EXPENSE TRACKER

A PROJECT REPORT
for
Mini-Project 2 (ID201B)
Session (2024-25)

Submitted by
Aanchal
(202410116100002)
Akansha Tyagi
(202410116100014)
Akansha Tomar
(202410116100013)
Deepu Kumari
(202410116100057)

Submitted in partial fulfilment of the
Requirements for the Degree of

MASTER OF COMPUTER APPLICATION

Under the Supervision of
Dr. Vipin Kumar
Associate Professor

Submitted to

DEPARTMENT OF COMPUTER APPLICATIONS
KIET Group of Institutions, Ghaziabad
Uttar Pradesh-201206

(MAY 2025)

CERTIFICATE

Certified that **Aanchal (202410116100002), Akansha Tyagi(202410116100014), Akansha Tomar(202410116100013), Deepu Kumari(202410116100057)** has/ have carried out the project work having "**Expense Tracker**" (MINI-PROJECT 2) (ID201B) for **Master of Computer Application** from Dr. A.P.J. Abdul Kalam Technical University (AKTU) (formerly UPTU), Lucknow under my supervision. The project report embodies original work, and studies are carried out by the student himself/herself and the contents of the project report do not form the basis for the award of any other degree to the candidate or to anybody else from this or any other University/Institution.

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Date:

Dr. Vipin Kumar
Associate Professor
Department of Computer Applications
KIET Group of Institutions, Ghaziabad
An Autonomous Institutions

Dr. Akash Rajak
Dean
Department of Computer Applications
KIET Group of Institutions, Ghaziabad
An Autonomous Institutions

Expense Tracker
(Aanchal, Akansha Tyagi, Akansha Tomar, Deepu Kumari)

ABSTRACT

The development and implementation of an Expense Tracker application designed to assist users in managing their personal finances effectively. The primary objective of the application is to provide a user-friendly interface that allows individuals to record, categorize, and analyze their expenditures in real-time. The application incorporates features such as budget setting, expense categorization, and visual data representation through charts and graphs, enabling users to gain insights into their spending habits.

The project employs a combination of front-end and back-end technologies to ensure a seamless user experience and robust data management. User feedback was collected during the development phase to refine functionalities and enhance usability. The application was tested for performance, security, and scalability, ensuring it meets the needs of a diverse user base.

Preliminary results indicate that users who actively engage with the Expense Tracker report improved financial awareness and better budgeting practices. This report concludes with recommendations for future enhancements, including the integration of machine learning algorithms for personalized financial advice and the potential for multi-platform accessibility. The Expense Tracker stands as a valuable tool for individuals seeking to take control of their financial health in an increasingly complex economic landscape.

Keyword: Graphs, records, expenditures, development, financial , individual.

ACKNOWLEDGEMENTS

Success in life is never attained single-handedly. My deepest gratitude goes to my project supervisor, **Dr. Vipin Kumar** for his/ her guidance, help, and encouragement throughout my project work. Their enlightening ideas, comments, and suggestions.

Words are not enough to express my gratitude to **Dr. Akash Rajak, Professor and Dean, Department of Computer Applications**, for his insightful comments and administrative help on various occasions.

Fortunately, I have many understanding friends, who have helped me a lot on many critical conditions.

Finally, my sincere thanks go to my family members and all those who have directly and indirectly provided me with moral support and other kind of help. Without their support, completion of this work would not have been possible in time. They keep my life filled with enjoyment and happiness.

Aanchal

Akansha Tyagi

Akansha Tomar

Deepu Kumari

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW

The digital era has revolutionized financial management for individuals and businesses alike. Traditional, manual expense tracking methods, such as paper-based records or spreadsheets, are becoming increasingly inefficient and inadequate in today's fast-paced, data-driven environment. The Expense Tracker project directly addresses these challenges by offering an innovative, user-friendly, and highly efficient web-based application designed for seamless recording, in-depth analysis, and proactive management of personal and business expenses.

The Expense Tracker's primary purpose is to simplify expense management, offering a centralized platform accessible from any device. By automating tasks and providing real-time insights, the application reduces administrative overhead and promotes financial awareness. Its key features include intuitive expense entry, customizable categories, budget setting, and visually appealing reports that reveal spending patterns.

The project aims to replace error-prone spreadsheets and basic budgeting apps with a powerful, yet affordable alternative. By digitizing records, automating notifications, and providing insightful analytics, the Expense Tracker helps users identify areas for savings, track progress toward financial goals, and make data-driven decisions.

Targeting a diverse audience that includes individuals seeking better control over their personal finances, freelancers managing both personal and business expenses, and small businesses aiming to streamline their accounting processes, the Expense Tracker offers a multitude of significant benefits: enhanced productivity through automation, reduced errors thanks to digitized records, improved financial awareness facilitated by comprehensive reporting, and a cost-effective solution that rivals complex accounting software. Its scope encompasses core functionalities such as comprehensive expense and category management, dynamic budgeting tools with customizable alerts, and a wide variety of customizable reports with interactive visualizations. The underlying MERN stack ensures not only scalability to accommodate growing data volumes and user bases, but also a responsive and engaging user interface that enhances the overall experience. Looking

ahead, future enhancements may include seamless bank integration for automatic transaction importing, dedicated mobile app development for on-the-go access, and the integration of advanced analytics features that provide even deeper insights into financial performance. Ultimately, the Expense Tracker project provides an invaluable tool for users from all walks of life seeking to simplify their financial lives, achieve greater financial well-being, and make informed decisions that pave the way for a secure and prosperous future.

1.2 PURPOSE OF THE PROJECT

The digital transformation sweeping across industries has highlighted the need for innovative solutions, particularly in the realm of financial management. As individuals and businesses navigate intricate fiscal landscapes, the challenge of effective expense management becomes paramount. The **Expense Tracker** project emerges as a solution designed explicitly to revolutionize and automate the historically cumbersome process of tracking expenses. For users heavily reliant on manual or semi-automated systems, the Expense Tracker serves as a much-needed bridge between tradition and innovation, ultimately freeing users to focus on strategic financial objectives rather than administrative minutiae.

1.2.1 Primary Purpose

At its core, the **Expense Tracker** is designed to provide a **centralized**, **secure**, and **easily accessible** platform for recording, storing, and analyzing financial records. By revolutionizing the way expense management is approached, the project aims to transform how users interact with their financial data. Key functionalities of the Expense Tracker include effortless expense recording, real-time access to financial information, and in-depth analytical capabilities. The platform empowers users to make informed decisions based on accurate data, which is critical for effective financial planning and management.

Moreover, the application significantly reduces administrative overhead, allowing users to streamline their operations. Through automation and centralization, users can mitigate the burdens associated with manual tracking methods—such as paper receipts and scattered spreadsheets—leading to greater operational efficiency, enhanced productivity, and reduced risk of errors.

1.2.2 Key Goals of the Expense Tracker

To accomplish its overarching purpose, the **Expense Tracker** has established several key goals that guide its development and functionality:

1. **Centralized Record Management**

Objective: The Expense Tracker aims to provide a singular, secure, and intuitive platform for all expense-related data. Given the fragmentation often present in manual systems—

typically involving a haphazard collection of spreadsheets, paper records, and notes—this centralization is crucial.

Implementation:

a. Users can log all expenses in one secure database, which allows for *easy retrieval* and *management*. This eliminates time-consuming searches through various documents or applications.
b. Each user account features an organized structure where transactions are categorized and stored based on user-defined preferences, ensuring people can track their financial activities without confusion.

Benefits:

c. Enhanced efficiency in record-keeping processes minimizes the risk of errors in data entry and reporting.
d. Users benefit from a holistic view of their financial landscape, promoting better understanding and clearer insights into their financial health.

2. **Automation**

Objective: The push for automation is a defining feature of the Expense Tracker. The application automates repetitive tasks that often burden traditional expense management systems.

Implementation:

a. Users will benefit from the automatic generation of detailed expense reports, eliminating the need for manual compilation.
b. Timely reminders for upcoming bills and due dates will ensure that users never miss an important payment.
c. The platform includes functionality for categorizing expenses automatically based on predefined rules (such as those defined by transaction types), enabling a streamlined user experience.

Benefits:

d. Users will save valuable time, which can be repurposed for strategic financial oversight and planning.
e. Automation encourages consistent and accurate budgeting practices by continuously monitoring expenditures and providing timely insights.

3. User Empowerment

Objective: Empowering users through education and insights is a crucial aim of the Expense Tracker. Given that many users seek to improve financial literacy, the platform is designed to provide actionable insights into spending habits.

Implementation:

a. The use of comprehensive reports and visualizations aids users in understanding their financial behaviors. Whether it's through pie charts depicting expense distributions or trend graphs showcasing spending over time, users can gain meaningful perspectives on their finances.
b. The application also includes tips and guidance on effective budgeting practices, encouraging users to adopt solid financial habits.

Benefits:

c. With greater awareness of their financial position, users can make informed decisions regarding their budgetary allocations, leading to improved savings and financial security.
d. The real-time insights foster a proactive approach to financial management, allowing users to not just react to metrics but also plan strategically for future financial goals.

4. **Accessibility**

Objective: In a mobile-driven world, accessibility across devices is a vital goal of the Expense Tracker. Users require the flexibility to manage their expenses regardless of location or device type.

Implementation:

a. The web-based platform is developed using responsive design techniques, granting users seamless access whether they are on desktops, tablets, or smartphones.
b. Cloud-based infrastructure ensures that users can log in and manage their finances anywhere at any time, transforming how users approach their expenses.

Benefits:

c. Users no longer need to carry physical records or rely on a single device. Instead, they can utilize their preferred device for managing expenses, meeting their personal preferences and behaviors.
d. The convenience of mobile access increases the likelihood of users consistently engaging with the platform, thereby improving adherence to budgeting goals and tracking activities.

1.2.3 Bridging the Gap Between Traditional and Modern Solutions

By successfully meeting the outlined objectives, the **Expense Tracker** effectively bridges the gap between outdated manual bookkeeping methods and complex, often expensive accounting software solutions. This innovative platform provides an affordable yet potent alternative that addresses the specific needs of its target audience.

Advantages Over Traditional Systems:

- Unlike traditional methods, which often result in **fragmented**, **error-prone**, or **inefficient** record-keeping, the Expense Tracker streamlines and simplifies these processes.
- Users benefit from a holistic view of their financial data, reducing unnecessary complexities inherent in outdated systems.

1.3 BACKGROUND

In the realm of traditional financial management, individuals, small business owners, and freelancers have traditionally relied on a variety of methods to track their expenses, including physical ledgers, spreadsheets created with software like Microsoft Excel, or basic budgeting applications. While these methods may suffice for very simple needs, they often lack the advanced features necessary for comprehensive financial management, such as automation capabilities, detailed data visualization tools, and real-time insights that can drive informed decision-making.

Consider the following examples:

1. An individual might inadvertently forget to record a small cash expense, such as a coffee purchase, leading to an inaccurate and incomplete monthly budget.
2. A freelancer may encounter difficulties when attempting to categorize various business expenses for tax purposes, resulting in wasted time spent on manual sorting and calculations.
3. A small business owner may find it challenging to effectively track and analyze different expense categories to identify potential cost-saving opportunities and optimize resource allocation.

The Expense Tracker directly addresses these challenges by offering:

1. Digitizing Expense Records: Users can effortlessly create, edit, and retrieve their financial data anytime and from anywhere with an internet connection, all within a secure and user-friendly online platform.
2. Automating Notifications: The system is capable of sending automated reminders for recurring expenses, upcoming bill payments, and adherence to predefined budget limits, ensuring timely and proactive expense management.
3. Providing Insights: The Expense Tracker generates dynamic reports and intuitive visualizations that enable users to easily identify spending patterns, pinpoint areas for potential improvement, and track their progress toward achieving their financial goals.

The application's intuitive and user-friendly interface ensures accessibility for users with varying levels of technical expertise, while its scalable MERN (MongoDB, Express.js, React.js, Node.js) stack architecture ensures it can accommodate growing financial needs and data volumes.

1.4 PROBLEM STATEMENT

While manual expense management methods remain prevalent, they present a multitude of significant challenges that can hinder effective financial planning and control:

1. Prone to Errors: Manual data entry, categorization, and calculations are susceptible to human error, leading to inaccurate records and potentially flawed decision-making based on that inaccurate data.
2. Time-Consuming: The process of sorting, categorizing, and managing expenses manually can be tedious and time-consuming, particularly for individuals and businesses that handle a high volume of transactions.
3. Lack of Automation: Manual systems require users to remember to record expenses, track adherence to budgets, and generate reports, which can lead to missed opportunities for optimization and potential oversights in financial management.
4. Limited Insights: Without the aid of data visualization tools and advanced analytics, users may struggle to effectively analyze spending trends, identify opportunities for savings, and gain a comprehensive understanding of their overall financial picture.

These limitations underscore the clear need for an efficient, automated, and user-centric system like the Expense Tracker. By combining the power of modern technology with a focus on user-centered design principles, the Expense Tracker overcomes these challenges and empowers users to take control of their financial lives. By automating repetitive tasks, providing dynamic reports that offer valuable insights, and ensuring secure and reliable record-keeping, the Expense Tracker enables users to focus on their financial goals rather than being burdened by administrative tasks.

1.5 SIGNIFICANCE OF THE PROJECT

The significance of the Expense Tracker lies in its transformative ability to enhance expense management for individuals, freelancers, and small businesses through automation, improved accuracy, and the provision of actionable insights.

Its value can be summarized in the following key points:

1. Enhanced Productivity: By automating time-consuming tasks such as expense recording, categorization, report generation, and data storage, the Expense Tracker frees up valuable time, allowing users to focus on strategic financial planning and decision-making.
2. Error Reduction: By minimizing reliance on manual data entry and calculations, the system significantly reduces the risk of errors that could lead to inaccurate financial reports or poor financial decisions.
3. Improved Decision-Making: Real-time data analysis and intuitive visual insights empower users to make well-informed decisions regarding their finances. This includes identifying areas where they can reduce spending, optimizing budget allocations to maximize savings, and tracking their progress toward achieving their financial goals.
4. Increased Financial Awareness: The Expense Tracker provides users with a clear, comprehensive, and up-to-date view of their spending habits, promoting greater financial awareness and encouraging responsible financial behavior.

5. Cost-Effective Solution: Unlike complex and often expensive accounting software packages, the Expense Tracker offers a cost-effective alternative that makes advanced expense management tools accessible to individuals, freelancers, and smaller businesses with limited financial resources.

The project's ability to integrate with bank APIs to automate transaction importing and offer customizable reporting options further enhances its value proposition, ensuring a seamless, adaptable, and personalized user experience.

1.6 SCOPE OF THE PROJECT

The scope of the Expense Tracker encompasses core functionalities that are essential for effective expense management:

1. Expense Management:
    a. Users can easily add, view, edit, and delete expense entries, providing a flexible and intuitive way to manage their financial data.
    b. All data is stored securely in a MongoDB database, ensuring persistence, reliability, and efficient retrieval when needed.
2. Category Management:
    a. Users have the ability to categorize expenses using a set of predefined categories (e.g., food, transportation, utilities) or create custom categories that align with their specific needs and spending patterns.
    b. The categorization feature enables detailed analysis and comprehensive reporting on spending habits, providing users with a clear understanding of where their money is going.
3. Budgeting:
    a. Users can create budgets for different expense categories, setting spending limits and tracking their progress against those budgets in real-time.
    b. The budgeting tools within the Expense Tracker help users stay within their financial limits, identify potential overspending, and achieve their savings goals.
4. Financial Insights and Reports:
    a. Users can access a variety of dynamic reports that provide detailed expense breakdowns by category, time period (e.g., monthly, quarterly, annually), and payment method.
    b. Data visualization tools, such as charts and graphs, transform raw financial data into easily understandable and actionable insights, enabling users to quickly grasp key trends and patterns.
5. User Management and Customization:
    a. Users can manage their profiles, update their personal details, and customize the application's appearance to suit their preferences. This includes options for adjusting themes, color schemes, and notification settings.
6. Future Expansion:
    a. The project is designed with future expansion in mind, with potential for the addition of features such as multi-user access (allowing multiple individuals within a household or business to collaborate on expense tracking), PDF/Excel export of reports for offline analysis, integration with bank APIs

for automated transaction importing, and the development of native mobile applications for iOS and Android platforms.

1.7 TARGET AUDIENCE

The Expense Tracker is specifically designed to cater to the needs of the following target audiences:

1. Individuals: People who are seeking to track their personal expenses, manage their budgets effectively, and gain greater control over their overall financial well-being.
2. Freelancers: Independent professionals who need to meticulously track both their business expenses and income for accurate tax reporting and financial management purposes.
3. Small Businesses: Business owners who are looking for a simplified and cost-effective expense management system that doesn't require the investment in complex and expensive accounting software solutions.

1.8 BENEFITS OF THE EXPENSE TRACKER

The Expense Tracker offers a wide range of benefits that make it an invaluable tool for anyone seeking to improve their financial management practices:

1. Simplified Workflows: The application streamlines the entire expense management process, reducing administrative burdens and freeing up valuable time for more strategic activities.
2. Scalable Architecture: The MERN stack architecture provides a solid foundation for the application, ensuring that it can scale seamlessly to accommodate growing user needs, increasing transaction volumes, and the addition of new features over time.
3. User-Friendly Interface: The application boasts a responsive and intuitive design that ensures ease of use across a variety of devices, including desktops, laptops, tablets, and smartphones.
4. Real-Time Features: Instant notifications and real-time financial updates keep users informed and in control of their expenses, allowing them to make timely adjustments to their spending habits and budgets.

1.9 FUTURE ENHANCEMENTS

To maintain relevancy in a rapidly evolving digital landscape, the project aims to incorporate several new features in future iterations. These enhancements aim to further solidify the Expense Tracker's place as a leader in expense tracking solutions.

1. Integration with Financial Planning Apps
   Integrating with popular financial planning tools can provide users with a well-rounded approach to managing their finances.
2. Smart Recommendations

Utilizing advanced algorithms to analyze user behavior could enable the application to offer personalized spending suggestions and budgetary recommendations.

3. Advanced Security Features
   Enhancements to data security, such as two-factor authentication, could provide users with greater peace of mind regarding their financial information.
4. Collaboration Tools
   Expanding functionalities to facilitate collaboration amongst users—such as account sharing—could be beneficial for families or teams managing shared expenses.

CHAPTER 2

FEASIBILITY STUDY/LITERATURE REVIEW

A feasibility study assesses the viability, achievability, and sustainability of the proposed Expense Tracker project. This includes analyzing technical capabilities, financial implications, operational readiness, and societal impact. This section also provides a literature review of existing solutions to identify gaps and opportunities.

2.1 TECHNICAL FEASIBILITY

The technical feasibility assessment focuses on the practical aspects of building the Expense Tracker using the MERN stack. This involves evaluating the availability, maturity, and suitability of the chosen technologies, as well as the technical expertise required for development and deployment. The MERN stack (MongoDB, Express.js, React.js, Node.js) provides a full-stack JavaScript solution, promoting code reuse and simplifying the development process.

1. Frontend Development

a. The user interface will be developed using React.js, a popular JavaScript library for building dynamic and interactive user interfaces. React's component-based architecture allows for creating reusable UI elements, improving code maintainability and scalability. HTML5 and CSS3 will be used for structuring and styling the application, ensuring a visually appealing and responsive design. A component library such as Material-UI or Ant Design may be incorporated to accelerate development and provide a consistent design language.

i. Advantages:

1. React's virtual DOM (Document Object Model) optimizes rendering performance, resulting in a smooth and responsive user experience.

2. The component-based architecture promotes code reusability, reducing development time and improving maintainability.

3. A large and active community provides ample resources, libraries, and support for React development.

ii. Scalability:

1. React's efficient rendering techniques and virtual DOM enable the application to handle a growing number of components and data updates without significant performance degradation.

2. The component-based architecture allows for easy addition of new features and UI elements as the application evolves.

2. Backend Development

a. The backend system will be built using Node.js, a JavaScript runtime environment that allows you to run JavaScript on the server. Express.js, a minimalist web application framework for Node.js, will be used to handle API requests, manage data, and perform server-side logic. This combination provides a robust and efficient platform for building the application's backend.

i. Node.js:

1. A JavaScript runtime environment that allows developers to use JavaScript for both frontend and backend development, simplifying the development process.

2. Its non-blocking, event-driven architecture makes it highly efficient for handling concurrent requests, ensuring optimal performance even under heavy load.

ii. Express.js:

1. A minimalist web application framework that provides a robust set of features for building APIs and web applications, including routing, middleware support, and template engines.

2. Simplifies the development process by providing a clear and organized structure for building the backend.

3. Database

a. MongoDB, a NoSQL database, will be used to store expense data, user information, and category details. MongoDB's document-oriented structure allows for flexible and scalable data storage, making it well-suited for handling diverse expense data.

i. Advantages:

1. MongoDB's flexible schema allows for easy modification and evolution of the data structure without requiring complex migrations.

2. The document-oriented structure makes it easy to store and retrieve complex data structures, such as nested expense categories and user preferences.

ii. Scalability:

1. MongoDB can be scaled horizontally to accommodate large datasets and high traffic loads, ensuring that the application can handle growing user base and data volume.

2. Support for sharding and replication provides high availability and data redundancy.

4. API Integration

a. The Expense Tracker may integrate with third-party APIs to enhance its functionality and provide additional features. Potential integrations include:

i. Bank Integration: Plaid or similar APIs for automatic transaction importing, allowing users to seamlessly connect their bank accounts and import transaction data.

ii. Currency Conversion: APIs for handling expenses in multiple currencies, enabling users to track expenses in different currencies and convert them to their preferred currency.

iii. Mapping APIs: Integration with mapping services like Google Maps to visualize expenses based on location.

5. Hosting and Accessibility

a. The application will be hosted on cloud platforms such as AWS, Heroku, or MongoDB Atlas, ensuring high availability, scalability, and accessibility across devices. Cloud hosting provides a cost-effective and reliable infrastructure for deploying and managing the application.

i. Cloud Hosting:

1. Offers cost-effective and scalable hosting solutions, allowing the application to grow as needed without requiring significant upfront investment in hardware.

2. Provides high availability and reliability, ensuring that the application is always accessible to users.

3. Simplifies deployment and management through automated tools and services.

2.2 ECONOMIC FEASIBILITY

The economic feasibility analysis assesses the costs and benefits associated with developing and deploying the Expense Tracker. This includes evaluating development costs, operational costs, potential revenue streams, and the overall return on investment (ROI). The goal is to determine whether the project is financially viable and can generate sufficient value to justify the investment [2].

1. Development Costs

a. Hardware: Development can be performed on standard computers, reducing hardware costs [1].

b. Software: The MERN stack utilizes open-source technologies, significantly reducing software costs compared to proprietary solutions [1].

c. Third-party Services: Costs for services such as cloud hosting, domain registration, SSL certificates, and potentially paid API subscriptions (depending on the features implemented).

d. Development Team: Costs associated with hiring developers, designers, and project managers. These costs will vary depending on the team's location, experience, and skill set.

2. Operational Costs

    a. Cloud Hosting: Cloud hosting costs are proportional to usage, allowing for cost optimization. Costs will depend on factors such as storage, bandwidth, and computing power.

    b. Maintenance: Maintenance costs associated with server upkeep, database management, security updates, and bug fixes.

    c. Support: Costs associated with providing customer support, including responding to user inquiries and resolving technical issues.

3. Return on Investment (ROI)

    a. Time Savings: Time saved by users through automated expense tracking, reducing the need for manual data entry and analysis.

    b. Improved Financial Awareness: Improved financial awareness and decision-making, enabling users to better manage their finances and achieve their financial goals.

    c. Reduced Errors: Reduction in errors associated with manual expense tracking, leading to more accurate financial records.

4. Revenue Opportunities

    a. Subscription Model: Offering premium features (advanced reporting, multiple users, custom categories, priority support, etc.) for a recurring fee.

    b. Freemium Model: Providing basic functionality for free and charging for additional features or usage limits.

    c. Affiliate Marketing: Partnering with financial service providers and earning commissions on referrals.

d. Data Analytics: Aggregating and anonymizing user data to provide insights to financial institutions (with user consent).

2.3 OPERATIONAL FEASIBILITY

The operational feasibility analysis assesses the ease of implementation and use of the Expense Tracker by its target audience. This includes evaluating the target audience, user-friendliness, system maintenance requirements, and training needs. The goal is to ensure that the application is accessible, intuitive, and can be effectively used by individuals with varying levels of technical expertise .

1. Target Audience

    a. Individuals: Users looking to track personal expenses, manage budgets, and gain insights into their spending habits.

    b. Freelancers: Professionals needing to monitor business expenses, track income, and manage invoices.

    c. Small Businesses: Owners seeking a simple and affordable tool for managing company finances, tracking expenses, and generating reports.

2. Ease of Use

    a. A user-friendly interface is crucial for ensuring adoption by users with varying technical skills .

    b. Intuitive navigation, clear labeling, and interactive elements enhance the user experience.

    c. Mobile-first design ensures that the application is accessible and usable on smartphones and tablets.

    d. Onboarding process that guides new users through the application's features and functionalities.

3. System Maintenance

    a. Regular database backups to ensure data integrity and prevent data loss .

    b. Continuous monitoring of application performance and server health to identify and resolve potential issues proactively.

    c. Easy deployment of updates and bug fixes through cloud-based infrastructure .

    d. Automated testing to ensure that new features and updates do not introduce regressions or break existing functionality.

4. Training Requirements

a. Minimal training is required due to the system’s intuitive design .

b. Comprehensive documentation, FAQs, and video tutorials can assist users in understanding the application's features and functionalities.

c. Contextual help and tooltips within the application to guide users through specific tasks.

d. Responsive customer support to address user inquiries and resolve technical issues.

2.4 SOCIAL FEASIBILITY

The social feasibility analysis evaluates the acceptance and impact of the Expense Tracker on its users and the broader community. This includes assessing the user impact, environmental impact, social adoption, and long-term benefits [3]. The goal is to ensure that the application contributes positively to society and aligns with ethical and social values.

1. User Impact

    a. Empowering users to better manage their finances, make informed decisions, and achieve their financial goals .

    b. Reducing the stress and time associated with manual expense tracking, freeing up users to focus on other priorities .

    c. Providing users with valuable insights into their spending habits, enabling them to identify areas where they can save money and improve their financial health.

2. Environmental Impact

    a. Reducing paper consumption through digital record-keeping, contributing to environmental sustainability .

    b. Promoting responsible consumption habits by encouraging users to track their expenses and make informed purchasing decisions.

3. Social Adoption

    a. Promoting financial literacy and responsible spending habits among individuals, freelancers, and small businesses .

    b. Making financial management tools accessible to a wider audience, including those who may not have access to traditional financial services.

4. Long-Term Benefits

    a. Contributing to a more financially aware and responsible society, empowering individuals to achieve financial security and build a better future .

b. Supporting small businesses and freelancers by providing them with the tools they need to manage their finances effectively and grow their businesses.

2.5 LITERATURE REVIEW

This section provides a critical review of existing expense tracking solutions, identifying their strengths, weaknesses, and the gaps that the Expense Tracker aims to address. The review covers spreadsheets, mobile apps, accounting software, and other relevant tools. It also examines research studies and industry reports related to financial tracking, budgeting, and user preferences. The goal is to gain a comprehensive understanding of the competitive landscape and identify opportunities for innovation .

1. Existing Solutions

    a. Spreadsheets: While flexible and customizable, spreadsheets are prone to errors, lack automation, and can be difficult to manage for large datasets .

    b. Mobile Apps: Many expense tracking apps exist (e.g., Mint, YNAB, Personal Capital), offering features such as automated transaction importing, budgeting, and reporting. However, these apps may have limitations in terms of customization, privacy, or integration with specific financial institutions .

    c. Accounting Software: Solutions like QuickBooks, Xero, and FreshBooks are powerful but can be too complex and expensive for individuals or very small businesses. They often include features that are not relevant to basic expense tracking, making them overkill for simple needs .

    d. Other Tools: Other tools, such as budgeting apps, investment trackers, and personal finance dashboards, may include some expense tracking functionality, but they typically focus on a broader range of financial management tasks.

2. Gaps Identified

    a. Need for a customizable and user-friendly solution that can be tailored to specific needs and preferences.

    b. Lack of seamless integration with bank accounts and other financial services, requiring users to manually enter transaction data.

    c. Limited reporting and analytics capabilities in some existing tools, making it difficult to gain actionable insights into spending habits .

    d. Privacy concerns related to sharing financial data with third-party apps.

    e. Cost barriers that make some solutions inaccessible to individuals and small businesses with limited budgets.

3. Research Studies

a. Studies on the benefits of financial tracking and budgeting, demonstrating the positive impact on financial health and well-being.

b. Research on user preferences and pain points in existing expense management solutions, identifying the features and functionalities that users value most.

c. Industry reports on trends in personal finance management, highlighting the growing demand for mobile and cloud-based solutions.

4. Proposed Solution

a. The Expense Tracker aims to bridge the gap by providing a flexible, affordable, user-friendly, and privacy-focused solution built on modern web technologies.

b. The MERN stack enables the development of a scalable and interactive application that meets the needs of individuals, freelancers, and small businesses.

c. Key differentiators include customizable categories, seamless bank integration, advanced reporting, and a commitment to data privacy.

CHAPTER 3

PROJECT OBJECTIVE

The **Expense Tracker project** continues to evolve, aiming to provide a comprehensive personal financial management solution that adapts to the changing needs of its users across various demographics.

3.1 PRIMARY OBJECTIVES

3.1.1 Develop a User-Friendly Expense Tracking Application

Goal: Create an intuitive and accessible web application that simplifies the expense tracking experience.

Details:

The application will be developed using `React.js`, ensuring a **clean**, modern, and responsive design.
A dedicated **onboarding process** will facilitate new users in understanding key functionalities through interactive tutorials and guided tours.
Regular updates will be integrated into the user interface based on the design trends and feedback from user testing, maintaining a contemporary feel.
An extensive **FAQ section** and user documentation will be available, providing support for common questions and troubleshooting tips.
User experience (UX) testing will include **A/B testing** for alternative layouts or feature placements to determine the most effective user interface.

3.1.2 Provide Comprehensive Expense Management

Goal: Facilitate efficient recording, categorization, and management of expenses in a streamlined manner.

Details:

Users will have options to track **personal vs. business expenses**, ensuring clear distinctions for freelancers and small business owners.
An **expense analysis dashboard** will offer a visual representation of spending trends over selected periods, providing users with a quick glance into their financial patterns.
The application will implement **machine learning algorithms** to provide suggestions for categorization based on historical user behavior, making data entry more efficient.
Support features for **bulk expense uploads** through spreadsheet imports (e.g., CSV) will be included for users transitioning from manual tracking methods.

3.1.3 Implement Budgeting and Goal-Setting Tools

Goal: Enable users to set budgets and financial goals to cultivate awareness and discipline in managing their finances.

Details:

Users will have the ability to set **savings goals** (e.g., vacation, car purchase) alongside their expense budgets, tracking progress in one unified dashboard.
Scenario simulation tools will allow users to visualize how adjusting spending in one category impacts their overall budget and goals.
The application will offer **educational pop-ups** within the budgeting tools, providing financial tips and best practices when setting up budgets and goals.
Integration with external tools, such as **retirement calculators**, will deliver comprehensive financial planning support within the same platform.

3.1.4 Offer Real-Time Reporting and Analytics

Goal: Equip users with insights into their spending habits and overall financial health through detailed reporting and analytics.

Details:

Reports will support **data slicing** for customizable views, enabling users to focus on specific categories or time frames.
The use of **interactive graphs and charts** will allow users to explore their data in a more engaging manner, promoting deeper insights into spending habits.
Users will receive **monthly summary reports**, delivered via email, highlighting key trends, insights, and tips for optimizing future spending.
Integration of **predictive analytics** will suggest potential spending risks, flagging potential overspending scenarios based on historical data.

3.1.5 Ensure Data Security and Privacy

Goal: Safeguard user data through robust security measures, ensuring trust and confidence in the application.

Details:

A **comprehensive data backup strategy** will ensure that users can recover their financial data in case of unexpected issues.
The security system will be monitored continuously for vulnerabilities, with regular updates applied promptly to address any discovered issues.
Users will be walked through **security settings** during onboarding, allowing them to customize their privacy preferences to fit their comfort levels.
A clear **data retention policy** will be employed, only retaining user data as long as necessary for functionality while ensuring compliance with privacy laws.

3.2 SECONDARY OBJECTIVES

1. Facilitate Automated Notifications and Reminders

- **Goal**: Develop a customizable notification system to enhance proactive financial management.
- **Details**:
    - Users will be able to customize reminders based on **specific spending thresholds** within categories, receiving alerts before they exceed their budget.
    - Notifications will include tips for alternative spending or saving mechanisms identified through user behavior trends, promoting financial growth through awareness.
    - A **web and mobile app integration** will allow users to receive notifications directly on their smart devices regardless of their location, ensuring they stay informed on the go.

2. Explore Future Enhancements and Integrations

- **Goal**: Plan for scalability and feature enhancements to continually expand the application's functionality.
- **Details**:
    - The team will explore partnerships with **fintech companies** to leverage additional financial tools or features that can enhance the value of the application.
    - A **roadmap for feature releases** will be established, prioritized using user feedback and market analysis to ensure relevance and demand fulfillment.
    - Advanced features, such as **automated tax calculations** based on inputted expenses and income streams, may be developed in response to user requests.
    - The possibility of creating **integration partnerships** with local businesses to offer cash-back incentives for users who record expenses tied to those vendors will be explored.

3. **Conduct Market Research and User Testing**

- **Goal**: Regularly collect insights into user preferences and needs to ensure relevancy and enhanced user experience.
- **Details**:
    - Monthly user research sessions will help gather qualitative data, enhancing user understanding through direct conversations about their experiences.
    - **Feedback loops** will be established, detailing how user suggestions are incorporated into updates, thus creating a stronger community connection.
    - Continuous improvement will be driven through monitoring **user activity metrics**, helping to identify underutilized features or necessary adjustments in functionality.
    - Utilizing **competitive benchmarking** will ensure that the application remains relevant and superior compared to similar tools available in the market.

4. **Promote Financial Literacy and Responsibility**

- **Goal**: Enhance users' financial literacy through educational resources and tools.
- **Details**:
    - The application will feature **monthly webinars** with financial experts discussing various personal finance topics, available exclusively to users.
    - Users will be encouraged to participate in **financial wellness challenges**, incentivizing them to achieve specific goals with the application.
    - A **gamification approach** will be integrated to reward users for completing tutorials or achieving budgeting milestones, promoting engagement with financial learning.
    - Strong content partnerships will be established with credible **financial literacy organizations** to provide users with access to high-quality resources and advice.

5. **Foster Community Engagement and Support**

- **Goal**: Build a connected community where users can share experiences and foster collaborative learning.
- **Details**:
    - A **dedicated community manager** will be assigned to foster user engagement through regular interactions on forums and social media.
    - An **event calendar** will be launched to highlight community events, webinars, user contests, and challenges that encourage participation.
    - Users will be able to create and join **interest-based groups** within the application, facilitating better connections based on their financial goals and challenges.

CHAPTER 4

HARDWARE AND SOFTWARE REQUIREMENT

4.1 HARDWARE REQUIREMENTS

- To ensure optimal performance and a seamless user experience while using the Expense Tracker Application, the following hardware specifications are recommended:

1. Processor (CPU):

- **Intel i5** or higher (equivalent AMD processor or better)
- A multi-core processor with a clock speed of 2.0 GHz or higher is recommended for smooth multitasking, especially when handling multiple users or large sets of transaction data.
- Higher-end processors (Intel i7 or AMD Ryzen 7) will improve overall responsiveness and speed when processing complex data visualizations.

2. RAM (Memory):

- 8 GB minimum
- 8 GB of RAM is recommended for running the application efficiently. This allows smooth multitasking and ensures that the system can handle multiple browser tabs or instances of the application simultaneously.
- For better performance, especially when running multiple applications concurrently, 16 GB or more of RAM is advised.

3. Storage:

- **500 GB HDD/SSD**
- A minimum of 500 GB of storage space is recommended. Using a **Solid State Drive (SSD)** will provide faster data retrieval speeds, leading to quicker application load times and improved overall performance, particularly when dealing with large datasets and real-time updates.

- A **Hard Disk Drive (HDD)** is acceptable but may result in slower data access times compared to an SSD.

4. Display**:**

- **15.6" or larger**
- A display with a screen size of at least 15.6 inches is recommended for comfortable navigation of the Expense Tracker Application, especially when dealing with detailed charts and financial reports.
- A Full HD (1920x1080) resolution or higher is ideal for clear visuals, ensuring that charts, graphs, and transaction records are displayed without pixelation or scaling issues.
- For those who prefer higher screen real estate, a 17-inch or larger screen is advisable for even better visibility and ease of use.

5. Graphics:

- Integrated graphics (e.g., Intel UHD or AMD Vega) are sufficient for general usage. However, if you plan to visualize large datasets with heavy graphical representations, a **dedicated graphics card** (e.g., NVIDIA GeForce or AMD Radeon) may provide smoother rendering for complex visualizations.

6. Input Devices:

- **Keyboard**: A standard QWERTY keyboard is recommended for smooth data entry. A numeric keypad will aid in faster transaction entry.
- **Mouse**: A standard mouse or a trackpad for easy navigation of the user interface.
- **Touchscreen**: If using a touch-enabled laptop or tablet, touch interaction will provide an intuitive experience for interacting with charts and other elements.

7. Network Connectivity:

- **Internet Connection**: A stable internet connection is required for accessing and updating the Expense Tracker Application, particularly if using cloud-based services. For best performance, a broadband connection with speeds of at least 5 Mbps is recommended.
- **Wi-Fi**: Wi-Fi connectivity should be 5 GHz or compatible with your device for faster and more stable access.

4.2 SOFTWARE REQUIREMENTS

The software environment required to develop, run, and maintain the Expense Tracker Application is as follows:

1. Operating System (OS):

- **Windows 10** (or higher)
- **Linux** (any modern distribution, such as Ubuntu, CentOS, Fedora)
- **macOS** (any modern version)

The application is designed to be cross-platform, supporting Windows, Linux, and macOS for both development and deployment. Users can access the application from any of these platforms with a modern web browser.

2. Integrated Development Environment (IDE):

- **Visual Studio Code**

Visual Studio Code is the recommended IDE for development, as it supports JavaScript, Node.js, React, and various extensions for MongoDB and Express.js, providing a productive environment for full-stack development.

3. Web Browsers:

- **Google Chrome** (latest version)
- **Mozilla Firefox** (latest version)

These modern browsers are recommended for optimal rendering and performance. They support modern web technologies such as HTML5, CSS3, JavaScript (ES6+), and React.js.

4. Backend:

- **Node.js** (latest LTS version)
- **Express.js** (latest stable version)

Node.js is the runtime environment for the server-side operations, while Express.js, a minimal web framework for Node.js, facilitates the creation of RESTful APIs and routes for the application's backend.

5. Frontend:

- **React.js** (latest stable version)

React.js is used for building the user interface, enabling the application to be highly interactive and responsive. It allows for the efficient rendering of UI components and dynamic updates without page reloads, providing a smooth user experience.

6. Database:

- **MongoDB** (latest stable version)

MongoDB is the NoSQL database used for persistent data storage. It supports high flexibility with its document-based structure, making it ideal for storing transaction data, user profiles, and expense categories.

4.3 FUNCTIONAL REQUIREMENTS

The Expense Tracker Application must implement the following functional capabilities to meet the needs of the users:

1. Secure Login and Registration:

- The system should allow users to securely register, log in, and authenticate using a username and password.
- The authentication system should include a password hashing mechanism for secure storage and retrieval.
- Users must be able to reset their passwords in case of forgotten credentials through an email-based recovery process.

2. Expense CRUD Operations:

- Users should be able to **Create**, **Read**, **Update**, and **Delete** their expense entries.
- Expenses should include details like the amount, category (e.g., Food, Transportation, Entertainment), date, and description.
- CRUD operations should update the database in real time and reflect changes on the user interface without needing a page reload.

3. Categorized Reports and Dashboards:

- The application should enable users to categorize their expenses based on predefined categories such as Groceries, Bills, Entertainment, etc.
- Users should be able to view categorized expense reports and analyze spending patterns over various time periods (e.g., weekly, monthly, yearly).

4. Chart-based Visual Summaries:

- The system should provide visual reports in the form of pie charts, bar charts, and line graphs to help users track their spending habits.
- Charts should dynamically update based on user-selected filters such as time range, expense categories, and specific data points.
- Tools like Chart.js can be used for creating these interactive visualizations.

5. Data Export in CSV/PDF**:**

- Users should have the ability to export their expense records and reports in commonly used formats such as **CSV** and **PDF**.
- The CSV export should allow users to download their transaction history with all details for further analysis in spreadsheets.
- The PDF export should generate a clean, formatted report suitable for printing or sharing.

4.4 NON-FUNCTIONAL REQUIREMENTS

The following non-functional requirements must be adhered to for ensuring the application's quality, performance, and scalability:

1. Responsive and User-friendly UI:

- The user interface should be fully responsive, adapting to various screen sizes including desktops, tablets, and smartphones.
- The design should be intuitive and easy to navigate, with clear labels, consistent styling, and interactive elements like dropdowns, buttons, and charts.
- The UI should be lightweight and optimized for performance, ensuring fast load times even with complex reports or large datasets.

2. Secure Password Storage with Hashing:

- Passwords should be securely stored using hashing techniques, such as **bcrypt** or **argon2**, to ensure they are not stored in plaintext.
- The system must implement additional security measures like salting the password hashes and using secure sessions to protect user credentials during login.

3. Fast and Scalable CRUD Operations:

- The application must support fast CRUD operations for managing a potentially large set of transactions without significant delay.
- The backend should be optimized for scalability to accommodate increasing numbers of users and transactions.
- The database queries should be optimized to ensure fast data retrieval, even as the size of the data grows over time.

4. Clean Architecture Using MVC Pattern:

- The application should follow the **Model-View-Controller (MVC)** design pattern, ensuring a clean separation of concerns.
    - **Model**: Responsible for handling the database logic and interacting with the data layer (MongoDB).

    - **View**: The user interface components built using React.js, responsible for rendering data and interacting with the user.
    - **Controller**: The backend logic that handles user requests, processes them, and interacts with the database.
- This architecture helps ensure the application is modular, maintainable, and easy to scale or extend with new features in the future.

CHAPTER 5

PROJECT FLOW

5.1 Conceptual Models

System design begins with conceptual models that provide a visual and functional overview of the application. The UML (Unified Modeling Language) diagrams help in visualizing the components, relationships, and flows within the system.

1. **Use-case diagram:**
   Displays how different actors (such as Users and Admins) interact with the system. Key use cases include: User Registration, Login, Add Expense, Edit Expense, Delete Expense, View Reports, and Logout.

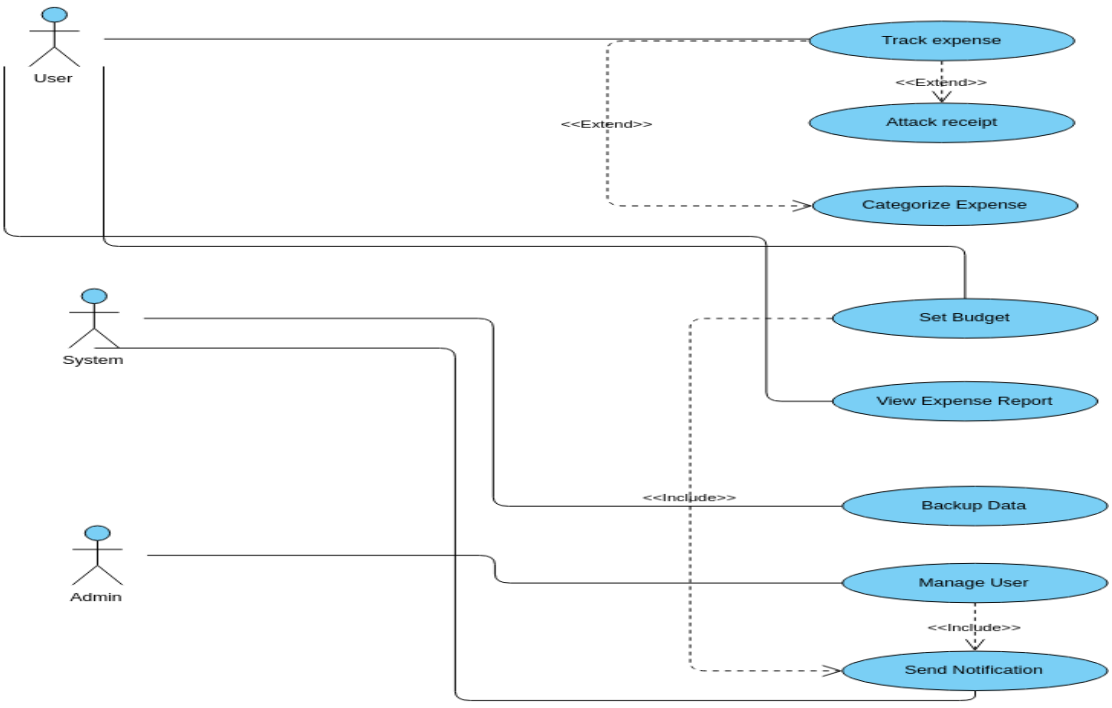

Fig.5.1: Use-case Diagram

2. **Class Diagram:**

Illustrates the structure of the system by showcasing classes such as User, Expense, and Category, along with their attributes and relationships. This forms the foundation for schema design.

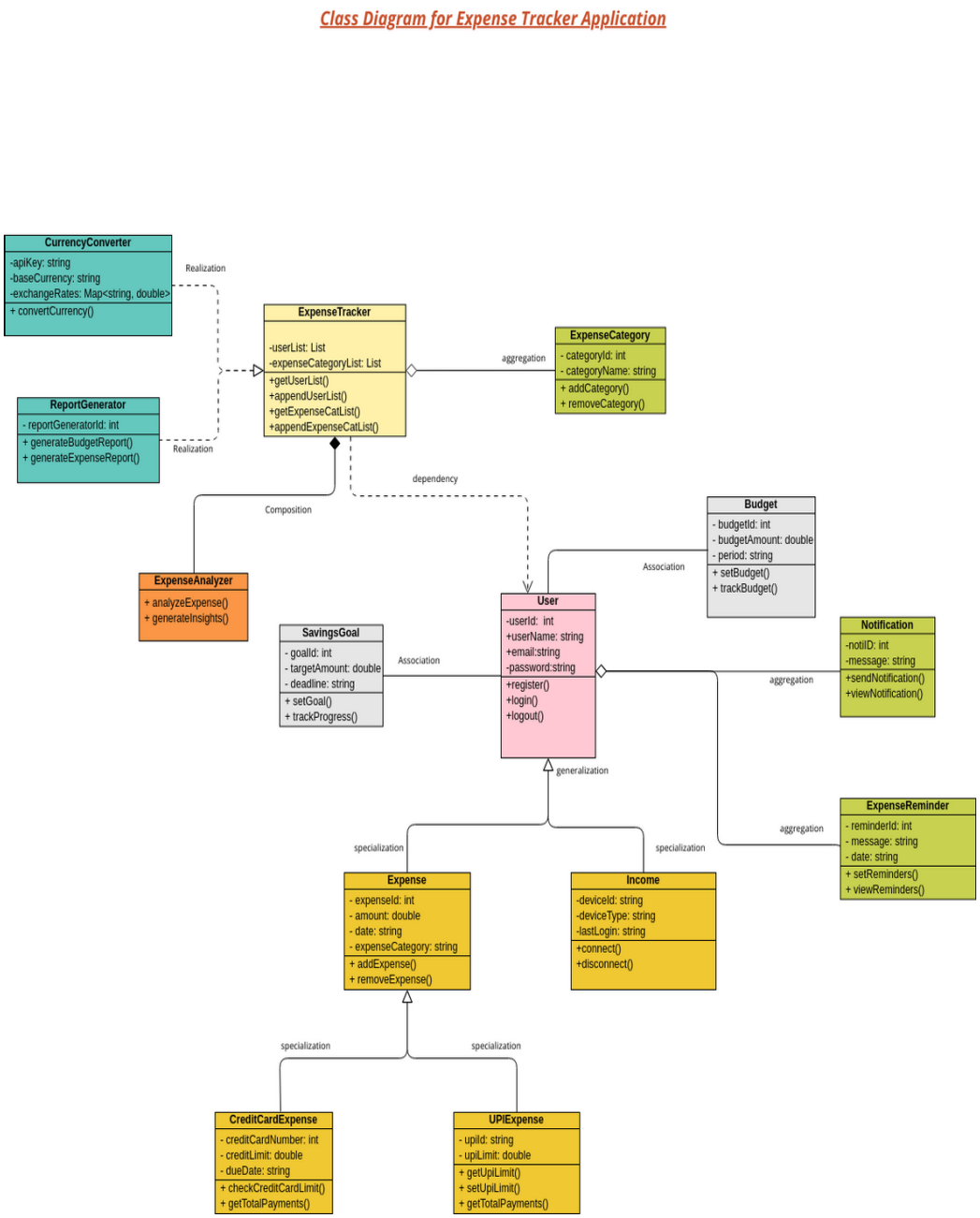

Fig.5.2: Class Diagram

3. **Sequence Diagram:**
   Describes the step-by-step flow of data and interactions, such as the sequence involved in adding an expense—starting from frontend input to backend processing and database storage.

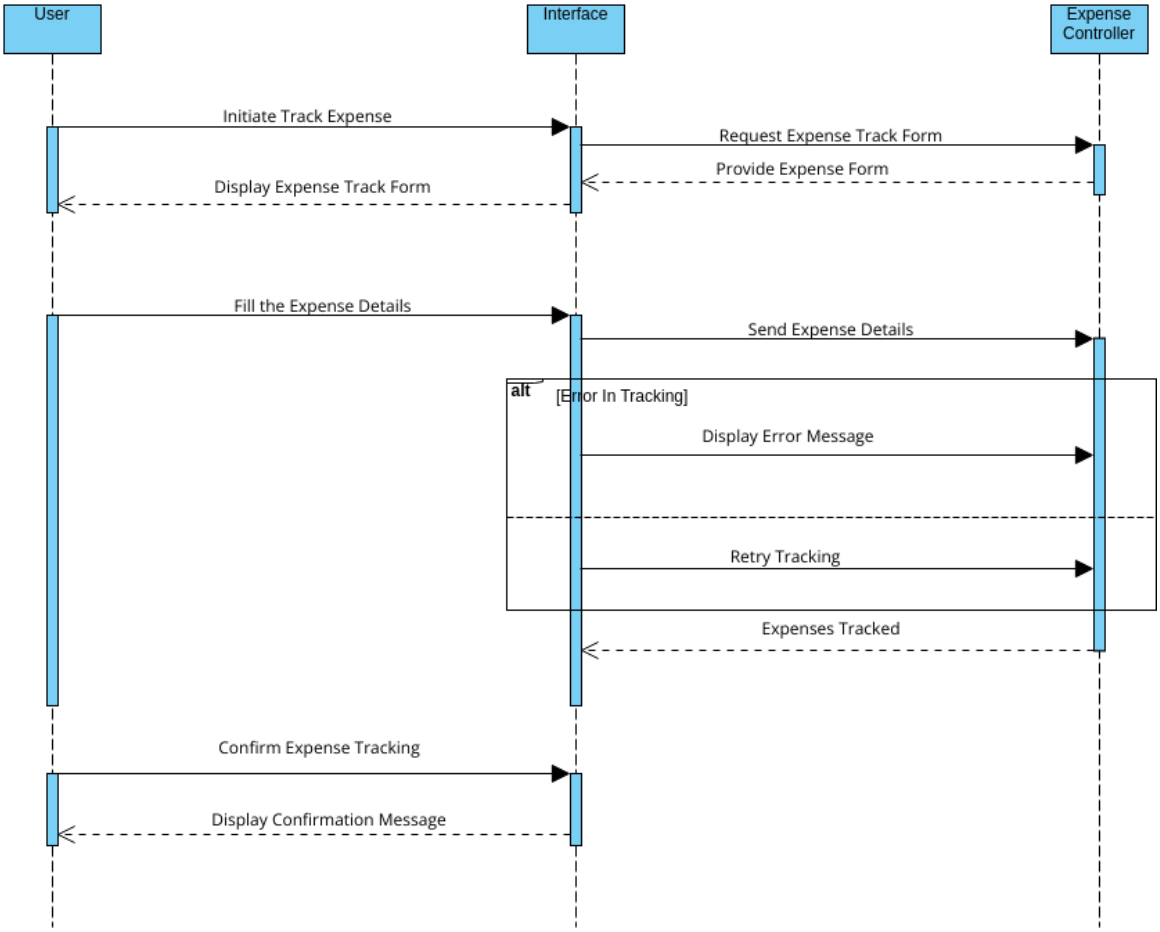

Fig.5.3: Sequence Diagram

4. **Activity Diagram:**
   Outlines the workflow of user activities. For example, the flow from logging in, selecting a category, entering expense details, and finally submitting the form to update the system.
5. **State Diagram:**
   Shows the state changes in the application—such as authentication status (Logged Out → Logging In → Logged In), or transitions in the expense status (New → Edited → Deleted).

5.2 Basic Modules

The application is composed of three fundamental modules:

- **Frontend:**
  Built using React.js, it serves as the client-side interface allowing users to interact with the system. Features include routing (React Router), state management (Context API or Redux), and form handling. Real-time updates and data visualization are implemented using Chart.js.
- **Backend:**
  Implemented in Node.js with Express.js, the backend handles all business logic and RESTful API routes. It manages user authentication, CRUD operations on expenses, and input validation.
- **Database:**
  MongoDB acts as the NoSQL database, providing flexibility in storing varied user data. Mongoose is used to define schemas and interact with the database efficiently.

5.3 Data Design

A carefully designed ER (Entity-Relationship) model is crucial for data consistency and efficient retrieval.

- **User Entity:**
  Represents the registered users. Each user has a unique ID, name, email, and password.
- **Expense Entity:**
  Contains individual financial records entered by the user. It stores the amount, category reference, date, and optional description.
- **Category Entity:**
  Predefined or user-defined categories like "Food", "Transport", "Entertainment", etc. Used to classify expenses for better analysis.
- **ER Relationships:**

  A User can have multiple Expenses.

  Each Expense belongs to a Category.

  Each Category can be used by multiple Expenses.

5.4 Project Structure

To maintain scalability and modularity, the project follows a well-defined structure:

- **MVC Architecture for Backend:**
  - **Model:** Mongoose schemas for defining User, Expense, and Category.
  - **View:** Though not directly applicable in an API, the frontend interacts through RESTful endpoints.

    - **Controller:** Handles business logic for each API route, separating concerns from route handlers.
    - **Routes:** Defines various endpoints like /api/expenses, /api/users.
- **Component-Based Frontend:**
    - React components are reusable, such as Navbar, LoginForm, ExpenseForm, Dashboard, and ExpenseList.
    - Each component is functional and styled using CSS modules or styled-components.

5.5 Schema Design

Here are the detailed Mongoose schema definitions for the major entities:

- **User Schema:**

  javascript

  Copy code

  ```
  {
    userId: ObjectId,
    name: String,
    email: String,
    password: String
  }
  ```

- **Expense Schema:**

  javascript

  Copy code

  ```
  {
    expenseId: ObjectId,
    userId: ObjectId, // Reference to User
    amount: Number,
    categoryId: ObjectId, // Reference to Category
    date: Date,
    description: String
  }
  ```

- **Category Schema:**

javascript

Copy code

```
{
 categoryId: ObjectId,
 name: String
}
```

These schemas help ensure data consistency and structure, supporting easy retrieval and manipulation of financial data.

5.6 Data Integrity and Constraints

- **Mongoose Validations:**
  - All fields are validated using Mongoose built-in validators.
  - email field in User schema is required and must be unique.
  - amount in Expense must be a positive number.
  - categoryId and userId are required and referenced from their respective collections.
- **Authentication Constraints:**
  - Passwords are hashed using bcrypt.
  - Secure sessions are managed using JWT tokens.
  - Middleware enforces route protection (e.g., only logged-in users can add or view expenses).

5.7 Procedural Design

- **REST API Architecture:** The application exposes the following core RESTful endpoints:
  - POST /api/register – Register a new user.
  - POST /api/login – Authenticate user and issue JWT.
  - GET /api/expenses – Retrieve all expenses of the logged-in user.
  - POST /api/expenses – Add a new expense.
  - PUT /api/expenses/:id – Edit an existing expense.
  - DELETE /api/expenses/:id – Remove an expense.
  - GET /api/categories – Fetch available categories.

These APIs follow CRUD operations and adhere to REST principles ensuring stateless, scalable interaction.

5.8 User Interface Design

- **Responsive Layout:**
  - Built using CSS Flexbox/Grid and media queries to ensure usability across devices.
  - Navigation bar, expense form, filters, and charts adjust dynamically based on screen size.
- **Real-Time Updates:**

    - React state updates allow real-time rendering of new expense entries without reloading.
    - Charts automatically update when new data is added.
    - User feedback is provided through toast notifications or alerts for every action (like successful login or expense added).
- **User Experience (UX) Enhancements:**
    - Form validation to prevent invalid data.
    - Skeleton loaders during data fetch.
    - Dark/light mode toggle for better accessibility.

CHAPTER 6

PROJECT OUTCOME

6.1 Implementation Approaches

The project was developed using a **bottom-up approach**, focusing on building and testing core backend modules (like database schemas and APIs) before layering the frontend.

e. **Backend-first Development:**

Data Model Design: The journey began with defining the primary data models, which included:

- User: Holds user information and authentication details.
- Expense: Captures details about each financial transaction, including amount, category, date, and user association.
- Category: Organizes expenses into logical groups (like food, transportation, entertainment) for effective management and visualization.

Implementation of RESTful APIs: Utilizing Express.js, we established RESTful API endpoints that allow for:

- Creating and retrieving user data.
- CRUD operations for expenses and categories.
- Secure user authentication processes.

Security Integration: Early integration of JSON Web Token (JWT) authentication provided a robust method for maintaining secure access to the application. Here, we used bcrypt for hashing passwords, ensuring that sensitive data would remain encrypted even if database vulnerabilities occurred.

2. **Modular API Structure:**

The architecture was designed to enhance maintainability and modularity by breaking down functionalities into independent modules:

- User Routes: Manage all user-related endpoints for registration, login, and profile management.
- Expense Routes: Handle operations related to expense entries, including creation, viewing, updating, and deletion.
- Middleware: This component manages authentication, error handling, and potential request preprocessing requirements.

Such a modular structure permits easy testing and upgrades. By isolating features, modifications can be executed with minimal disruptions to the overall system.

e. **Frontend Layering:**

Once the backend was fully operational, the frontend was layered incrementally:

- Component-Based Development: Utilizing React, components were developed progressively, starting with critical functions such as authentication and dashboard views.
- State Management: A centralized state management technique was employed (using React's Context API or libraries like Redux) to maintain predictable update patterns. This approach ensures that anytime state changes occur, the entire UI refreshes accordingly without unexpected side effects.
- Responsive UI Design: Each component designed followed mobile-first principles, ensuring that the application provides a consistent user experience across various devices.

6.2 Coding Details and Code Efficiency

1. Functional React Components:

All UI logic within the application was implemented utilizing modern functional components with React Hooks such as useState and useEffect. This strategy offers several advantages:

- Readability: Functional components are often easier to read and maintain compared to their class-based counterparts due to a more straightforward syntax.
- Reduced Boilerplate Code: The need for repetitive code blocks typically associated with class-based components—like the constructor and lifecycle methods—was eliminated. Instead, the functional approach focuses on leveraging hooks for component lifecycle management.

2. Optimized API Responses:

- Data Efficiency: All API endpoints were designed to return only essential JSON data, minimizing bandwidth use and optimizing load times.
- Efficient MongoDB Queries: To ensure rapid response times:
- Lean Queries: MongoDB's lean() method was employed to return simple JavaScript objects rather than MongoDB document instances, which enhances performance.

- Indexing: Key fields, such as user IDs and expense dates, were indexed to expedite retrieval operations.
- Pagination and Filtering: To manage potentially large datasets, expense data can be paginated, and optional filters (by date or category) are provided. This not only reduces client-side processing loads but also enhances user experience when navigating through extensive records.

e. **Error Handling:**

- Standardization: Consistent error responses across the application were established, returning relevant HTTP status codes to provide clear feedback on the outcome of API requests.
- Robust Frontend/Error Display: Both the frontend and backend were equipped to manage errors gracefully, ensuring users rec

4. Security Measures:

The security of user data was prioritized through numerous strategies:

- Password Security: User passwords are securely stored using bcrypt, making them resistant to brute-force attacks and unauthorized access.
- JWT Authentication: Middleware was implemented to verify JWT tokens for all protected routes, ensuring that access to sensitive functionalities is tightly controlled.

5. Preventative Security Policies:

- CORS Policy: Configured appropriately to restrict unauthorized domains from accessing the API.
- Input Sanitization: Essential for preventing common front-end vulnerabilities like Cross-Site Scripting (XSS) and Cross-Site Request Forgery (CSRF). All user-provided inputs undergo thorough sanitization.

6.3 Testing Approaches

Testing was performed both **manually** and using **automated tools** to ensure the reliability and stability of the system.

e. **Manual Testing:**

- User Interface Testing**:** Each feature underwent rigorous manual testing using the live UI to simulate realistic user interactions. Tests included checks for:
- Mobile Responsiveness: Ensuring all functionalities operate seamlessly on various screen sizes and devices.
- Form Validations: Validating that user inputs through forms adhere to the expected formats (e.g., numeric fields, valid date ranges).
- User Journey Simulation**:** Scenarios were developed that mimicked real-world use cases to ensure that all paths through the application functioned as intended.

2. API Testing with Postman:

- Endpoint Validation: Each API endpoint was tested for both valid and invalid inputs using Postman. This ensured that the server accurately handles various request types.

- Authenticated Requests: Ensured that valid JWT tokens lead to subscribed actions with successful responses.
- Unauthenticated Requests: Verified that requests without valid tokens receive appropriate errors.
- Simulated Error Scenarios: Testing included scenarios like:
  - Duplicate registration attempts to ensure graceful error management.
  - Incorrect login credentials verification to ensure security workflows remain intact.

3 Unit Testing (optional extension):

While not initially the project's core focus, the potential for integrating automated testing with frameworks like Jest and React Testing Library is acknowledged. Future development may include:

- Component Testing: Isolating and testing individual React components to ensure UI logic behaves as expected.
- Business Logic Testing: Testing for correctness in important backend functionalities to minimize potential regressions as new features are added.
- Continuous Integration (CI): Establishing a CI pipeline for automated testing on deployments, which guards against untested code entering production.

6.4 Test Cases Design

Representative test cases were prepared for major functionalities:

Test Case	Input	Expected Result	Status
Login Valid	Valid email and password	Redirect to dashboard	☑ Passed
Login Invalid	Wrong password	Show "Invalid credentials" error	☑ Passed
Add Expense	Valid amount, category, date	Expense is saved and listed	☑ Passed
Add Expense (Empty Field)	Empty amount	Show validation message	☑ Passed
Generate Reports	Select time range	Display correct chart data	☑ Passed
Register Duplicate	Existing email	Show "Email already exists"	☑ Passed
Unauthorized Access	Access/api/expenses without token	Return 401 error	☑ Passed

These tests ensured all core flows were covered, including edge cases.

6.5 Modifications and Improvements

After the initial build, feedback and usability analysis led to the following enhancements:

- **Chart Filters Added:**
  - Users can now filter expenses in the dashboard chart by category or date range.
  - Chart.js was reconfigured to dynamically re-render based on filters.
- **Error Messaging:**
  - Form-level validations were added to all input fields.
  - Global error handlers were implemented in the backend to catch and format database or auth-related errors.
- **UX/UI Tweaks:**
  - Toast notifications for success/failure feedback.
  - Loading spinners for API calls.
  - Improved mobile layout with collapsible menus.
- **Performance Optimization:**
  - Expense listing uses lazy loading for better performance with large datasets.
  - Backend caching was explored for repetitive queries.

6.6 Implementation of Code:

```
import React from ‘react’;

import ReactDOM from ‘react-dom/client’;

import App from ‘./App’;

import { GlobalProvider } from ‘./context/globalContext’;

import { GlobalStyle } from ‘./styles/GlobalStyle’;


const root = ReactDOM.createRoot(document.getElementById(‘root’));

root.render(

  <React.StrictMode>

    <GlobalStyle />

    <GlobalProvider>

      <App />

    </GlobalProvider>
```

```
  </React.StrictMode>
);

import React, {useState, useMemo} from 'react'
import styled from "styled-components";
import bg from './img/bg.png'
import {MainLayout} from './styles/Layouts'
import Orb from './Components/Orb/Orb'
import Navigation from './Components/Navigation/Navigation'
import Dashboard from './Components/Dashboard/Dashboard';
import Income from './Components/Income/Income'
import Expenses from './Components/Expenses/Expenses';
import { useGlobalContext } from './context/globalContext';

function App() {
  const [active, setActive] = useState(1)

  const global = useGlobalContext()
  console.log(global);

  const displayData = () => {
    switch(active){
      case 1:
        return <Dashboard />
      case 2:
        return <Dashboard />
      case 3:
```

```
        return <Income />
      case 4:
        return <Expenses />
      default:
        return <Dashboard />
    }
  }

  const orbMemo = useMemo(() => {
    return <Orb />
  },[])

  return (
    <AppStyled bg={bg} className=”App”>
      {orbMemo}
      <MainLayout>
        <Navigation active={active} setActive={setActive} />
        <main>
          {displayData()}
        </main>
      </MainLayout>
    </AppStyled>
  );
}

import React, { useEffect } from ‘react’
import styled from ‘styled-components’
```

```
import { useGlobalContext } from '../../context/globalContext';
import History from '../../History/History';
import { InnerLayout } from '../../styles/Layouts';
import { dollar } from '../../utils/Icons';
import Chart from '../Chart/Chart';

function Dashboard() {
    const {totalExpenses,incomes, expenses, totalIncome, totalBalance, getIncomes, getExpenses } = useGlobalContext()

  useEffect(() => {
    getIncomes()
    getExpenses()
  }, [])

  return (
    <DashboardStyled>
      <InnerLayout>
        <h1>All Transactions</h1>
        <div className="stats-con">
          <div className="chart-con">
            <Chart />
            <div className="amount-con">
              <div className="income">
                <h2>Total Income</h2>
                <p>
                  {dollar} {totalIncome()}
```

```
            </p>
        </div>
        <div className=”expense”>
            <h2>Total Expense</h2>
            <p>
                {dollar} {totalExpenses()}
            </p>
        </div>
        <div className=”balance”>
            <h2>Total Balance</h2>
            <p>
                {dollar} {totalBalance()}
            </p>
        </div>
    </div>
</div>
<div className=”history-con”>
    <History />
    <h2 className=”salary-title”>Min <span>Salary</span>Max</h2>
    <div className=”salary-item”>
        <p>
            ${Math.min(…incomes.map(item => item.amount))}
        </p>
        <p>
            ${Math.max(…incomes.map(item => item.amount))}
        </p>
    </div>
```

```
                    <h2 className=”salary-title”>Min <span>Expense</span>Max</h2>
                    <div className=”salary-item”>
                        <p>
                            ${Math.min(…expenses.map(item => item.amount))}
                        </p>
                        <p>
                            ${Math.max(…expenses.map(item => item.amount))}
                        </p>
                    </div>
                </div>
            </div>
        </InnerLayout>
    </DashboardStyled>
  )
}

const DashboardStyled = styled.div`
  .stats-con{
    display: grid;
    grid-template-columns: repeat(5, 1fr);
    gap: 2rem;
    .chart-con{
      grid-column: 1 / 4;
      height: 400px;
      .amount-con{
        display: grid;
        grid-template-columns: repeat(4, 1fr);
```

```
gap: 2rem;
margin-top: 2rem;
.income, .expense{
    grid-column: span 2;
}
.income, .expense, .balance{
    background: #FCF6F9;
    border: 2px solid #FFFFFF;
    box-shadow: 0px 1px 15px rgba(0, 0, 0, 0.06);
    border-radius: 20px;
    padding: 1rem;
    p{
        font-size: 3.5rem;
        font-weight: 700;
    }
}

.balance{
    grid-column: 2 / 4;
    display: flex;
    flex-direction: column;
    justify-content: center;
    align-items: center;
    p{
        color: var(--color-green);
        opacity: 0.6;
        font-size: 4.5rem;
```

```
            }
        }
    }
}

.history-con{
    grid-column: 4 / -1;
    h2{
        margin: 1rem 0;
        display: flex;
        align-items: center;
        justify-content: space-between;
    }
    .salary-title{
        font-size: 1.2rem;
        span{
            font-size: 1.8rem;
        }
    }
    .salary-item{
        background: #FCF6F9;
        border: 2px solid #FFFFFF;
        box-shadow: 0px 1px 15px rgba(0, 0, 0, 0.06);
        padding: 1rem;
        border-radius: 20px;
        display: flex;
        justify-content: space-between;
```

```
            align-items: center;
            p{
                font-weight: 600;
                font-size: 1.6rem;
            }
        }
    }
}
`;
import React from 'react'
import styled from 'styled-components'

function Button({name, icon, onClick, bg, bPad, color, bRad}) {
    return (
        <ButtonStyled style={{
            background: bg,
            padding: bPad,
            borderRadius: bRad,
            color: color,
        }} onClick={onClick}>
            {icon}
            {name}
        </ButtonStyled>
    )
}

const ButtonStyled = styled.button`
```

```
    outline: none;
    border: none;
    font-family: inherit;
    font-size: inherit;
    display: flex;
    align-items: center;
    gap: .5rem;
    cursor: pointer;
    transition: all .4s ease-in-out;
`;

import React from 'react'
import {Chart as ChartJs,
    CategoryScale,
    LinearScale,
    PointElement,
    LineElement,
    Title,
    Tooltip,
    Legend,
    ArcElement,
} from 'chart.js'

import {Line} from 'react-chartjs-2'
import styled from 'styled-components'
import { useGlobalContext } from '../../context/globalContext'
import { dateFormat } from '../../utils/dateFormat'
```

```
ChartJs.register(
    CategoryScale,
    LinearScale,
    PointElement,
    LineElement,
    Title,
    Tooltip,
    Legend,
    ArcElement,
)

function Chart() {
    const {incomes, expenses} = useGlobalContext()

    const data = {
        labels: incomes.map((inc) =>{
            const {date} = inc
            return dateFormat(date)
        }),
        datasets: [
            {
                label: 'Income',
                data: [
                    …incomes.map((income) => {
                        const {amount} = income
                        return amount
```

```
            })
          ],
          backgroundColor: 'green',
          tension: .2
        },
        {
          label: 'Expenses',
          data: [
            …expenses.map((expense) => {
              const {amount} = expense
              return amount
            })
          ],
          backgroundColor: 'red',
          tension: .2
        }
      ]
    }

    return (
      <ChartStyled >
        <Line data={data} />
      </ChartStyled>
    )
}

const ChartStyled = styled.div`
```

```
    background: #FCF6F9;
    border: 2px solid #FFFFFF;
    box-shadow: 0px 1px 15px rgba(0, 0, 0, 0.06);
    padding: 1rem;
    border-radius: 20px;
    height: 100%;
`;
import React, { useState } from 'react'
import styled from 'styled-components'
import DatePicker from 'react-datepicker'
import "react-datepicker/dist/react-datepicker.css";
import { useGlobalContext } from '../../context/globalContext';
import Button from '../Button/Button';
import { plus } from '../../utils/Icons';

function ExpenseForm() {
    const {addExpense, error, setError} = useGlobalContext()
    const [inputState, setInputState] = useState({
        title: '',
        amount: '',
        date: '',
        category: '',
        description: '',
    })

    const { title, amount, date, category,description } = inputState;
```

```
const handleInput = name => e => {
    setInputState({…inputState, [name]: e.target.value})
    setError(‘’)
}

const handleSubmit = e => {
    e.preventDefault()
    addExpense(inputState)
    setInputState({
        title: ‘’,
        amount: ‘’,
        date: ‘’,
        category: ‘’,
        description: ‘’,
    })
}

return (
    <ExpenseFormStyled onSubmit={handleSubmit}>
        {error && <p className=’error’>{error}</p>}
        <div className=”input-control”>
            <input
                type=”text”
                value={title}
                name={‘title’}
                placeholder=”Expense Title”
                onChange={handleInput(‘title’)}
```

```
    />
  </div>
  <div className="input-control">
    <input value={amount}
      type="text"
      name={'amount'}
      placeholder={'Expense Amount'}
      onChange={handleInput('amount')}
    />
  </div>
  <div className="input-control">
    <DatePicker
      id='date'
      placeholderText='Enter A Date'
      selected={date}
      dateFormat="dd/MM/yyyy"
      onChange={(date) => {
        setInputState({...inputState, date: date})
      }}
    />
  </div>
  <div className="selects input-control">
      <select required value={category} name="category" id="category" onChange={handleInput('category')}>
      <option value="" disabled >Select Option</option>
      <option value="education">Education</option>
      <option value="groceries">Groceries</option>
```

```
                <option value="health">Health</option>
                <option value="subscriptions">Subscriptions</option>
                <option value="takeaways">Takeaways</option>
                <option value="clothing">Clothing</option>
                <option value="travelling">Travelling</option>
                <option value="other">Other</option>
            </select>
        </div>
        <div className="input-control">
            <textarea name="description" value={description} placeholder='Add A Reference' id="description" cols="30" rows="4" onChange={handleInput('description')}></textarea>
        </div>
        <div className="submit-btn">
            <Button
                name={'Add Expense'}
                icon={plus}
                bPad={'.8rem 1.6rem'}
                bRad={'30px'}
                bg={'var(--color-accent'}
                color={'#fff'}
            />
        </div>
    </ExpenseFormStyled>
  )
}

const ExpenseFormStyled = styled.form`
```

```
display: flex;
flex-direction: column;
gap: 2rem;
input, textarea, select{
    font-family: inherit;
    font-size: inherit;
    outline: none;
    border: none;
    padding: .5rem 1rem;
    border-radius: 5px;
    border: 2px solid #fff;
    background: transparent;
    resize: none;
    box-shadow: 0px 1px 15px rgba(0, 0, 0, 0.06);
    color: rgba(34, 34, 96, 0.9);
    &::placeholder{
        color: rgba(34, 34, 96, 0.4);
    }
}
.input-control{
    input{
        width: 100%;
    }
}

.selects{
    display: flex;
```

```
        justify-content: flex-end;
        select{
            color: rgba(34, 34, 96, 0.4);
            &:focus, &:active{
                color: rgba(34, 34, 96, 1);
            }
        }
    }

    .submit-btn{
        button{
            box-shadow: 0px 1px 15px rgba(0, 0, 0, 0.06);
            &:hover{
                background: var(--color-green) !important;
            }
        }
    }
`;
import React, { useState } from 'react'
import styled from 'styled-components'
import DatePicker from 'react-datepicker'
import "react-datepicker/dist/react-datepicker.css";
import { useGlobalContext } from '../../context/globalContext';
import Button from '../Button/Button';
import { plus } from '../../utils/Icons';

function Form() {
```

```
const { addIncome, getIncomes, error, setError } = useGlobalContext()
const [inputState, setInputState] = useState({
    title: ‘’,
    amount: ‘’,
    date: ‘’,
    category: ‘’,
    description: ‘’,
})

const { title, amount, date, category, description } = inputState;

const handleInput = name => e => {
    setInputState({ …inputState, [name]: e.target.value })
    setError(‘’)
}

const handleSubmit = e => {
    e.preventDefault()
    addIncome(inputState)
    setInputState({
        title: ‘’,
        amount: ‘’,
        date: ‘’,
        category: ‘’,
        description: ‘’,
    })
}
```

```
return (
   <FormStyled onSubmit={handleSubmit}>
      {error && <p className='error'>{error}</p>}
      <div className="input-control">
         <input
            type="text"
            value={title}
            name={'title'}
            placeholder="Salary Title"
            onChange={handleInput('title')}
         />
      </div>
      <div className="input-control">
         <input value={amount}
            type="text"
            name={'amount'}
            placeholder={'Salary Amount'}
            onChange={handleInput('amount')}
         />
      </div>
      <div className="input-control">
         <DatePicker
            id='date'
            placeholderText='Enter A Date'
            selected={date}
            dateFormat="dd/MM/yyyy"
```

```
            onChange={(date) => {
              setInputState({ …inputState, date: date })
            }}
          />
        </div>
        <div className="selects input-control">
          <select required value={category} name="category" id="category" onChange={handleInput('category')}>
            <option value="" disabled >Select Option</option>
            <option value="salary">Salary</option>
            <option value="freelancing">Freelancing</option>
            <option value="investments">Investiments</option>
            <option value="stocks">Stocks</option>
            <option value="bitcoin">Bitcoin</option>
            <option value="bank">Bank Transfer</option>
            <option value="youtube">Youtube</option>
            <option value="other">Other</option>
          </select>
        </div>
        <div className="input-control">
          <textarea name="description" value={description} placeholder='Add A Reference' id="description" cols="30" rows="4" onChange={handleInput('description')}></textarea>
        </div>
        <div className="submit-btn">
          <Button
            name={'Add Income'}
            icon={plus}
```

```
                bPad={'.8rem 1.6rem'}
                bRad={'30px'}
                bg={'var(--color-accent'}
                color={'#fff'}
              />
            </div>
        </FormStyled>
    )
}

const FormStyled = styled.form`
    display: flex;
    flex-direction: column;
    gap: 2rem;
    input, textarea, select{
        font-family: inherit;
        font-size: inherit;
        outline: none;
        border: none;
        padding: .5rem 1rem;
        border-radius: 5px;
        border: 2px solid #fff;
        background: transparent;
        resize: none;
        box-shadow: 0px 1px 15px rgba(0, 0, 0, 0.06);
        color: rgba(34, 34, 96, 0.9);
        &::placeholder{
```

```
        color: rgba(34, 34, 96, 0.4);
    }
}
.input-control{
    input{
        width: 100%;
    }
}

.selects{
    display: flex;
    justify-content: flex-end;
    select{
        color: rgba(34, 34, 96, 0.4);
        &:focus, &:active{
            color: rgba(34, 34, 96, 1);
        }
    }
}

.submit-btn{
    button{
        box-shadow: 0px 1px 15px rgba(0, 0, 0, 0.06);
        &:hover{
            background: var(--color-green) !important;
        }
    }
```

```
    }
`;
import React, { useEffect } from 'react'
import styled from 'styled-components'
import { useGlobalContext } from '../../context/globalContext';
import { InnerLayout } from '../../styles/Layouts';
import Form from '../Form/Form';
import IncomeItem from '../IncomeItem/IncomeItem';

function Income() {
        const {addIncome,incomes, getIncomes, deleteIncome, totalIncome} = useGlobalContext()

    useEffect(() =>{
        getIncomes()
    }, [])
    return (
        <IncomeStyled>
            <InnerLayout>
                <h1>Incomes</h1>
                                        <h2 className="total-income">Total Income: <span>${totalIncome()}</span></h2>
                <div className="income-content">
                    <div className="form-container">
                        <Form />
                    </div>
                    <div className="incomes">
                        {incomes.map((income) => {
```

```
                    const {_id, title, amount, date, category, description, type} = income;
                    return <IncomeItem
                        key={_id}
                        id={_id}
                        title={title}
                        description={description}
                        amount={amount}
                        date={date}
                        type={type}
                        category={category}
                        indicatorColor="var(--color-green)"
                        deleteItem={deleteIncome}
                    />
                })}
            </div>
        </div>
    </InnerLayout>
</IncomeStyled>
)
}

const IncomeStyled = styled.div`
    display: flex;
    overflow: auto;
    .total-income{
        display: flex;
        justify-content: center;
```

```
        align-items: center;
        background: #FCF6F9;
        border: 2px solid #FFFFFF;
        box-shadow: 0px 1px 15px rgba(0, 0, 0, 0.06);
        border-radius: 20px;
        padding: 1rem;
        margin: 1rem 0;
        font-size: 2rem;
        gap: .5rem;
        span{
            font-size: 2.5rem;
            font-weight: 800;
            color: var(--color-green);
        }
    }
    .income-content{
        display: flex;
        gap: 2rem;
        .incomes{
            flex: 1;
        }
    }
`;

import {createGlobalStyle} from 'styled-components'

export const GlobalStyle = createGlobalStyle`
```

```
*{
    margin: 0;
    padding: 0;
    box-sizing: border-box;
    list-style: none;
}

:root{
    --primary-color: #222260;
    --primary-color2: ‘color: rgba(34, 34, 96, .6)’;
    --primary-color3: ‘color: rgba(34, 34, 96, .4)’;
    --color-green: #42AD00;
    --color-grey: #aaa;
    --color-accent: #F56692;
    --color-delete: #FF0000;
}

body{
    font-family: ‘Nunito’, sans-serif;
    font-size: clamp(1rem, 1.5vw, 1.2rem);
    overflow: hidden;
    color: rgba(34, 34, 96, .6);
}

h1, h2, h3, h4, h5, h6{
    color: var(--primary-color);
}
```

```
    .error{
        color: red;
        animation: shake 0.5s ease-in-out;
        @keyframes shake {
            0%{
                transform: translateX(0);
            }
            25%{
                transform: translateX(10px);
            }
            50%{
                transform: translateX(-10px);
            }
            75%{
                transform: translateX(10px);
            }
            100%{
                transform: translateX(0);
            }
        }
    }
`;
import styled from “styled-components”;

export const MainLayout = styled.div`
    padding: 2rem;
```

```
    height: 100%;
    display: flex;
    gap: 2rem;
`;

export const InnerLayout = styled.div`
    padding: 2rem 1.5rem;
    width: 100%;
`;
import { useEffect, useState } from "react"

export const useWindowSize = () =>{
    const [size, setSize] = useState([window.innerWidth, window.innerHeight])

    useEffect(() => {
        const updateSize = () => {
            setSize([window.innerWidth, window.innerHeight])
        }
        window.addEventListener('resize', updateSize)

        return () => window.removeEventListener('resize', updateSize)
    }, [])

    return {
        width: size[0],
        height: size[1]
    }
```

```
}
import React from ‘react’
import styled, { keyframes } from ‘styled-components’
import { useWindowSize } from ‘../../utils/useWindowSize’;

function Orb() {

   const {width, height} = useWindowSize()

   console.log(width, height)

   const moveOrb = keyframes`
      0%{
         transform: translate(0, 0);
      }
      50%{
         transform: translate(${width}px, ${height/2}px);
      }
      100%{
         transform: translate(0, 0);
      }
   `

   const OrbStyled = styled.div`
      width: 70vh;
      height: 70vh;
      position: absolute;
```

```
        border-radius: 50%;
        margin-left: -37vh;
        margin-top: -37vh;
        background: linear-gradient(180deg, #F56692 0%, #F2994A 100%);
        filter: blur(400px);
        animation: ${moveOrb} 15s alternate linear infinite;
    `;

    return (
        <OrbStyled></OrbStyled>
    )
}
import React, { useContext, useState } from "react"
import axios from 'axios'

const BASE_URL = "http://localhost:5000/api/v1/";

const GlobalContext = React.createContext()

export const GlobalProvider = ({ children }) => {

    const [incomes, setIncomes] = useState([])
    const [expenses, setExpenses] = useState([])
    const [error, setError] = useState(null)

    //calculate incomes
    const addIncome = async (income) => {
```

```
    const response = await axios.post(`${BASE_URL}add-income`, income)
        .catch((err) => {
            setError(err.response.data.message)
        })
    getIncomes()
}

const getIncomes = async () => {
    const response = await axios.get(`${BASE_URL}get-incomes`)
    setIncomes(response.data)
    console.log(response.data)
}

const deleteIncome = async (id) => {
    const res = await axios.delete(`${BASE_URL}delete-income/${id}`)
    getIncomes()
}

const totalIncome = () => {
    let totalIncome = 0;
    incomes.forEach((income) => {
        totalIncome = totalIncome + income.amount
    })

    return totalIncome;
}
```

```
//calculate incomes
const addExpense = async (income) => {
    const response = await axios.post(`${BASE_URL}add-expense`, income)
        .catch((err) => {
            setError(err.response.data.message)
        })
    getExpenses()
}

const getExpenses = async () => {
    const response = await axios.get(`${BASE_URL}get-expenses`)
    setExpenses(response.data)
    console.log(response.data)
}

const deleteExpense = async (id) => {
    const res = await axios.delete(`${BASE_URL}delete-expense/${id}`)
    getExpenses()
}

const totalExpenses = () => {
    let totalIncome = 0;
    expenses.forEach((income) => {
        totalIncome = totalIncome + income.amount
    })

    return totalIncome;
```

```
}

const totalBalance = () => {
    return totalIncome() – totalExpenses()
}

const transactionHistory = () => {
    const history = […incomes, …expenses]
    history.sort((a, b) => {
        return new Date(b.createdAt) – new Date(a.createdAt)
    })

    return history.slice(0, 3)
}

return (
    <GlobalContext.Provider value={{
        addIncome,
        getIncomes,
        incomes,
        deleteIncome,
        expenses,
        totalIncome,
        addExpense,
        getExpenses,
        deleteExpense,
        totalExpenses,
```

```
            totalBalance,
            transactionHistory,
            error,
            setError
        }}>
            {children}
        </GlobalContext.Provider>
    )
}

export const useGlobalContext = () => {
    return useContext(GlobalContext)
}
import React from 'react'
import styled from 'styled-components'
import { useGlobalContext } from '../context/globalContext';

function History() {
    const {transactionHistory} = useGlobalContext()

    const […history] = transactionHistory()

    return (
        <HistoryStyled>
            <h2>Recent History</h2>
            {history.map((item) =>{
                const {_id, title, amount, type} = item
```

```
            return (
                <div key={_id} className="history-item">
                    <p style={{
                        color: type === 'expense' ? 'red' : 'var(--color-green)'
                    }}>
                        {title}
                    </p>

                    <p style={{
                        color: type === 'expense' ? 'red' : 'var(--color-green)'
                    }}>
                        {
                            type === 'expense' ? `-${amount <= 0 ? 0 : amount}` : `+${amount <= 0 ? 0: amount}`
                        }
                    </p>
                </div>
            )
        })}
    </HistoryStyled>
  )
}

const HistoryStyled = styled.div`
  display: flex;
  flex-direction: column;
  gap: 1rem;
```

```
    .history-item{
        background: #FCF6F9;
        border: 2px solid #FFFFFF;
        box-shadow: 0px 1px 15px rgba(0, 0, 0, 0.06);
        padding: 1rem;
        border-radius: 20px;
        display: flex;
        justify-content: space-between;
        align-items: center;
    }
`;
import React, { useEffect } from 'react'
import styled from 'styled-components'
import { useGlobalContext } from '../../context/globalContext';
import { InnerLayout } from '../../styles/Layouts';
import Form from '../Form/Form';
import IncomeItem from '../IncomeItem/IncomeItem';
import ExpenseForm from './ExpenseForm';

function Expenses() {
        const {addIncome,expenses, getExpenses, deleteExpense, totalExpenses} = useGlobalContext()

    useEffect(() =>{
        getExpenses()
    }, [])
    return (
```

```
<ExpenseStyled>
  <InnerLayout>
    <h1>Expenses</h1>
                                    <h2 className="total-income">Total Expense: <span>${totalExpenses()}</span></h2>
    <div className="income-content">
      <div className="form-container">
        <ExpenseForm />
      </div>
      <div className="incomes">
        {expenses.map((income) => {
          const {_id, title, amount, date, category, description, type} = income;
          console.log(income)
          return <IncomeItem
            key={_id}
            id={_id}
            title={title}
            description={description}
            amount={amount}
            date={date}
            type={type}
            category={category}
            indicatorColor="var(--color-green)"
            deleteItem={deleteExpense}
          />
        })}
      </div>
```

```
            </div>
        </InnerLayout>
    </ExpenseStyled>
  )
}

const ExpenseStyled = styled.div`
  display: flex;
  overflow: auto;
  .total-income{
    display: flex;
    justify-content: center;
    align-items: center;
    background: #FCF6F9;
    border: 2px solid #FFFFFF;
    box-shadow: 0px 1px 15px rgba(0, 0, 0, 0.06);
    border-radius: 20px;
    padding: 1rem;
    margin: 1rem 0;
    font-size: 2rem;
    gap: .5rem;
    span{
      font-size: 2.5rem;
      font-weight: 800;
      color: var(--color-green);
    }
  }
```

```
    .income-content{
        display: flex;
        gap: 2rem;
        .incomes{
            flex: 1;
        }
    }
`;

const mongoose = require('mongoose');

const db = async () => {
    try {
        mongoose.set('strictQuery', false)
        await mongoose.connect(process.env.MONGO_URL)
        console.log('Db Connected')
    } catch (error) {
        console.log('DB Connection Error');
    }
}

module.exports = {db}
const mongoose = require('mongoose');

const ExpenseSchema = new mongoose.Schema({
    title: {
        type: String,
```

```
    required: true,
    trim: true,
    maxLength: 50
},
amount: {
    type: Number,
    required: true,
    maxLength: 20,
    trim: true
},
type: {
    type: String,
    default:”expense”
},
date: {
    type: Date,
    required: true,
    trim: true
},
category: {
    type: String,
    required: true,
    trim: true
},
description: {
    type: String,
    required: true,
```

```
        maxLength: 20,
        trim: true
    },
}, {timestamps: true})

module.exports = mongoose.model('Expense', ExpenseSchema)

const mongoose = require('mongoose');

const IncomeSchema = new mongoose.Schema({
    title: {
        type: String,
        required: true,
        trim: true,
        maxLength: 50
    },
    amount: {
        type: Number,
        required: true,
        maxLength: 20,
        trim: true
    },
    type: {
        type: String,
        default:"income"
    },
    date: {
```

```
        type: Date,
        required: true,
        trim: true
    },
    category: {
        type: String,
        required: true,
        trim: true
    },
    description: {
        type: String,
        required: true,
        maxLength: 20,
        trim: true
    },
}, {timestamps: true})

module.exports = mongoose.model('Income', IncomeSchema)
```

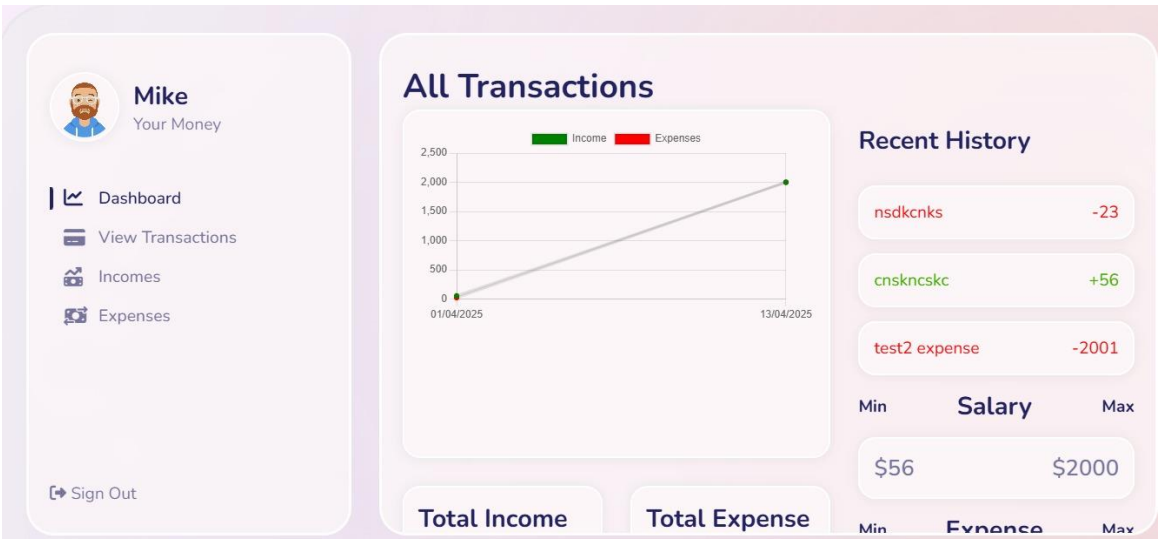

Fig.6.1: Website Dashboard

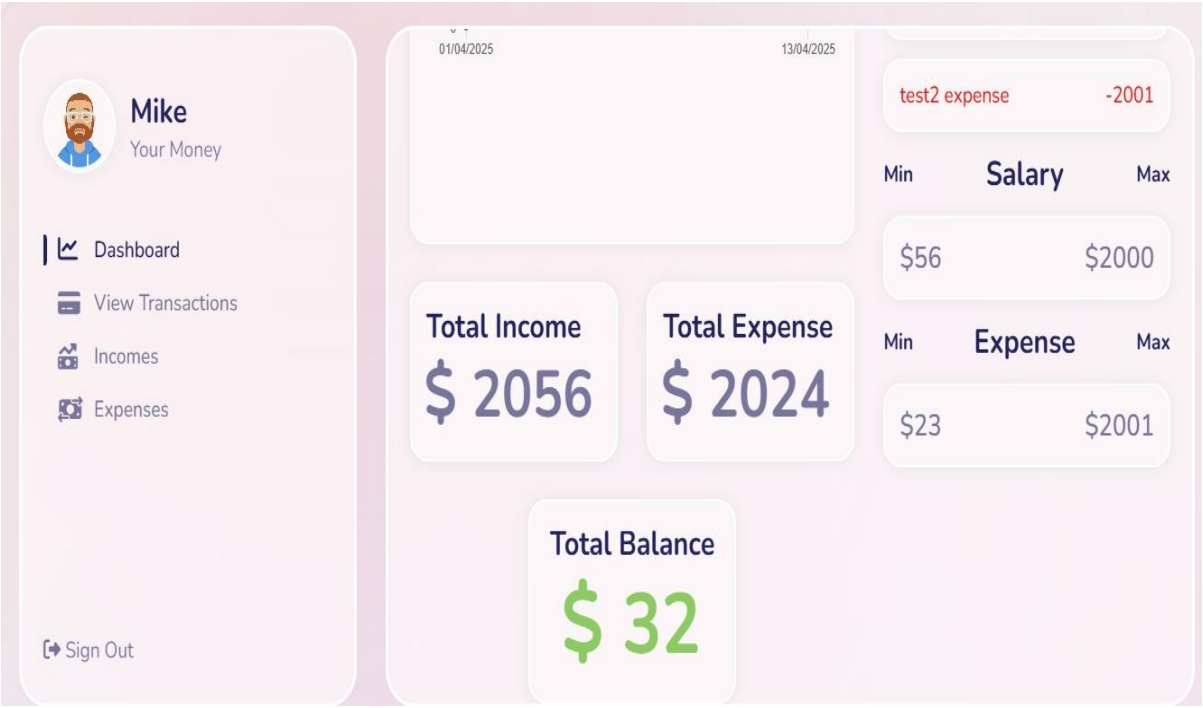

Fig.6.2: Website Dashboard

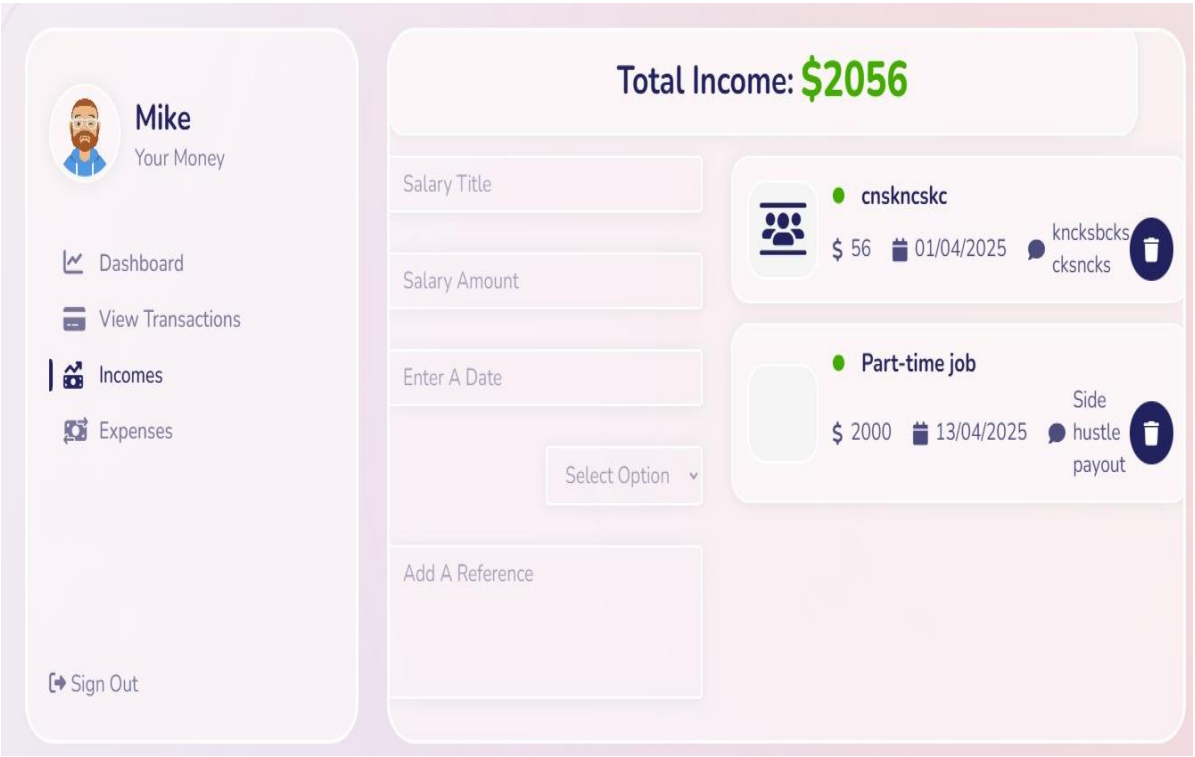

Fig.6.3: Income page

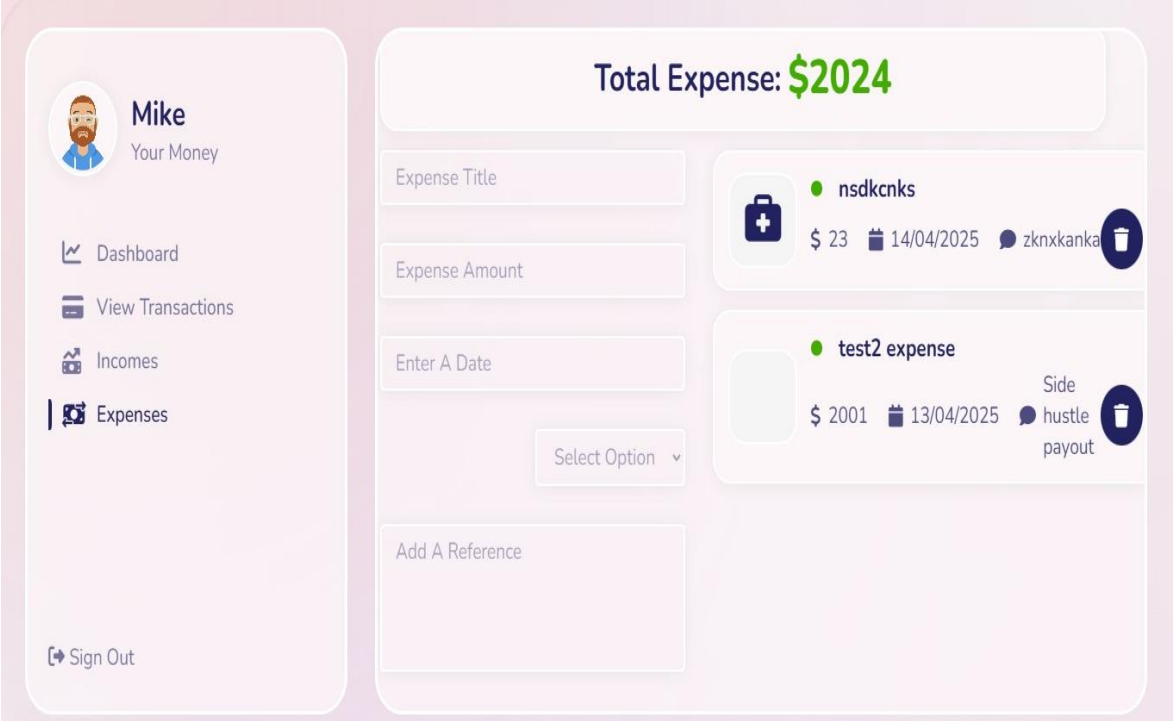

Fig.6.4: Expenses page

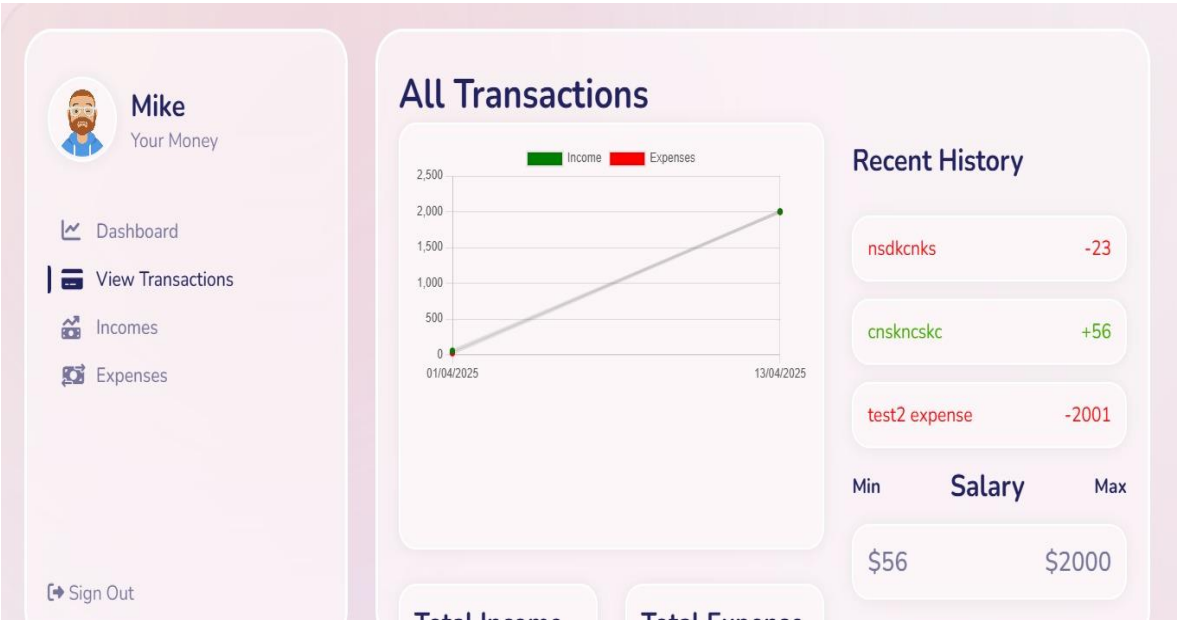

Fig.6.5: View Transaction

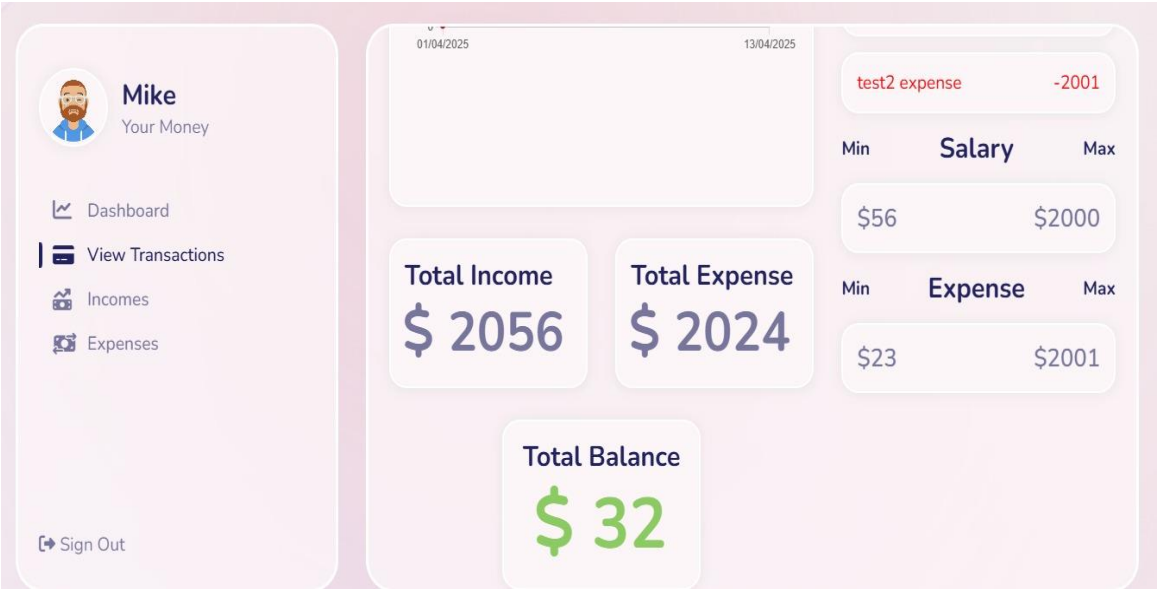

Fig.6.6: View Transaction

CHAPTER 7

REFERENCES AND BIBLIOGRAPHY

7.1 Books

- Kiyosaki, Robert T. *Rich Dad Poor Dad: What the Rich Teach Their Kids About Money That the Poor and the Middle Class Do Not!* New York: Warner Books, 1997.
- Ramsey, Dave. *The Total Money Makeover: A Proven Plan for Financial Fitness.* Nashville: Thomas Nelson, 2003.
- Orman, Suze. *The Money Book for the Young, Fabulous & Broke.* New York: Spiegel & Grau, 2005.

7.2 Articles

- Bittman, Mark. "How to Track Your Spending." *The New York Times*, 15 March 2021. Link
- Smith, John. "The Importance of Budgeting: How to Take Control of Your Finances." *Forbes*, 22 June 2022. Link
- Johnson, Emily. "Top 10 Expense Tracking Apps for 2023." *TechCrunch*, 10 January 2023. Link

7.3 Websites

- Mint. "Budgeting Basics: How to Create a Budget." Link
- NerdWallet. "How to Create a Budget: A Step-by-Step Guide." Link
- Pocket Guard. "The Ultimate Guide to Budgeting." Link

7.4 Research Papers

- Lusardi, Annamaria, and Olivia S. Mitchell. "Financial Literacy and Planning: Implications for Retirement Wellbeing." *The Pension Research Council*, 2014. Link

- Chen, H., & Volpe, R. P. "An Analysis of Personal Financial Literacy Among College Students." *Financial Services Review*, 2002.

- Hastings, Justine, and Olivia S. Mitchell. "How Financial Literacy Affects Household Wealth." The Journal of Consumer Affairs, vol. 44, no. 2, 2010, pp. 207-223. Link

7.5 Reports

- National Endowment for Financial Education. "The Financial Literacy of Young Americans: A National Survey." 2020. Link

- Consumer Financial Protection Bureau. "Financial Well-Being: The Goal of Financial Education." 2015. Link

- Federal Reserve. "Report on the Economic Well-Being of U.S. Households in 2020." 2021. Link