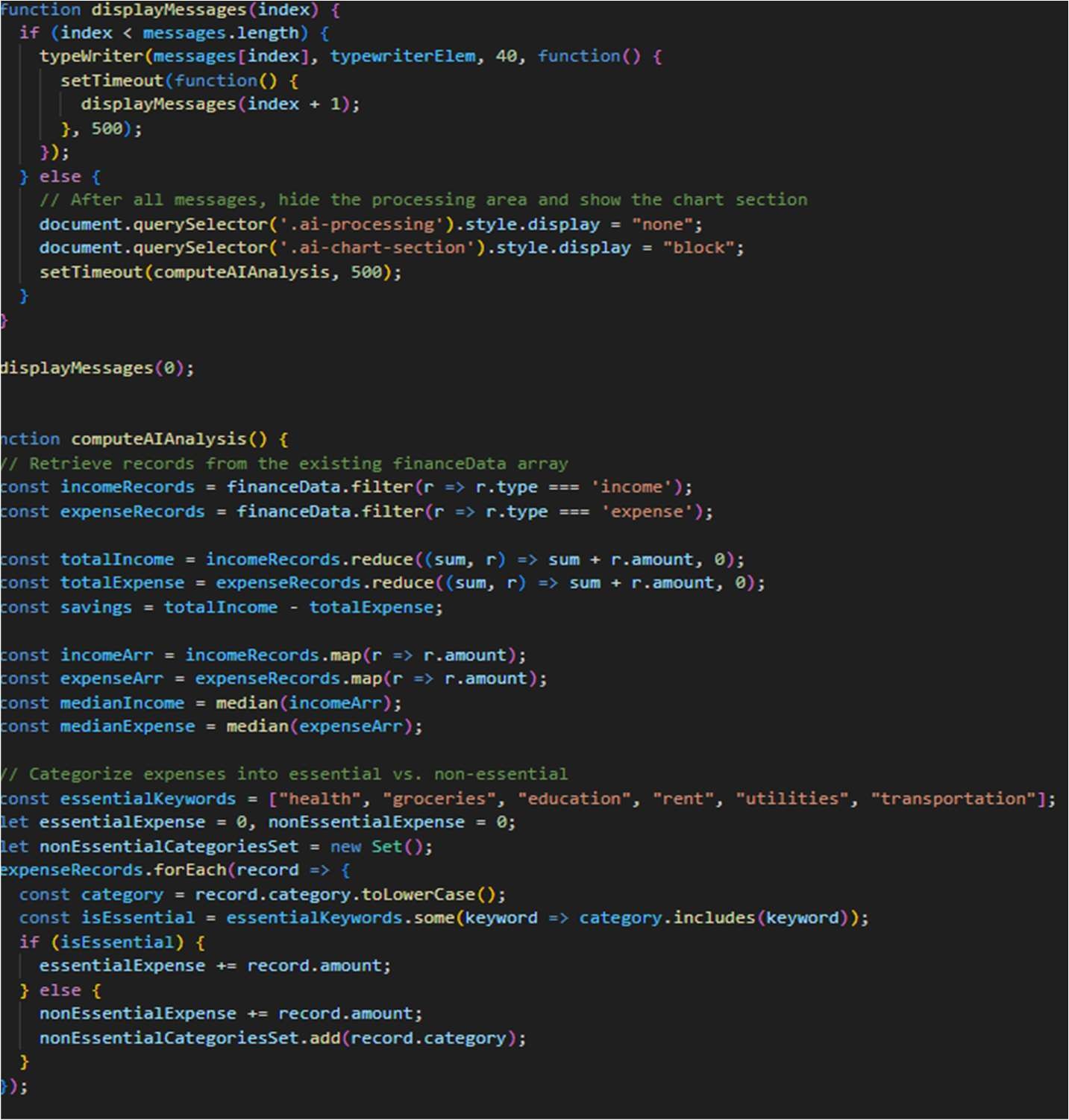
* + You have monthly expenses
  + Optional income
  + An interest rate (investment return)
  + And inflation

How it works:

* 1. On button click (calc-sustain-btn):
     + Gets values for:
       - Corpus (initial savings)
       - Monthly expense
       - Monthly income
       - Annual interest rate
       - Annual inflation
  2. Calls computeSustainability():
     + Calculates month-by-month:
       - Interest added to corpus
       - Inflation increase on expenses & income
       - Corpus reduced by net outflow (expense - income)
       - Real (inflation-adjusted) corpus tracked o Stops when corpus is depleted or after 100 years.

1. Displays:
   * + Years the money will last
     + Real value of what's left (after inflation)

Code snippet 8:



In this code:

Purpose:

This code handles an AI-driven analysis of a user's financial data and gives visual insights after showing messages with a typing effect.

displayMessages(index)

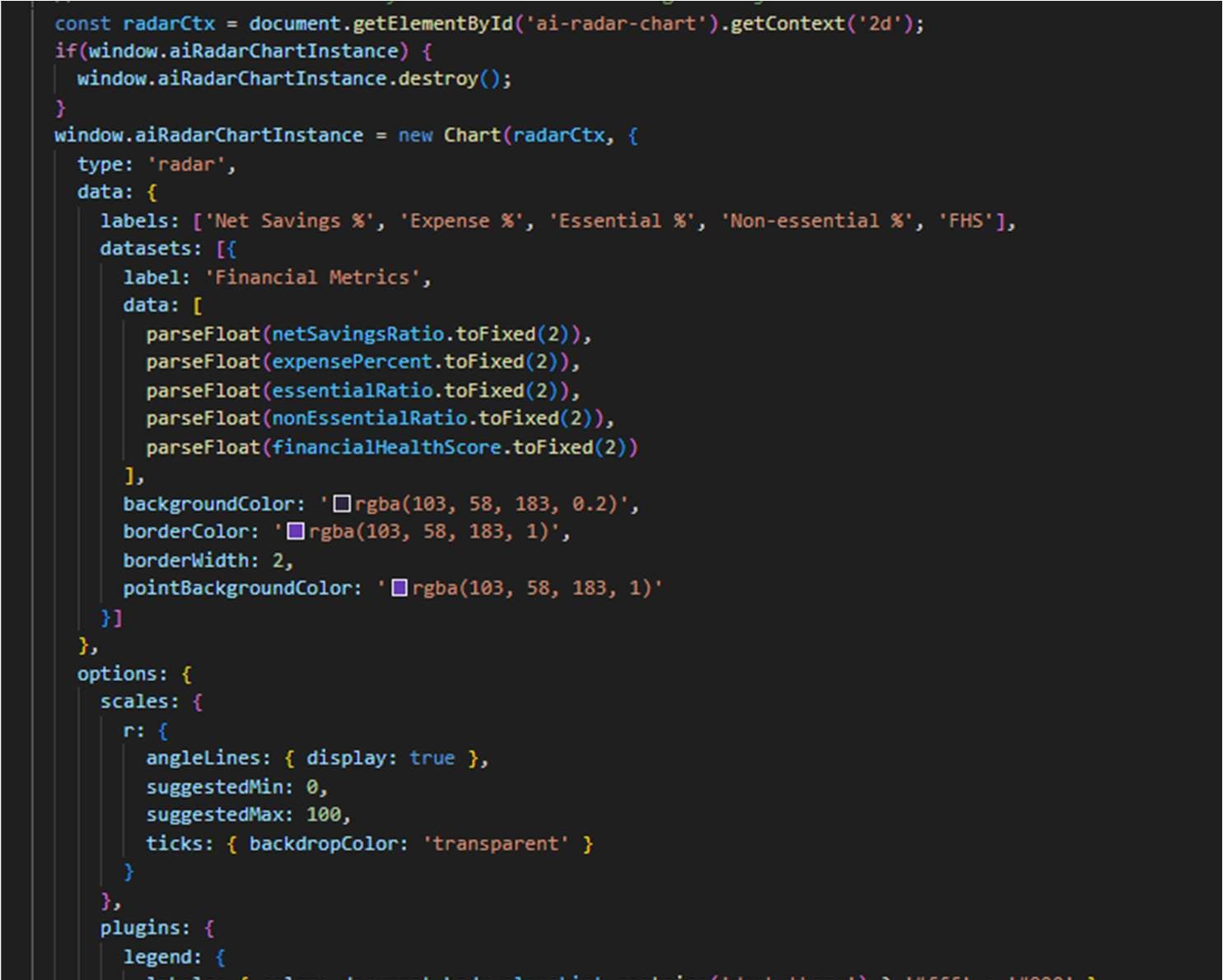
* + Shows a list of predefined messages one at a time using a typewriter animation.
  + After all messages are shown:
    - Hides the loading spinner/section. o Shows the AI chart section. o Calls computeAIAnalysis() to analyze data.

computeAIAnalysis() Analyzes the financeData array:

* 1. Classifies records:
     + Separates incomes and expenses.
  2. Calculates:
     + Total income o Total expenses o Savings (income - expenses) o Median income and median expense
  3. Categorizes expenses:
     + Tags each expense as essential (e.g., rent, groceries, health, etc.) or non-essential
     + Sums up totals for each
     + Collects a list of non-essential categories

Code snippet 9:

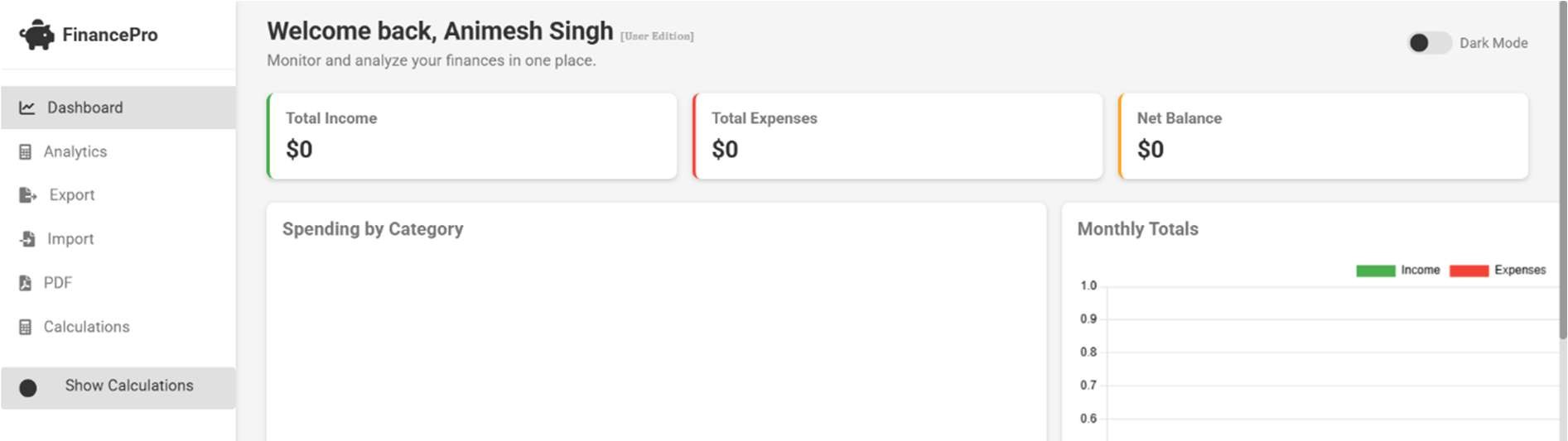
In this code:



* 1. Radar Chart: It creates or updates a radar chart showing financial metrics like Net Savings %, Expense %, and Financial Health Score using Chart.js.
  2. Median Function: It calculates the median of a list of numbers by sorting and finding the middle value.

In short, it visualizes financial data and calculates the median.

Result:



Top Section:

Welcome Message: “Welcome back, Animesh Singh" - This personalizes the dashboard for the logged-in user.

Tagline: "Monitor and analyze your finances in one place." - This clearly states the website's core purpose.

Key Financial Summary:

Total Income: $0 - Displays the total income recorded by the user. Currently showing zero, likely because no income has been entered yet.

Total Expenses: $0 - Shows the total expenses tracked by the user. Also currently zero.

Net Balance: $0 - Calculates and displays the difference between total income and total expenses. In this case, it's zero.

Dark Mode Toggle: A switch to enable or disable a dark color scheme for the interface.

Left Sidebar Navigation:

This provides access to different sections and features of the website:

Dashboard: Likely the current page, providing an overview of the user's finances.

Analytics: This section probably offers visual representations of financial data, such as charts and graphs, to help users understand their spending patterns, income sources, etc.

Export: Suggests the ability to export financial data in various formats (e.g., CSV, Excel).

import: Implies the functionality to import financial data from external sources (e.g., bank statements, other financial apps).

PDF:Might allow users to generate PDF reports of their financial data or summaries.

Calculations: Could offer financial calculators for budgeting, loan amortization, investment returns, etc.

Show Calculations: This button might toggle the visibility of a dedicated "Calculations" area on the dashboard or navigate to the "Calculations" page.

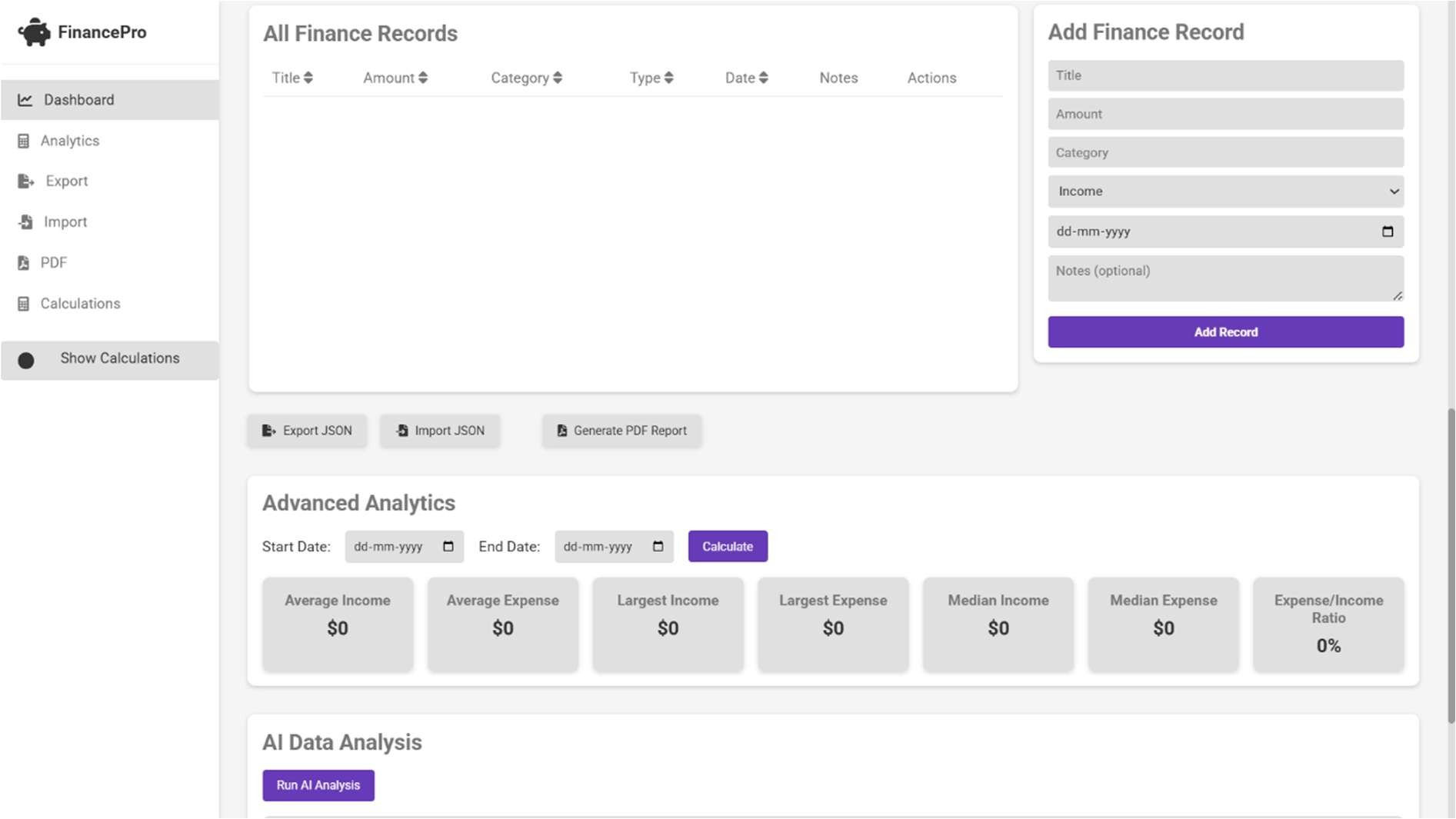
Main Content Area:

Spending by Category: This section will likely display a breakdown of the user's expenses categorized (e.g., food, transportation, utilities). It's currently empty, indicating no expense data has been categorized yet. This is often visualized using pie charts or bar graphs.

Monthly Totals: This appears to be a bar chart visualizing income and expenses on a monthly basis. The y-axis shows numerical values (likely amounts), and the x-axis would represent months. The legend indicates green for "Income" and red for "Expenses." Currently, there's no data displayed in the chart.

In summary, FinancePro is a web application designed to help users track, analyze, and manage their personal finances. It provides a dashboard overview, tools for categorization, visualization of spending and income trends, and options for data import/export and financial calculations.

To get a complete understanding, you'd need to see the other sections (Analytics, Export, Import, etc.) and interact with the website to understand its full capabilities.



Conclusion:

In conclusion, the development of this Personal Finance Management System represents a significant step towards empowering individuals to take control of their financial well-being. By providing a centralized platform for tracking income, expenses, and overall financial health, this system simplifies the often complex task of budget management.

The system's key features, including income and expense tracking, visualization tools, and reporting capabilities, offer users valuable insights into their spending habits and financial patterns. This enables them to make informed decisions, identify areas for improvement, and ultimately achieve their financial goals. The intuitive design and user-friendly interface further enhance the accessibility of these tools, making effective financial management attainable for a broader audience.

While the current version lays a strong foundation, future enhancements could significantly expand its functionality and user experience, transforming it into a comprehensive financial hub. These may include:

* Integration with bank accounts and other financial institutions: This would enable automated data input, reducing the need for manual entry and ensuring greater accuracy.
* Implementation of advanced budgeting techniques: Incorporating features like goal-based budgeting, forecasting, and envelope budgeting would provide users with more sophisticated tools for financial planning.
* Development of mobile applications: Creating dedicated mobile apps for iOS and Android platforms would allow users to manage their finances on the go, increasing convenience and accessibility.
* Incorporation of investment tracking and analysis tools: Expanding the system to include investment portfolio management, performance tracking, and basic analysis tools would cater to users with investment portfolios, further consolidating their financial management needs.
* Personalized financial advice and recommendations: Integrating AIpowered personalized advice and recommendations based on user data could provide valuable guidance and support, helping users make optimal financial decisions.
* Bill Payment and Management: Adding bill payment reminders and scheduling could streamline the process of managing recurring expenses. Ultimately, this Personal Finance Management System has the potential to be a valuable tool for anyone seeking to improve their financial literacy and achieve greater financial stability. By empowering users with the knowledge and tools to effectively manage their finances, this system can contribute to reduced financial stress, increased savings, and the achievement of long-term financial security.

Reference:

* www.investopedia.com
* www.github.com
* ChatGpt

)

223 | P a g