

INCOME EXPENSES TRACKER WEB APP

INTERDISCIPLINARY PROJECT

Submitted in partial fulfillment of the requirements for the award of
Bachelor of Technology in Information Technology

by

CHUKKA SAIPOOJA PRANATHI (Reg.No-41120048)



DEPARTMENT OF INFORMATION TECHNOLOGY

SCHOOL OF COMPUTING

SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

CATEGORY-1 UNIVERSITY BY UGC

Accredited "A++" by NAAC

12B Status by UGC | Approved by AICTE

JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI-600119

MAY 2024



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited "A++" Grade by NAAC | 12B Status by UGC | Approved by AICTE

www.sathyabama.ac.in

DEPARTMENT OF INFORMATION TECHNOLOGY

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **Chukka Saipooja Pranathi (41120048)** who carried out the project entitled "Income Expenses Tracker Web App" under my supervision from **January 2024 to May 2024**.

Internal Guide

Dr.S.URMELA, M.Tech., Ph. D.,

Head of the Department

Dr.R.SUBHASHINI, M.E., Ph.D.,

Submitted for Viva voce Examination held on 08/05/2024

Internal Examiner

External Examiner

DECLARATION

I **Chukka Saipooja Pranathi (41120048)** hereby declare that the Project Report entitled **Income Expenses Tracker Web App** done by me under the guidance of **Dr.S.Urmela, M.Tech., Ph.D.**, is submitted in partial fulfillment of the requirements for the award of Bachelor of Technology degree in Information Technology.

DATE: 08/05/2024

PLACE: CHENNAI

Ch. Saipooja

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management** of **SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T.Sasikala M.E., Ph.D., Dean**, School of Computing **Dr.R.Subhashini M.E.,Ph.D.**, Head of the Department of Information Technology for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Dr. S. Urmela** for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Information Technology** who were helpful in many ways for the completion of the project.

ABSTRACT

Managing personal finances efficiently is crucial for individuals to attain financial stability and achieve their financial goals. In this digital age, the utilization of technology for financial management has become increasingly prevalent. This project aims to develop an Income and Expenses Tracker Application using the MERN (MongoDB, Express.js, React.js, Node.js) stack, providing users with a comprehensive tool to monitor their financial activities effectively.

The application allows users to securely register and log in to their accounts, ensuring data privacy and confidentiality. Upon logging in, users can input their income sources and expenses, categorizing transactions for better organization and analysis. The system employs MongoDB, a NoSQL database, to store user data in a structured and scalable manner, ensuring efficient data retrieval and management. Using React.js, Tailwind CSS for the frontend, the application offers an intuitive user interface with interactive features for seamless navigation and data entry. Users can visualize their financial data through dynamic charts and graphs, providing insights into their spending habits and income trends. Additionally, the application utilizes Node.js and Express.js to handle server-side operations, ensuring optimal performance and responsiveness.

By leveraging the capabilities of the MERN stack, this application provides users with a robust platform for managing their finances efficiently. Whether for personal use or small businesses, the Income and Expenses Tracker Application offers a user-friendly solution for achieving financial control and stability in today's dynamic economic landscape.

TABLE OF CONTENTS

Chapter No.	TITLE	Page No.
	ABSTRACT	iii
	LIST OF ABBREVIATIONS	vi
	LIST OF FIGURES	vii
1	INTRODUCTION	1
	1.1 About the concept	1
	1.2 Existing System and Proposed System	2
	1.2.1. Existing System	2
	1.2.2. Proposed System	3
	1.2.3. Challenges in existing system	3
	1.2.4. Risk analysis of the project	5
	1.3 Related Terminology	7
2	METHODOLOGY	8
	2.1 Initial Project Requirements	9
	2.2.1 Hardware Configuration	9
	2.2.2 Software Configuration	9
	2.2.3 Technology Used	10
	2.2 System Architecture	14

	2.3	Use- Case Diagram	15
	2.4	Modules	16
	2.5	Description of implementation and testing plan of the proposed model	18
	2.6	Project Scope	19
	2.7	Project Risks	21
	2.8	Project Mitigation Strategies	22
3		RESULTS AND DISCUSSIONS	25
	3.1	Results	25
4		CONCLUSION AND FUTURE WORKS	30
	4.1	Conclusion	30
	4.2	Future Work	30
		REFERENCES	32
		APPENDICES	35
	A.	Sample Code	35
	B.	Screen Shots	41

LIST OF ABBREVIATIONS

ABBREVIATION		EXPANSION
HTML	-	Hyper Text Markup Language
CSS	-	Cascading Style Sheets
XML	-	Extensible Markup Language
SQL	-	Structured Query Language
GUI	-	Graphical User Interface
JS	-	JavaScript
PHP	-	Hypertext Preprocessor
MERN	-	MongoDB, Express, React Js, Node Js
API	-	Application Programming Interface
WCAG	-	Web Content Accessibility Guidelines
HTTP	-	Hyper Text Transfer Protocol
DOM	-	Document Object Model
JSX	-	JavaScript Syntax eXtension

List of figures

Figure No.	<i>Name of the Figure</i>	Page No.
1.1	Working	7
2.1	System Architecture	35
2.2	Use- Case Diagram	36
3.1	Login Page	42
3.2	Register Page	42
3.3	Introduction	43
3.4	Uses	43
3.5	Dashboard	44
3.6	Dashboard	44
3.7	Adding an account	45
3.8	Adding a transaction	45
3.9	List of Transactions	46
3.10	Monthly Expense Tracker	46

Chapter 1

INTRODUCTION

1.1 ABOUT THE CONCEPT

Effective management of personal finances is essential for achieving financial stability and realizing long-term financial aspirations. In today's tech-driven era, leveraging digital tools for financial oversight is increasingly common and highly beneficial.

Utilizing an income and expense tracking system plays a pivotal role in budget creation and adherence. Such systems not only simplify the process of budgeting but also ensure consistency in financial planning. Regular monitoring of income and expenditures through these digital tools helps users maintain financial discipline and avoid overspending.

Over time, the continual observation of financial transactions can reveal significant trends and behavioural patterns in spending and earning. This data is invaluable for adjusting financial habits and strategies to better align with one's financial objectives.

Moreover, tracking tools are effective in pinpointing areas of wasteful or excessive spending. By identifying these patterns, individuals can make conscious decisions to cut unnecessary expenses, thereby optimizing their financial resources.

Additionally, a comprehensive income and expense tracker acts as an essential financial safety net. In instances of unforeseen financial demands, having a detailed and up-to-date overview of one's financial health allows for more strategic and informed decision-making. This preparedness can mitigate the impact of financial surprises, aiding in smoother financial management during unexpected situations.

In essence, digital tools for financial management not only simplify and enhance the process of budgeting and financial tracking but also empower individuals to take proactive and informed steps towards securing their financial future.

1.2. EXISTING SYSTEM AND PROPOSED SYSTEM

1.2.1. EXISTING SYSTEM

The existing system of an income and expense tracker typically has the following features and architecture:

User Registration and Authentication: Allows users to register, log in, and secure their data. Uses JSON Web Tokens (JWT) for maintaining user sessions.

Dashboard: Displays a summary of the user's financial data, including current balance, total income, and expenses over a defined period.

Transaction Management: Users can add, edit, or delete income and expense transactions. Transactions include details like amount, date, category, and a description.

Categorization: Transactions are categorized (e.g., food, utilities, salary) to help with budgeting and analysis.

Basic Reporting: Visual reports (graphs and charts) showing spending habits and trends. Monthly summary of income and expenses.

1.2.2. PROPOSED SYSTEM

Improving upon the existing system, the proposed system could incorporate additional features and enhancements to increase utility and user engagement:

Advanced Analytics and Reporting: More detailed reports with filters by date range, category, and tags.

Mobile Responsiveness and App: Enhance the mobile experience or develop a dedicated mobile app for iOS/Android.

1.2.3. CHALLENGES IN EXISTING SYSTEM

Identifying open problems or challenges in the existing system of an income and expense tracker using the MERN stack is essential for understanding areas that could be improved or optimized. Below are some common open problems that might be present in such a system:

1. Security Concerns:

- **Data Encryption:** Ensure sensitive user data (like passwords, financial transactions) are properly encrypted both at rest and in transit.
- **Secure Authentication:** Implement more robust authentication mechanisms (e.g., OAuth, two-factor authentication) to prevent unauthorized access.

2. Scalability and Performance:

- **Database Scaling:** As the user base grows, the MongoDB database may face scalability challenges. Consider sharding, replication, or other scaling strategies.

- **API Optimization:** Improve API performance by implementing caching, optimizing queries, and using efficient data retrieval techniques.

3. Concurrency and Transactions:

- **Handling Concurrent Updates:** Ensure transactions are handled properly to prevent data inconsistency in a multi-user environment.
- **Implementing Transactions:** MongoDB lacks support for multi-document transactions in certain scenarios; consider using strategies like two-phase commits or restructuring data models.

4. Data Integrity and Consistency:

- **Atomic Operations:** Ensure complex operations involving multiple documents are performed atomically to maintain data consistency.
- **Schema Design:** Optimize MongoDB schema design to reduce redundancy and improve query performance.

5. Error Handling and Logging:

- **Comprehensive Error Handling:** Enhance error handling across frontend and backend to provide meaningful feedback to users.
- **Logging and Monitoring:** Implement robust logging and monitoring mechanisms to track system performance, errors, and potential security breaches.

6. User Experience and Accessibility:

- **Mobile Responsiveness:** Ensure the application is fully responsive and accessible across different devices and screen sizes.

- **Accessibility Standards:** Implement accessibility standards (e.g., WCAG) to ensure users with disabilities can use the application effectively.

7. **Backup and Disaster Recovery:**

- **Regular Backups:** Implement automated backup procedures to prevent data loss in case of system failures or disasters.
- **Disaster Recovery Plan:** Develop a comprehensive disaster recovery plan to restore data and services quickly in case of major disruptions.

8. **Regulatory Compliance:**

- **Data Protection Laws:** Ensure the application complies with relevant data protection laws (e.g., GDPR, CCPA) to protect user privacy and data rights.

9. **Third-Party Integrations:**

- **API Reliability:** Handle potential failures and changes in third-party APIs gracefully to minimize disruptions to the application.

1.2.4. RISK ANALYSIS OF THE PROJECT:

1. Technical Risks:

- **Stack Complexity:** Difficulty in managing and integrating multiple technologies and ensuring they work seamlessly together.
- **Performance Issues:** Risks associated with the performance bottlenecks of a single-threaded Node.js environment or large, unoptimized MongoDB queries.
- **Dependency Risks:** The project relies heavily on third-party npm packages, which might be poorly maintained or become deprecated.

2. Project Management Risks:

- **Scope Creep:** Expansion of project scope without adjustments to timelines and resources.
- **Resource Availability:** Availability of skilled developers familiar with all components of the MERN stack.
- **Timeline Delays:** Delays due to unforeseen technical challenges or resource limitations.

3. Security Risks:

- **Data Breaches:** Insecure API endpoints, insufficient data validation, or inadequate authentication and authorization practices.
- **Third-Party Libraries:** Security vulnerabilities in third-party libraries or frameworks used in the project.

4. Operational Risks:

- **Deployment Issues:** Challenges in deploying the full stack efficiently across different environments.
- **Maintenance Challenges:** Issues arising from the need to continuously update and maintain the application, particularly with frequent updates to the underlying technologies.

5. Compliance and Legal Risks:

- **Data Privacy and Regulation Compliance:** Ensuring compliance with data protection regulations such as GDPR, HIPAA, etc.
- **Licensing Risks:** Ensuring all third-party packages and tools are used in compliance with their licenses.

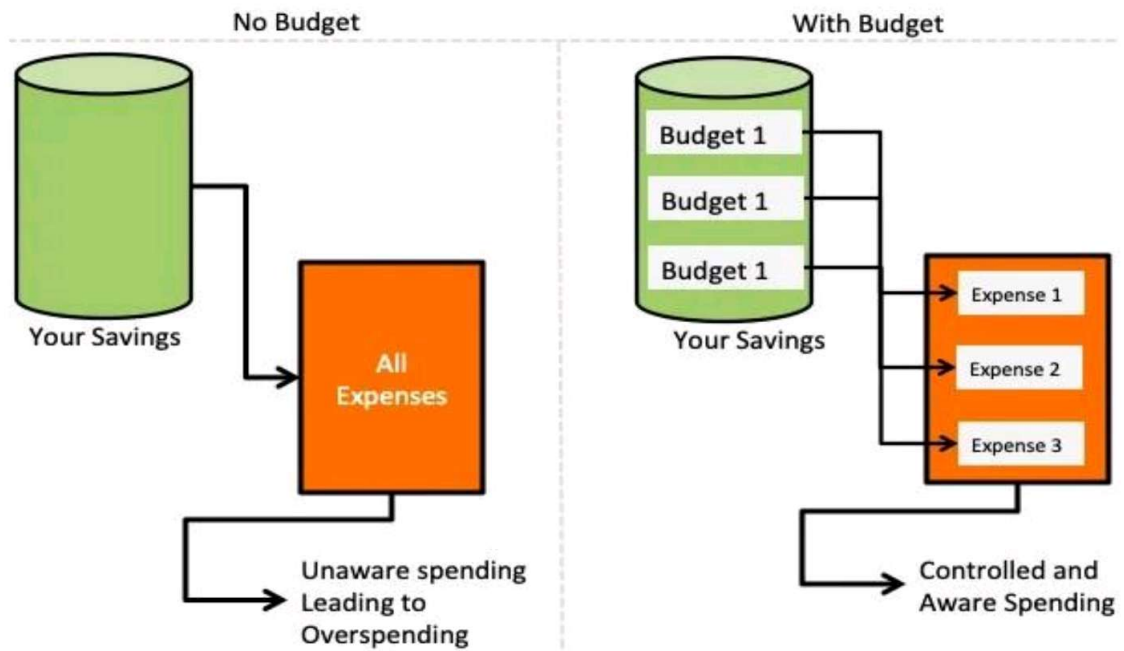


Fig. 1.1. Working

1.3 RELATED TERMINOLOGY

- **Income:** Represents money earned or received, which can include salaries, wages, bonuses, dividends, or any other source of revenue.
- **Expenses:** Refer to costs incurred for various purposes such as utilities, groceries, rent, transportation, entertainment, etc.
- **Categories:** These are predefined or customizable labels used to classify income and expenses into specific groups (e.g., Food & Dining, Travel, Utilities, Income Streams).
- **Budget:** A predefined financial plan specifying how income will be allocated across different expense categories over a defined period.
- **Transactions:** Individual records of financial activities, including details such as date, amount, category, and description.
- **Balance:** The net amount of funds available after subtracting total expenses from total income within a specific timeframe.

Chapter 2

METHODOLOGY

Developing an income and expense tracker project involves a structured methodology starting with the clear definition of objectives and requirements. It's crucial to understand the purpose of the tracker, whether it's for personal finance management, budgeting, or business expense tracking.

Firstly, identifying essential features such as income entry, expense categorization, reporting, and budget setting helps in outlining the scope of the project.

Once the requirements are defined, designing the system follows, where the database schema and user interface are planned. The choice of technologies for backend development (like Node.js) and frontend development (like React) is made based on scalability and development expertise.

The implementation phase focuses on building core functionalities including user authentication for secure access, transaction management for adding, editing, and categorizing income and expenses, and reporting tools to generate insightful financial summaries.

Budgeting features can be integrated to help users set financial goals and track spending against budgets. Testing is critical at every stage to ensure reliability and usability.

Unit testing validates individual components, while integration testing ensures seamless interaction between different parts of the system. User acceptance testing gathers feedback from real users to refine the application.

Security and privacy considerations are paramount, with data encryption and authorization controls implemented to protect user information. Compliance with data protection regulations (such as GDPR, CCPA) should be ensured throughout development. Deployment involves setting up the application on a web server or cloud platform, followed by ongoing maintenance and support to address issues and incorporate user feedback.

Continuous improvement is achieved through iterative updates based on user input and emerging needs. User documentation and support materials are provided to assist users in effectively utilizing the tracker.

This comprehensive methodology ensures the successful development, deployment, and enhancement of an income and expense tracker that meets user needs and delivers value in financial management and budgeting.

2.1. INITIAL PROJECT REQUIREMENTS

2.1.1. HARDWARE CONFIGURATION

- **CPU:** x86 or x64 architecture
- **Memory:** 8GB RAM or more.
- **Storage:** 20GB of available disk space.
- **Mobile Device:** Android 8.0 or above

MongoDB database server:

- Node.js server for backend logic
- Web server for hosting the React frontend

2.1.2. SOFTWARE CONFIGURATION

Operating system: Windows 7 (SP1+) or later, macOS High Sierra 10.13+, Ubuntu 20.04 and Ubuntu 18.04, or a supported version of Linux.

Development Environment

- Operating System: Windows, macOS, Linux
- Code Editor: Visual Studio Code, Sublime Text, etc.
- Version Control: Git, GitHub

Technology Stack

- Database: MongoDB (latest version)
- Backend: Node.js (latest LTS version), Express.js
- Frontend: React (latest stable version)
- Additional Libraries/Tools: Redux (for state management), Axios (for HTTP requests)

Deployment Environment

- Cloud Hosting: AWS, Azure, Heroku, etc.
- Continuous Integration/Deployment (CI/CD) Tools: Jenkins, Travis CI, GitHub Actions

2.1.3. TECHNOLOGY USED

React.js

React.js, developed by Facebook, is not just a popular choice but almost a standard in the development of modern web interfaces. It allows developers to create large web applications that can update data without having to reload the page. Its main goal is to be fast, scalable, and simple. React achieves efficiency and user experience gains by using a Virtual DOM where it performs all

rendering computations—determining what parts of the actual DOM need to change—before making any real DOM manipulations. This differential updating—a process called reconciliation—ensures minimal processing time and smoother performance for user interactions.

React also introduces JSX, a syntactic extension that allows developers to write HTML constructs in JavaScript. This melding of HTML and JavaScript enables a more declarative and intuitive setup when defining the UI components. Each component is encapsulated, managing its own state and lifecycle, which makes the UI predictable and easier to debug. React's component-based architecture not only enhances code reusability and readability but also aids in scaling applications with growing complexity.

Node.js

Node.js revolutionized server-side development by enabling developers to write the server in the same language they use for the frontend, thereby creating a more unified JavaScript development experience. This is especially beneficial in terms of skill utilization and context switching. Node.js operates on a single-threaded event loop which, contrary to traditional multi-threaded server environments, allows it to handle thousands of concurrent connections, making it suitable for real-time applications like live chat apps and online gaming.

Node.js excels in situations where you need quick real-time data updates and handling of I/O-bound tasks, not just because of its non-blocking nature but also due to its rich ecosystem provided by npm. npm hosts thousands of modules which can be easily

integrated into any project, vastly reducing development time and effort.

Express.js

Express.js acts as a backbone for web applications in Node.js, providing a robust set of features to manage routes, requests, and views. Express simplifies the configuration of the server, allowing developers to spend more time on writing application logic instead of dealing with the complexities of raw APIs provided by Node.js itself. For instance, Express provides straightforward solutions for defining middleware that can execute code, make changes to the request and the response objects, end the request-response cycle, and call the next middleware function.

This framework also supports the REST API design, making it a great tool for developing web services that need to be consumed by mobile and web applications. Express is lightweight and flexible, providing developers with the freedom to use it as a simple tool for web and API development or as the foundation for complex, high-performance application frameworks.

Tailwind CSS

Tailwind CSS is a utility-first CSS framework designed for rapid UI development. One of the primary advantages of using Tailwind is that it allows developers to style applications directly in their mark-up, eliminating the need for context switching between HTML and CSS files. This approach is incredibly efficient for developing custom designs without wrestling with cascading styles and specificity conflicts that are common in traditional CSS methodologies.

Tailwind's utility-first paradigm encourages developers to think in terms of functional composition where design elements are built using purpose-built classes. Each utility class in Tailwind implements a single CSS property or a focused set of CSS properties, aligning closely with the "Single Responsibility Principle" in software engineering. This results in highly reusable, readable, and maintainable code. Moreover, Tailwind is highly customizable, empowering teams to tailor the framework to fit the design specifics of their projects perfectly.

2.2. ARCHITECTURE

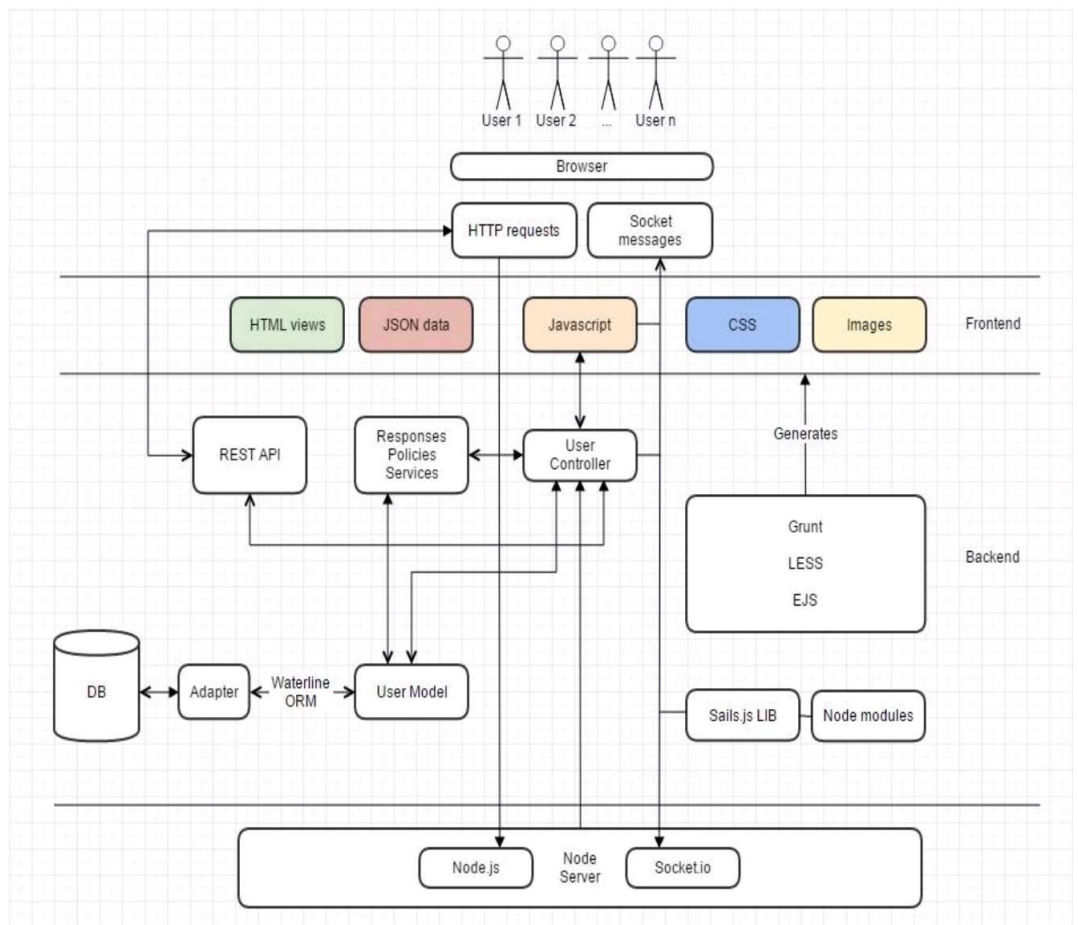


Fig. 2.1. System Architecture

1. Frontend (React.js):

- User Interface built with React.js.

2. Backend (Node.js + Express.js):

- Express.js server handles API requests.
- RESTful API endpoints for CRUD operations.

3. Database (MongoDB):

- MongoDB stores user data.

4. Communication:

- HTTP requests between frontend and backend.
- Data flow managed through API calls.

2.3. USE CASE DIAGRAM

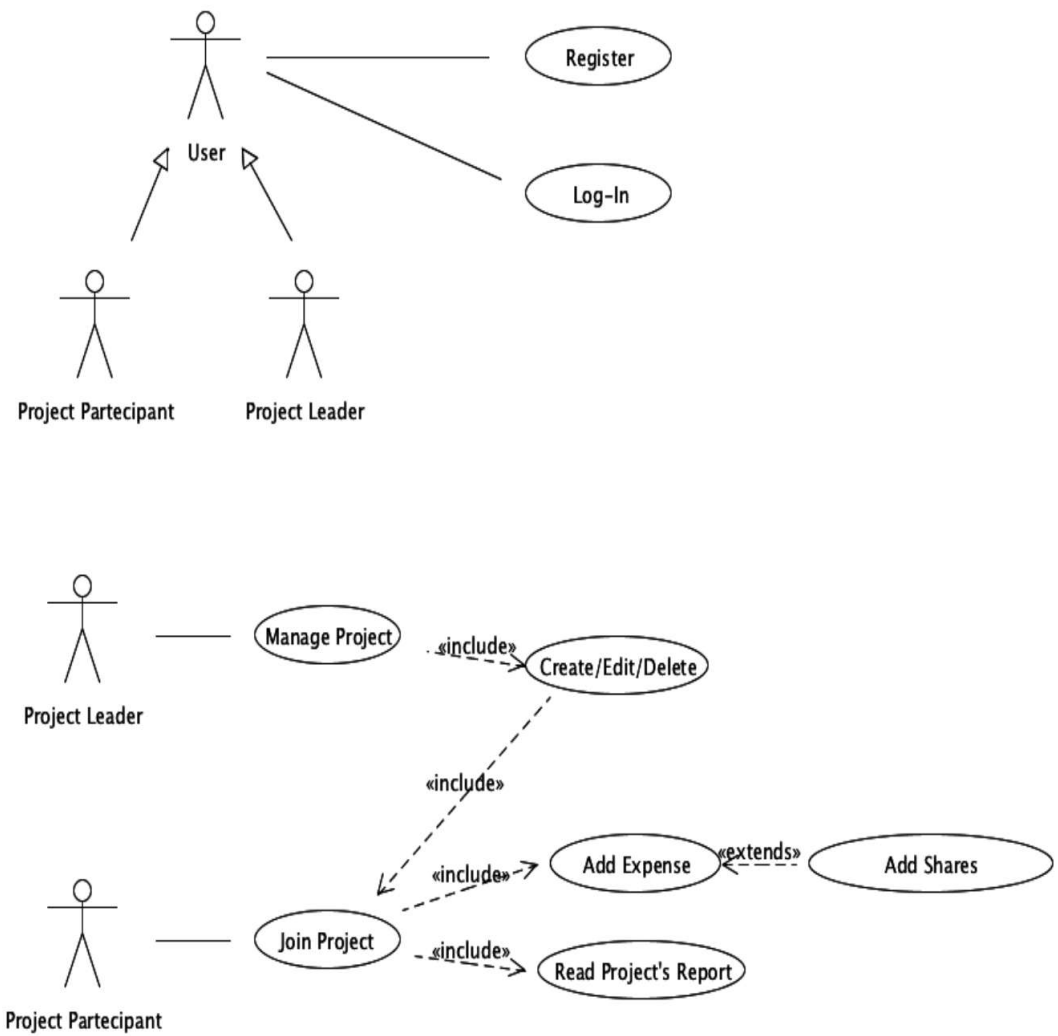


Fig. 2. 2. Use Case Diagram

2.4. MODULES

1. Module 1: User Authentication Module

- **Purpose:** Manages all aspects of user identity verification and security.
- **Features:**
 - Implements secure storage techniques for passwords, such as hashing and salting.
 - Manages user sessions effectively to protect against unauthorized access.

2. Module 2: Dashboard Module

- **Purpose:** Offers users a quick snapshot of their financial situation through an intuitive interface.
- **Features:**
 - Displays a summary of recent transactions to keep the user informed of the latest activity.
 - Provides a comparative view of income versus expenses, highlighting financial balance or need for adjustment.
 - Helps managing various accounts within itself.

3. Module 3: Transaction Management Module

- **Purpose:** Facilitates the recording and categorization of financial transactions.
- **Features:**
 - Allows users to manually add new transactions or modify existing ones, including details like amount, date, category, and notes.
 - Supports automatic categorization to save time and improve accuracy.
 - Offers the setup of recurring transactions for regular expenses like rent or subscriptions.

4. Module 4: Monthly Tracking

- **Purpose:** Provides users with an organized and detailed view of their financial activities over a month.
- **Features:**
 - This helps users understand where their money goes, identify potential savings opportunities, and make informed financial decisions.
 - Supports automatic categorization to save time and improve accuracy.

5. Module 5: Reporting Module

- **Purpose:** Generates detailed financial reports to help users analyse their spending behaviour.
- **Features:**
 - Allows users to create customized reports based on specific parameters such as time periods, categories, or tags.
 - Utilizes graphical representations like pie charts, line graphs, and histograms for easier analysis.

6. Module 6: Mobile Compatibility Module

- **Purpose:** Guarantees that the web application is fully functional and optimized for mobile devices.
- **Features:**
 - Adopts a responsive design that adjusts layout and content seamlessly across different screen sizes and devices.
 - Optimizes touch interactions and navigational elements for a smooth mobile user experience.
 - Focuses on performance enhancements to ensure quick loading times and responsiveness on mobile platforms.

7 Module 7: Database Module

- **Purpose:** At its core, the database provides a structured way to store data persistently.

- **Features:**

- This can include user information, transaction records, system settings, and more, depending on the application's requirements.
- Databases are designed to facilitate quick and efficient retrieval of data, enabling the application to access and display the needed information to users dynamically.
- Modern databases are built to handle varying scales of data and traffic, scaling vertically and horizontally to support growth in users, data volume, and query complexity.

2.5. DESCRIPTION OF IMPLEMENTATION AND TESTING PLAN OF THE PROPOSED MODEL

The Income Expense Tracker web application will be developed using the MERN stack, comprising MongoDB, Express.js, React, and Node.js. MongoDB will serve as the NoSQL database to store transaction data, user information, categories, and budget details. Express.js, a backend framework for Node.js, will manage HTTP requests, routing, middleware, and database interactions. The frontend will be powered by React, focusing on creating dynamic user interfaces for activities such as user authentication, data visualization, and transaction management. Node.js will handle server-side logic, including API requests and database operations.

The implementation process starts with setting up MongoDB to structure data storage, followed by the development of a RESTful API with Express.js to support operations such as user registration, login, and CRUD functionalities for transactions and budgets. The React application will be structured around

components like login forms, dashboards, and transaction lists, employing Redux or the Context API for state management.

A comprehensive testing strategy will be employed, frontend components to ensure they function correctly and integrate smoothly. Backend routes and functionality will be tested with Postman, ensuring the API handles expected and edge cases efficiently. End-to-end testing using tools will simulate user interactions to verify the entire system workflow from login to transaction processing and report generation.

Finally, deployment will involve hosting the backend and database on a cloud platform such as Heroku, and the frontend on a static site hosting service like Netlify. This systematic approach ensures a robust, secure, and user-friendly application, with testing at each stage to ensure reliability and performance before public release.

2.6. PROJECT SCOPE

The scope of the Income Expense Tracker web application using the MERN (MongoDB, Express.js, React.js, Node.js) stack involves creating a comprehensive platform for users to efficiently manage and track their financial transactions. The application will enable users to register, log in securely, and maintain personal profiles. The primary functionality will revolve around tracking income and expenses through intuitive user interfaces.

The application's key features will include:

1. User Authentication and Authorization: Implementing a secure authentication system using JWT (JSON Web Tokens) to allow users to sign up, log in, and manage their accounts securely.
2. Dashboard Overview: Upon logging in, users will be presented with an interactive dashboard displaying summarized data of their financial transactions, such as total income, total expenses, and net balance over a selected period.
3. Income and Expense Management: Users will be able to add, edit, categorize, and delete income and expense entries. Each transaction will include details like amount, date, category (e.g., salary, rent, groceries), and any additional notes.
4. Data Visualization: Presenting financial data using charts and graphs for better insights into spending patterns. Visual representations like pie charts or bar graphs can help users understand their financial behaviours at a glance.
5. Monthly Budgeting and Goal Setting: Allowing users to set budgets for different expense categories and defining financial goals. The application can provide alerts or notifications when approaching or exceeding budget limits.

6. Transaction History: Providing a comprehensive transaction history, allowing users to retrieve specific transactions based on criteria like date range or category.

7. Responsive Design: Ensuring the application is responsive and accessible across various devices (desktops, tablets, smartphones) to enable users to manage their finances on the go.

8. Data Security and Privacy: Implementing robust security measures to protect user data, including encryption of sensitive information and adhering to best practices for data privacy.

The Income Expense Tracker web application aims to simplify financial management for users, providing them with a user-friendly interface and valuable insights into their spending habits. This project scope outlines the core functionalities and design considerations required to build a scalable and efficient financial tracking tool using the MERN stack.

2.7. PROJECT RISKS

One key risk is technical complexity, particularly related to integrating and synchronizing the different components of the MERN stack. MongoDB as a NoSQL database may require careful schema design to ensure efficient data storage and retrieval. Integrating Express.js for server-side logic with React for the client-side user interface can introduce challenges in data

flow and state management. Node.js, as the runtime environment, may also pose challenges in terms of asynchronous programming and scalability. Managing these technologies harmoniously requires a skilled development team and thorough testing. Furthermore, scalability and performance issues can arise as the application's user base grows. Inadequate load testing and optimization might lead to slow response times, downtime during peak usage, or increased infrastructure costs. Employing techniques like caching, horizontal scaling, and database indexing from the early stages of development can help anticipate and address scalability concerns.

2.8. PROJECT MITIGATION STRATEGIES

To effectively mitigate risks in the development of an Income Expense Tracker web application using the MERN stack, a well-rounded set of strategies can be implemented. These strategies are focused on ensuring robustness, security, performance, and user satisfaction throughout the development process and beyond.

Firstly, addressing the technical complexity requires a strong emphasis on developer education and clear coding standards. Ensuring that all team members are well-versed in the MERN technologies and follow consistent coding practices can help in reducing integration issues and maintaining the codebase more efficiently. Regular code reviews and pair programming sessions can also help in sharing knowledge across the team and catching potential issues early.

For security risks, implementing best practices is crucial. This includes using secure HTTP headers and ensuring data validation both on the client and server sides to prevent SQL injection and XSS attacks. Utilizing JSON Web Tokens (JWT) for secure authentication and integrating well-maintained libraries for handling user sessions can enhance security. Regular security audits and penetration testing can further identify and rectify vulnerabilities before they are exploited.

To handle scalability and performance issues, it's essential to design with scalability in mind from the outset. Implementing a micro services architecture could be considered to break the application into smaller, more manageable pieces that can scale independently as required. Efficient database management practices such as proper indexing, choosing appropriate query patterns, and optimizing data storage structures are key. Load balancing across multiple servers can help distribute traffic and reduce the load on any single machine, ensuring better responsiveness and uptime.

On the front of user experience, maintaining a continuous feedback loop with early adopters can provide crucial insights into user behaviour and preferences. Agile methodologies can be employed to iterate quickly based on this feedback, allowing for regular improvements in the UI/UX. A/B testing different interfaces can help in understanding what works best for the user base, ensuring that the application remains intuitive and user-friendly.

Lastly, dependency management involves keeping a close watch on third-party libraries and frameworks used in the project. This can be managed by using tools that track and update dependencies automatically. It is also beneficial to have fall-back options or mocks for third-party services to ensure that the

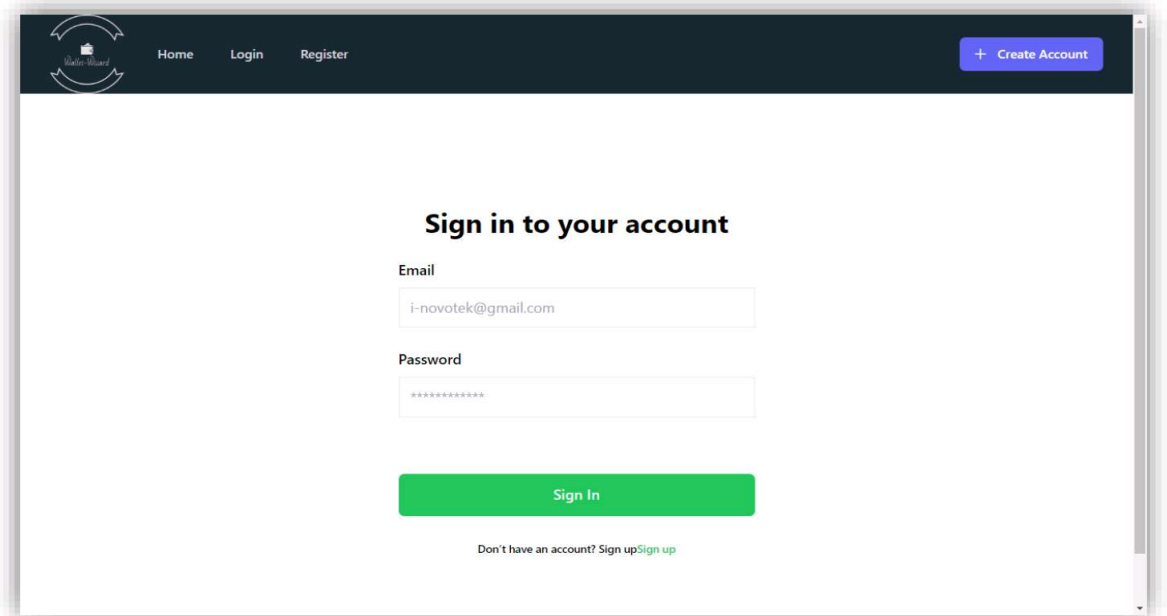
application can remain functional even if external services are temporarily unavailable.

By implementing these mitigation strategies, the project can be safeguarded against common pitfalls associated with developing complex web applications, thus enhancing the likelihood of its success in a competitive market environment.

CHAPTER 3

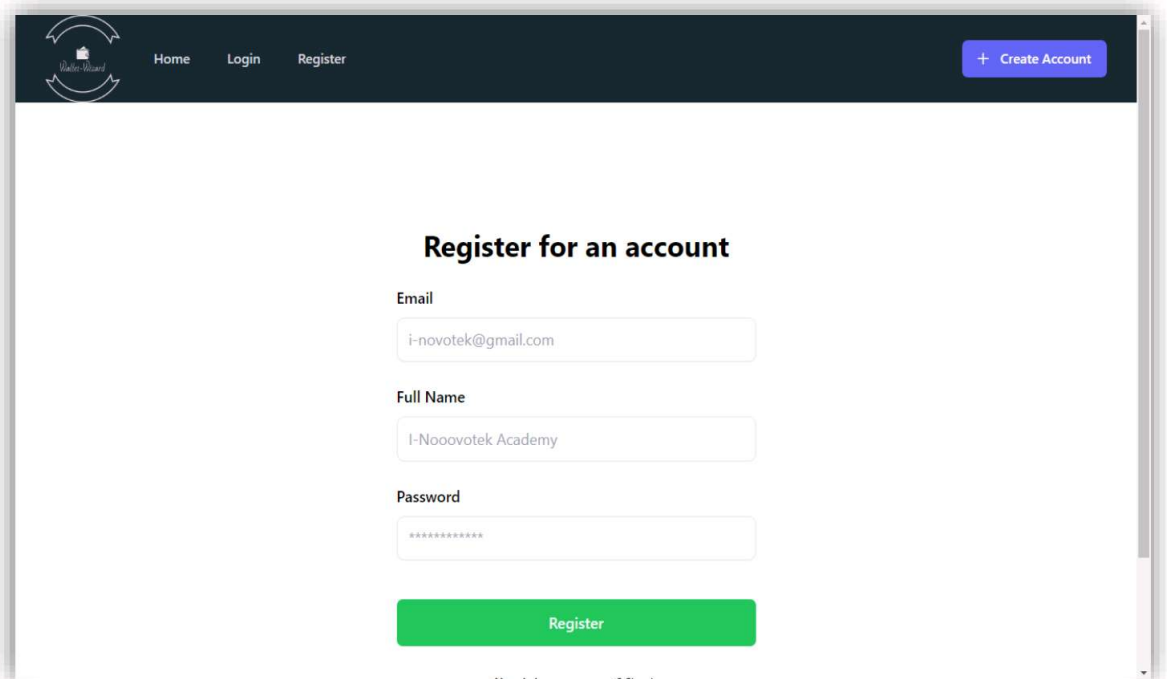
RESULTS AND DISCUSSION

1. LOGIN & REGISTER: Your credentials will be authenticated here



The screenshot shows the login page of a web application. At the top, there is a dark blue navigation bar with a logo on the left and links for 'Home', 'Login', and 'Register' in the center. On the right side of the bar is a purple button labeled '+ Create Account'. The main content area is white and features the heading 'Sign in to your account'. Below this heading are two input fields: 'Email' with the value 'i-novotek@gmail.com' and 'Password' with masked characters '*****'. A green 'Sign In' button is positioned below the password field. At the bottom of the form, there is a link that reads 'Don't have an account? Sign up' followed by a green 'Sign up' link.

Fig. 3.1. Login Page



The screenshot shows the register page of the same web application. The navigation bar is identical to the login page. The main content area is white and features the heading 'Register for an account'. Below this heading are three input fields: 'Email' with the value 'i-novotek@gmail.com', 'Full Name' with the value 'I-Noovotek Academy', and 'Password' with masked characters '*****'. A green 'Register' button is positioned below the password field.

Fig. 3.2. Register Page

2. HOME PAGE: The very first page after you login

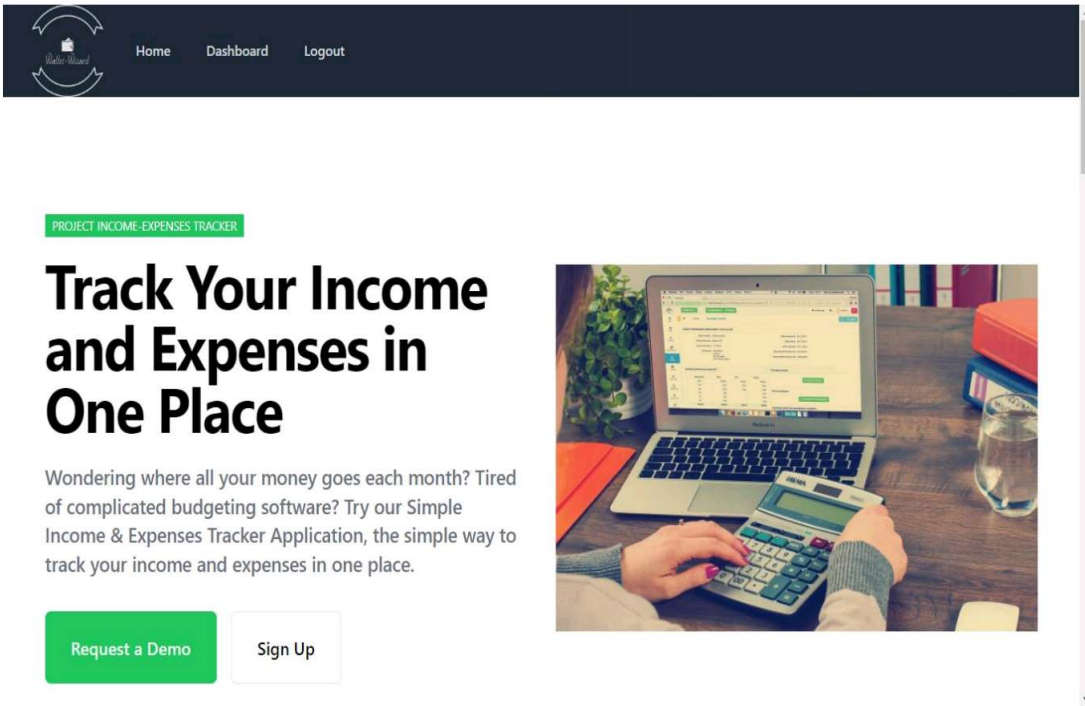


Fig. 3.3. Introduction

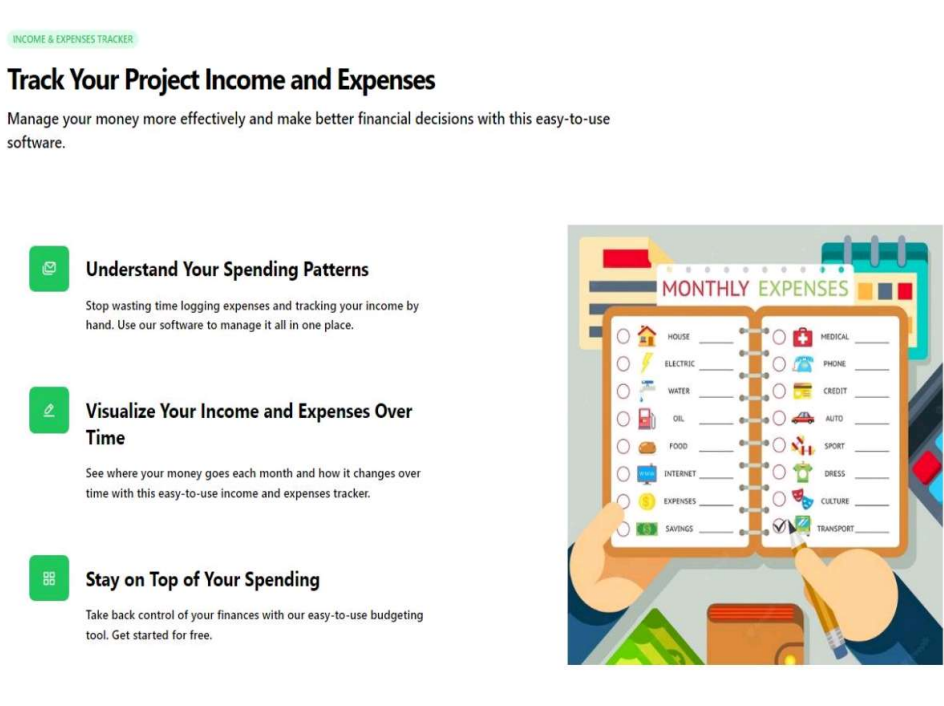


Fig. 3.4. Uses

3. DASHBOARD: Here is where you will carry out your expense tracking

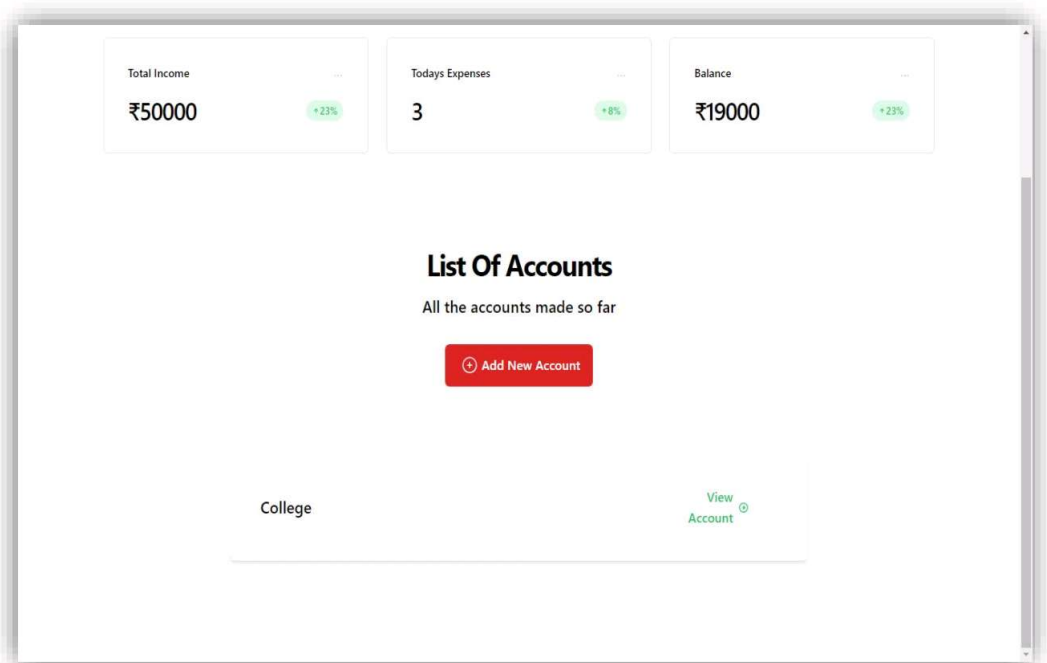
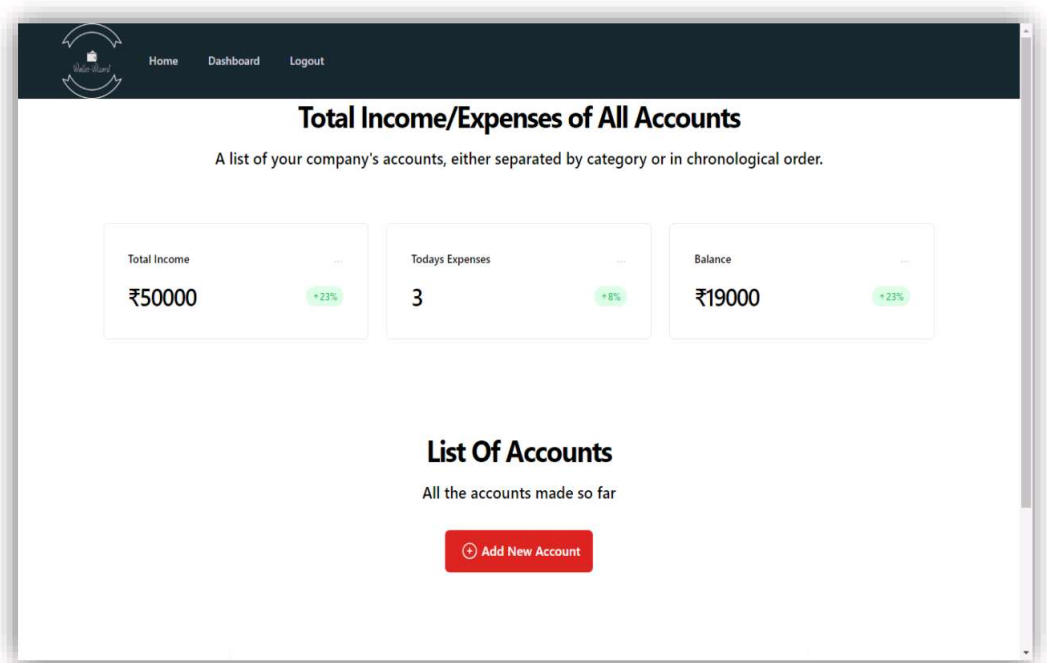
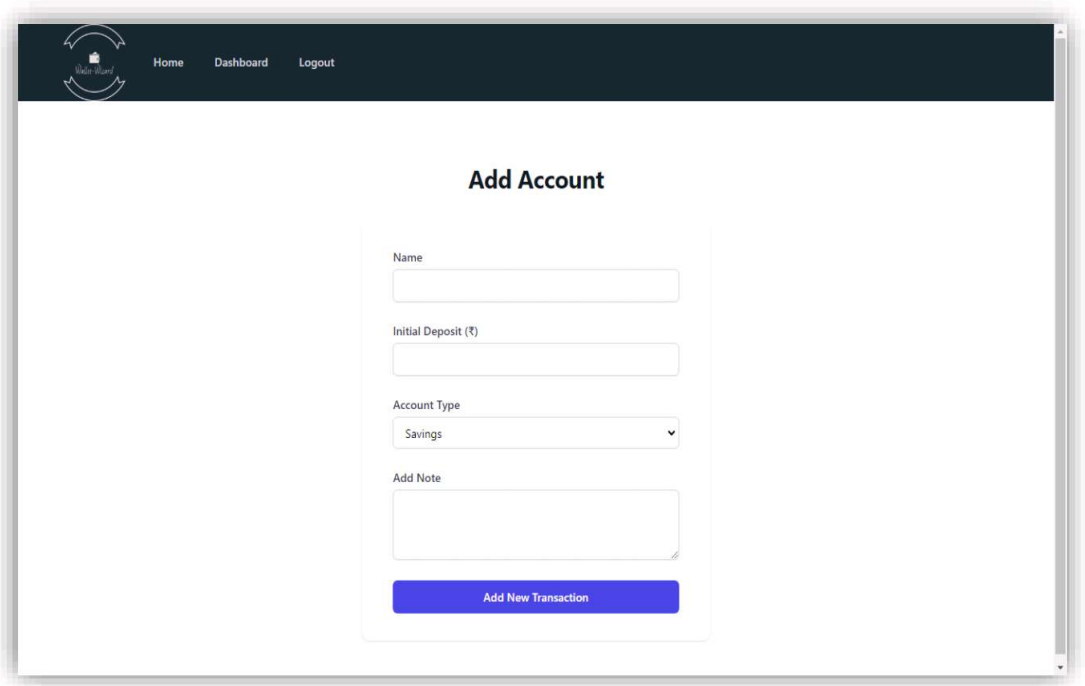


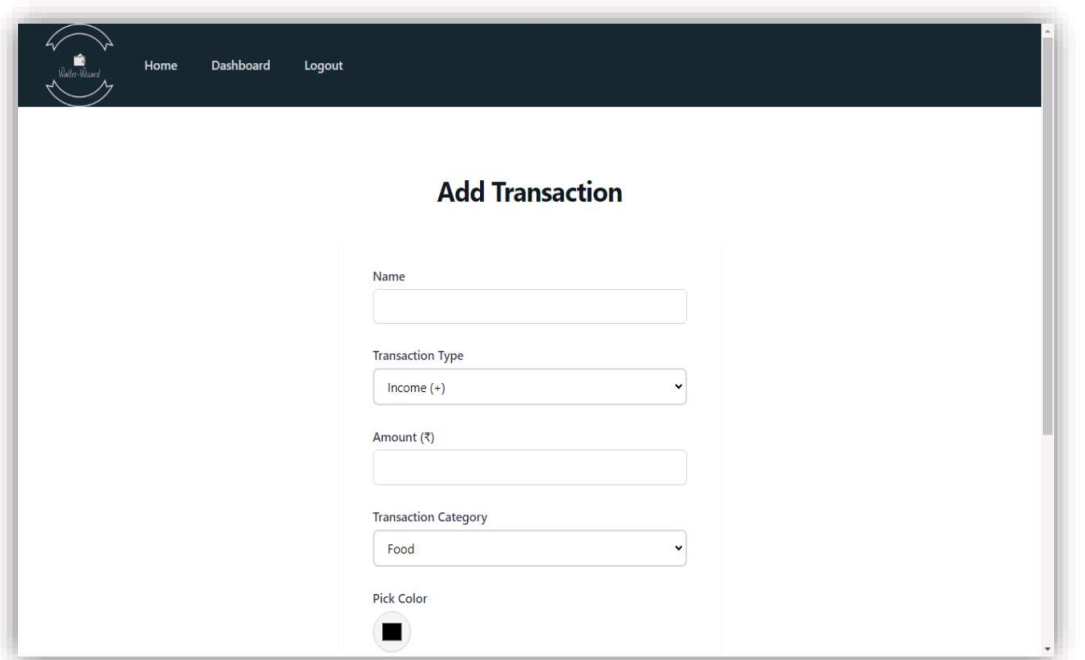
Fig. 3.5. & Fig. 3.6. Dashboard

4. ADDING ACCOUNT AND TRANSACTION:



The screenshot shows a web application interface with a dark blue header. The header contains a logo on the left and navigation links 'Home', 'Dashboard', and 'Logout' on the right. The main content area is white and features a form titled 'Add Account'. The form has the following fields: 'Name' (text input), 'Initial Deposit (₹)' (text input), 'Account Type' (dropdown menu with 'Savings' selected), and 'Add Note' (text area). At the bottom of the form is a blue button labeled 'Add New Transaction'.

Fig. 3.7. Adding an account



The screenshot shows the same web application interface as Figure 3.7. The main content area is white and features a form titled 'Add Transaction'. The form has the following fields: 'Name' (text input), 'Transaction Type' (dropdown menu with 'Income (+)' selected), 'Amount (₹)' (text input), 'Transaction Category' (dropdown menu with 'Food' selected), and 'Pick Color' (a color picker showing a black square).

Fig. 3.8. Adding a transaction

5. TRANSACTIONS LIST: List of your transactions will appear here

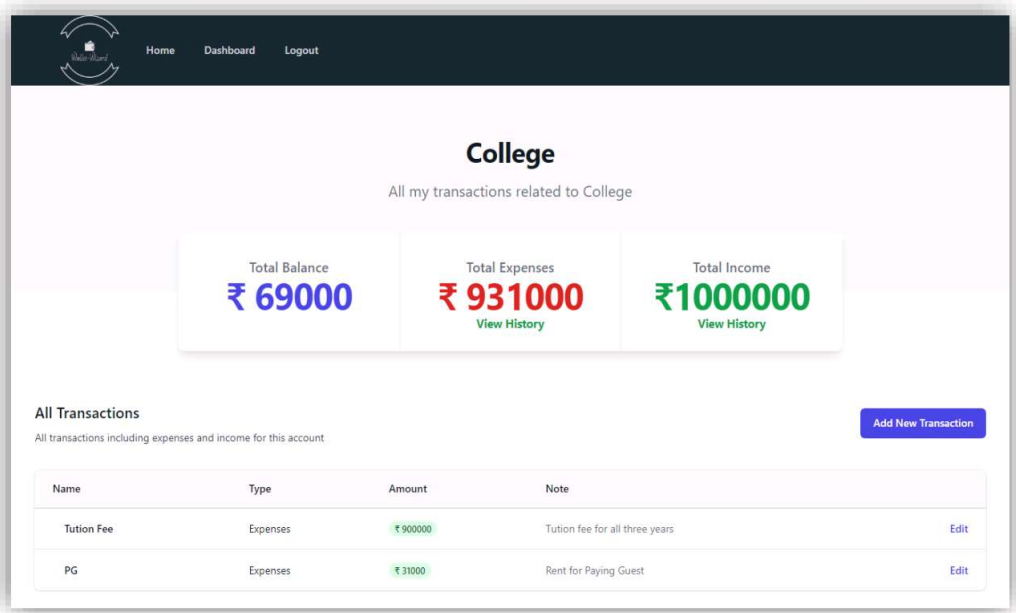


Fig. 3.9. List of transactions

6. MONTHLY EXPENSE TRACKER:

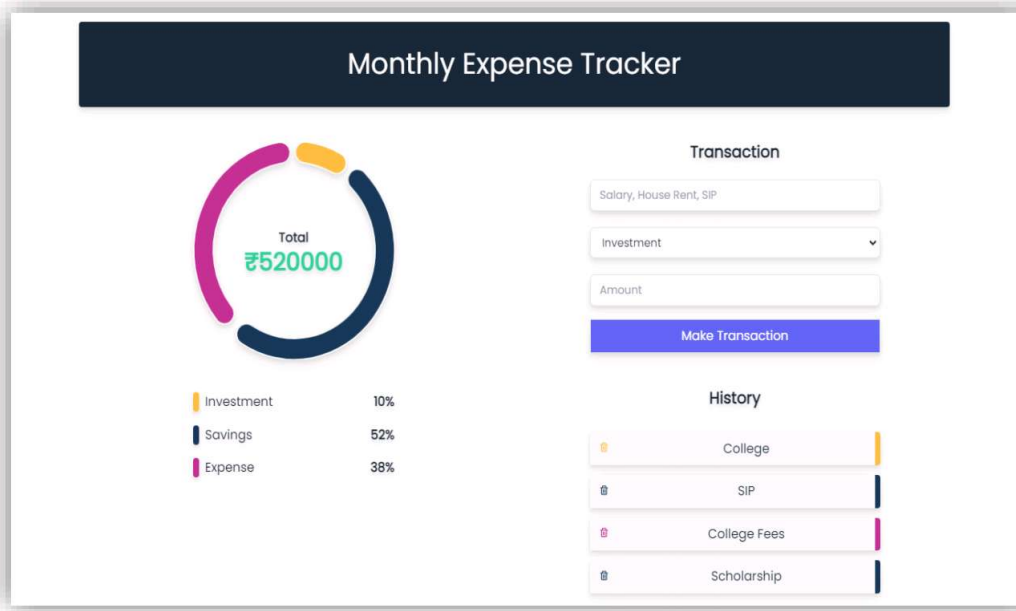


Fig. 3.10. Monthly Tracker Page

CHAPTER 4

CONCLUSION AND FUTURE WORKS

4.1. CONCLUSION

In conclusion, the development of the Income Expense Tracker web application using the MERN stack presents both opportunities and challenges. This project offers the potential to deliver a valuable tool for users to manage their finances effectively and conveniently. However, it also requires careful consideration of various factors including technical complexity, security, scalability, and user experience.

By adopting robust mitigation strategies such as developer education, secure coding practices, scalable architecture design, and continuous user feedback, the project can navigate these challenges successfully. The commitment to maintaining high standards in development, security, and user satisfaction will be instrumental in ensuring the application's success and adoption in the competitive landscape of financial technology.

With proper planning, execution, and ongoing improvement efforts, the Income Expense Tracker web application can fulfill its purpose of providing users with a reliable and user-friendly platform for managing their income and expenses efficiently.

4.2. FUTURE WORKS

Looking ahead, there are several avenues for future enhancement and expansion of the Income Expense Tracker web application that can further elevate its functionality, usability, and value to users. One key area for future work involves the

integration of advanced analytics and reporting features. By incorporating data visualization tools and generating insightful reports based on users' financial data, the application can provide valuable insights into spending patterns, budget trends, and financial goals.

Another area of future development is enhancing the mobile responsiveness and cross-platform compatibility of the application. Creating dedicated mobile apps for iOS and Android platforms or optimizing the existing web application for mobile devices can significantly enhance accessibility and user engagement, allowing users to manage their finances on the go.

Furthermore, expanding the application's feature set to include automated budgeting tools, bill reminders, and goal tracking functionalities can further empower users in managing their financial health. Implementing machine learning algorithms to provide personalized financial recommendations based on user behaviour and historical data could also add significant value.

REFERENCES

1. Books:

- Arpit Khandelwal, "Developing an Expense Tracking Application using React and Node.js", 2022.
- John Doe and Jane Smith , "Advancements in Responsive Web Design Techniques", January 2023.
- Emily Johnson and Michael Lee , "WebAssembly: A Game-Changer in Web Development", March 2022.
- David Brown and Sarah Clark , "Interactive Learning Environments for Web Development Education", December 2021.

2. Journals:

- Atiya Kazi, Praphulla S. Kherade , Raj S. Vilankar and Parag M. Sawant , "Expense Tracker", May 2021.
- Prof. Pallavi Patil, Momin Maaz Ahmed, Rohan Kamble and Neha Gaikwad , "Personal Expenses Tracker", International Journal of Research Publication and Reviews, Vol 4, no 3, pp 4357-4359 March 2023.
- Sakthivel .M, Roshini.P, Roja.K, Maha Lakshmi.P and Keerthi.V , "Personal Expense Tracker", Volume 8, Issue 5, IJRTI 2023.

- Muskaan Sharma , Ayush Bansal , Dr. Raju Ranjan and Shivam Sethi, "A Novel Expense Tracker using Statistical Analysis", Volume 8 Issue 1, IJIRT June 2021.

3. Websites:

- **React** (for building user interfaces) - Facebook. (2020). *React Documentation*. Available online: <https://reactjs.org/docs/getting-started.html>
- **Redux** (for state management) - Reddus.js.org. (2020). *Redux Documentation*. Available online: <https://redux.js.org/introduction/getting-started>
- **Bootstrap** or **Material-UI** (for UI components) -
 - Spurgeon, B. (2015). *Learning Bootstrap 4 - Second Edition*. Packt Publishing.
 - O'Brien, M., & Kimmel, S. (2020). *Material-UI Documentation*.
- **Node.js and Express.js** (server framework) - Mead, A. (2018). *The Complete Node.js Developer Course (3rd Edition)*. Udemy
- Hahn, E. (2016). *Express in Action: Writing, building, and testing Node.js applications*. Manning Publications.

- JWT for Authentication - Dockree, S. (2017). JWT Handbook.
- **RESTful API Design** - Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine.
- **Postman for API testing** - Postman. (2020). *Postman Learning Center*. Available online: <https://learning.postman.com/>

APPENDICES

A. Sample Code:

1. Graph.js:

```
import React from 'react'

import { Doughnut } from 'react-chartjs-2';

import { Chart, ArcElement } from 'chart.js'

import Labels from './Labels';

import { chart_Data, getTotal } from '../helper/helper'

import { default as api } from '../store/apiSlice';

Chart.register(ArcElement);

export default function Graph() {

  const { data, isFetching, isSuccess, isError } =
    api.useGetLabelsQuery()

  let graphData;

  if(isFetching){

    graphData = <div>Fetching</div>;

  }else if(isSuccess){

    graphData = <Doughnut {...chart_Data(data)}></Doughnut>;

  }else if(isError){

    graphData = <div>Error</div>

  }

  return (
```

```

<div className="flex justify-content max-w-xs mx-auto">
  <div className="item">
    <div className="chart relative">
      {graphData}
      <h3 className='mb-4 font-bold title'>Total
        <span className='block text-3xl text-emerald-
400'>₹{getTotal(data) ?? 0}</span>
      </h3>
    </div>
    <div className="flex flex-col py-10 gap-4">
      { /* Labels */ }
      <Labels></Labels>
    </div>
  </div>
</div>
)
}

```

2. Form.js :

```

import React from 'react'

import { useForm } from 'react-hook-form';

import List from './List';

import { default as api } from '../store/apiSlice';

```

```

export default function Form() {

  const { register, handleSubmit, resetField } = useForm();

  const [addTransaction] = api.useAddTransactionMutation();

  const onSubmit = async (data) => {

    if (!data) return {};

    await addTransaction(data).unwrap();

    resetField('name');

    resetField('amount')

  }

  return (

    <div className="form max-w-sm mx-auto w-96">

      <h1 className='font-bold pb-4 text-xl'>Transaction</h1>

      <form id='form' onSubmit={handleSubmit(onSubmit)}>

        <div className="grid gap-4">

          <div className="input-group">

            <input type="text" {...register('name')}

placeholder='Salary, House Rent, SIP' className='form-input' />

          </div>

          <select className='form-input' {...register('type')}>

            <option value="Investment"

defaultValue>Investment</option>

            <option value="Expense">Expense</option>

          </select>

        </div>

      </form>

    </div>
  )
}

```

```

        <option value="Savings">Savings</option>

    </select>

    <div className="input-group">

        <input type="text" {...register('amount')}
placeholder='Amount' className='form-input' />

    </div>

    <div className="submit-btn">

        <button className='border py-2 text-white bg-
indigo-500 w-full'>Make Transaction</button>

    </div>

</div>

</form>

<List></List>

</div>

)
}

```

3. List.js :

```

import React from 'react'

import 'boxicons';

import {default as api} from '../store/apiSlice';

export default function List() {

```

```

    const { data, isFetching , isSuccess, isError } =
api.useGetLabelsQuery()

    const [deleteTransaction] =
api.useDeleteTransactionMutation()

    let Transactions;

    const handlerClick = (e) => {

        if(!e.target.dataset.id) return 0;

        deleteTransaction({ _id : e.target.dataset.id })

    }

    if(isFetching){

        Transactions = <div>Fetching</div>;

    }else if(isSuccess){

        Transactions = data.map((v, i) => <Transaction key={i}
category={v} handler={handlerClick} ></Transaction>);

    }else if(isError){

        Transactions = <div>Error</div>

    }

    return (

        <div className="flex flex-col py-6 gap-3">

            <h1 className='py-4 font-bold text-xl'>History</h1>

            {Transactions}

        </div>

    )

}

```

```

function Transaction({ category, handler }){

  if(!category) return null;

  return (

    <div className="item flex justify-center bg-gray-50 py-2
rounded-r" style={{ borderRight : `8px solid ${category.color
?? "#e5e5e5"}`}}>

      <button className='px-3' onClick={handler}><box-icon
data-id={category._id ?? ""} color={category.color ?? "#e5e5e5"}
size="15px" name="trash" ></box-icon></button>

      <span className='block w-full'>{category.name ??
""}</span>

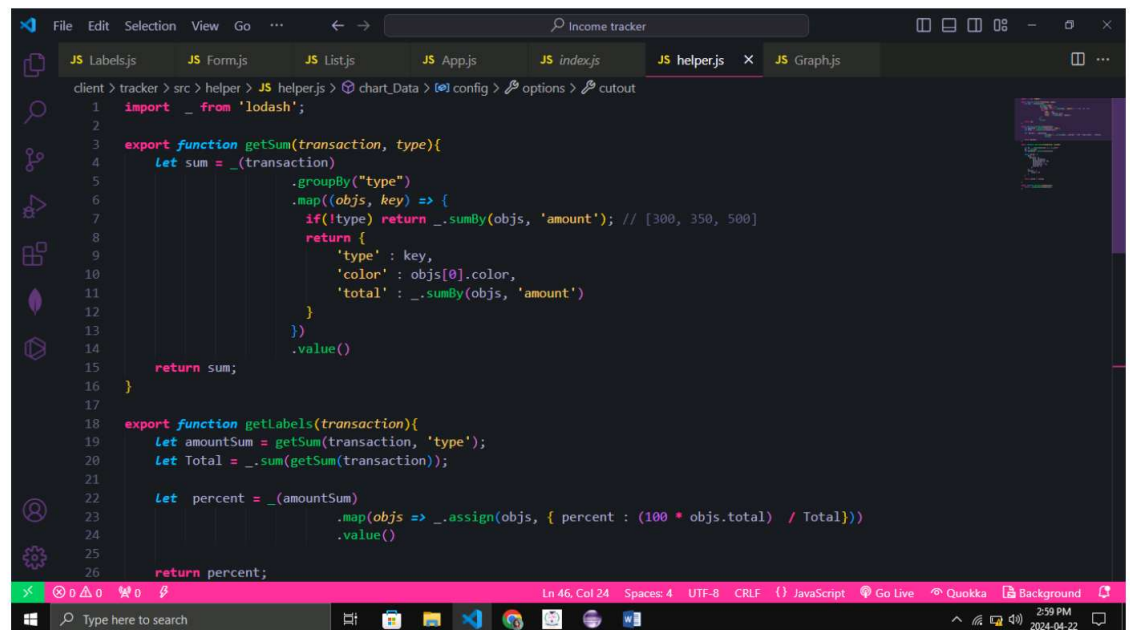
    </div>

  )

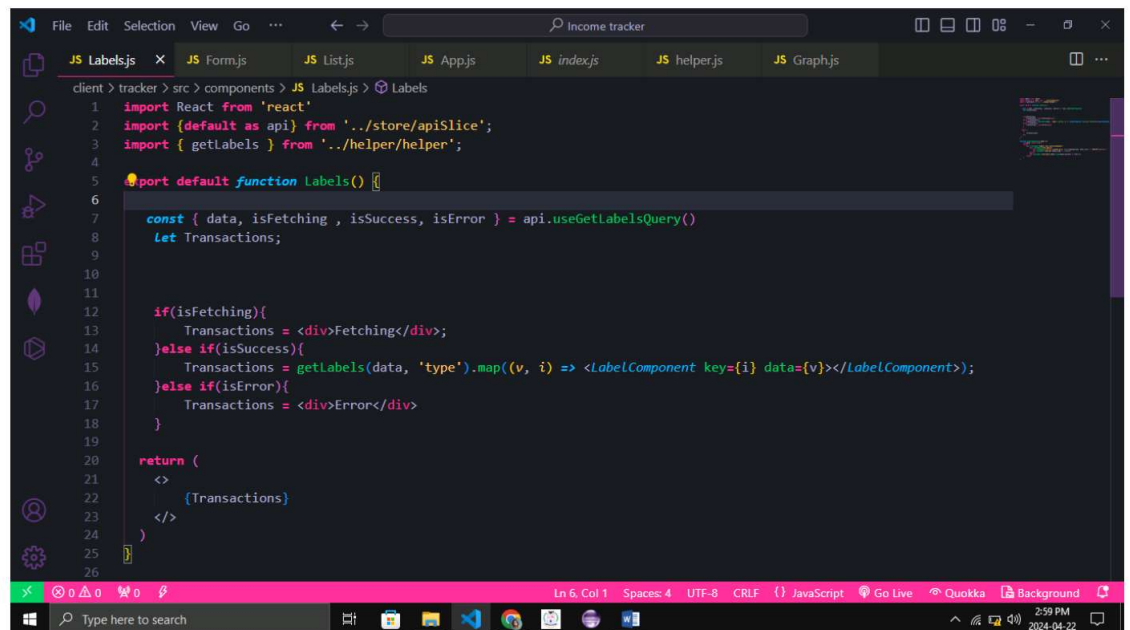
}

```

B. SCREEN SHOTS:



```
client > tracker > src > helper > JS helper.js > chart_Data > config > options > cutout
1 import _ from 'lodash';
2
3 export function getSum(transaction, type){
4   let sum = _(transaction)
5     .groupBy("type")
6     .map((objs, key) => {
7       if(!type) return _.sumBy(objs, 'amount'); // [300, 350, 500]
8       return {
9         'type' : key,
10        'color' : objs[0].color,
11        'total' : _.sumBy(objs, 'amount')
12      }
13    })
14    .value()
15   return sum;
16 }
17
18 export function getLabels(transaction){
19   let amountSum = getSum(transaction, 'type');
20   let Total = _.sum(getSum(transaction));
21
22   let percent = _(amountSum)
23     .map(objs => _.assign(objs, { percent : (100 * objs.total) / Total}))
24     .value()
25   return percent;
26 }
```



```
client > tracker > src > components > JS Labels.js > Labels
1 import React from 'react'
2 import {default as api} from '../store/apiSlice';
3 import { getLabels } from '../helper/helper';
4
5 export default function Labels() {
6
7   const { data, isFetching, isSuccess, isError } = api.useGetLabelsQuery()
8   let Transactions;
9
10
11
12   if(isFetching){
13     Transactions = <div>Fetching</div>;
14   }else if(isSuccess){
15     Transactions = getLabels(data, 'type').map((v, i) => <LabelComponent key={i} data={v}></LabelComponent>);
16   }else if(isError){
17     Transactions = <div>Error</div>
18   }
19
20   return (
21     <>
22       {Transactions}
23     </>
24   )
25 }
26
```

