VIETNAM NATIONAL UNIVERSITY OF HOCHIMINH CITY
THE INTERNATIONAL UNIVERSITY
SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

WEB APPLICATION DEVELOPMENT

BUDGET MANAGEMENT SYSTEM (IHATEMONEY.LIVE)

By
Dang Nhat Huy - ITITIU20043
Ho Huu Hiep - ITITIU20202

A report submitted to Assoc. Prof. Nguyen Van Sinh
in partial fulfillment of the requirements for the assignment of
Web Application Development course in Semester 2 (2023 - 2024)

Ho Chi Minh City, Vietnam
2024

BUDGET MANAGEMENT SYSTEM (IHATEMONEY.LIVE)

APPROVED BY:

_____,

Nguyen Van Sinh, Assoc. Prof., Chair

_____ ,

Tran Khai Minh, MSc., Lab Lecturer

_____ ,

Dang Nhat Huy, Student, ITITIU20043

_____ .

Ho Huu Hiep, Student, ITITIU20202

ACKNOWLEDGMENTS

We extend our heartfelt thanks to Assoc. Prof. Nguyen Van Sinh for his invaluable theoretical guidance and MSc. Tran Khai Minh for his adept mentoring in the laboratory. Their expertise and support have been fundamental to our studies.

TABLE OF CONTENTS

ACKNOWLEDGMENTS **3**
TABLE OF CONTENTS **4**
LIST OF FIGURES **6**
LIST OF TABLES **7**
ABSTRACT **8**
CHAPTER 1 ..... **9**
INTRODUCTION **9**
1.1. Motivation 9
1.2. Problem Statement 9
1.3. Scope and Objectives 11
1.3.1. Scope 11
1.3.2. Objectives 11
1.4. Assumptions and Solutions 12
1.4.1. Assumptions 12
1.4.2. Solutions 13
1.5. Structure of report 14
CHAPTER 2 ..... **15**
RELATED WORK & COMPARISON **15**
2.1. Related Work 15
2.1.1. Mint 15
2.1.2. You Need a Budget (YNAB) 16
2.1.3. PocketGuard 17
2.2. Comparison 18
2.3. Background 19
2.3.1. Front-End Technologies 19
2.3.2. Back-End Technologies 20
2.3.3. Database Management 21
2.3.4. Authentication and Security 21
2.3.5. Data Visualization 22
2.3.6. Deployment 22
CHAPTER 3 ..... **23**
METHODOLOGY **23**
3.1. Requirement analysis 23
3.1.1. Functional requirements 23
3.1.2. Non-functional requirements 26
3.1.3. Use Case diagram 29
3.2. System Design 32
3.2.1. Activity Diagram 32
3.2.2. Class Diagram 34

3.2.3. Entity Relationship Diagram....36
CHAPTER 4....39
IMPLEMENT AND RESULTS....39
4.1. Implement....39
4.1.1. Implement View....39
4.1.2. Implement Data....41
4.1.3. Implement Security....42
4.2. Results....45
CHAPTER 5....52
DISCUSSION AND EVALUATION....52
5.1. Discussion....52
5.2. Comparison....53
5.3. Evaluation....54
CHAPTER 6....55
CONCLUSION AND FUTURE WORK....55
6.1. Conclusion....55
6.2. Future work....55
REFERENCES....57
APPENDIX....58

LIST OF FIGURES

Figure 1. Some inspirations..... 10
Figure 3. Mint..... 15
Figure 5. PocketGuard..... 17
Figure 6. Front-end Technologies..... 20
Figure 7. Back-end Technologies..... 21
Figure 9. Spring Security..... 21
Figure 10. Visulaize Tools..... 22
Figure 11. Microsoft Azure..... 22
Figure 12. User Use Case Diagram..... 29
Figure 13. Admin Use Case Diagram..... 30
Figure 15. Class diagram..... 34
Figure 16. ER Diagram..... 36
Figure 17. View Directory..... 39
Figure 18. Data Directory..... 41
Figure 19. Security directory..... 42
Figure 20. Customize the security settings..... 43
Figure 21. Login Page..... 45
Figure 22. Forgot Password page..... 46
Figure 23. Sample Email Received..... 46
Figure 24. Renew Password page..... 47
Figure 25. Register page..... 47
Figure 26. Dashboard view..... 48
Figure 27. Transactions view..... 48
Figure 28. Update a Transaction..... 49
Figure 29. Budgets view..... 49
Figure 30. Income vs. Expenses view..... 50
Figure 31. Export Data view..... 50
Figure 32. Settings view..... 51
Figure 33. Log out button..... 51

LIST OF TABLES

Table 1. Comparison between existing tools.....18
Table 2. Functional Requirement: User Authentication System.....23
Table 3. Functional Requirement: Budget Management Interface.....24
Table 4. Functional Requirement: Transaction Recording and Categorization.....24
Table 5. Functional Requirement: Advanced Financial Reporting and Analytics.....25
Table 6. Functional Requirement: Automated Data Backup and Recovery System.....25
Table 7. Non-functional Requirement: High Performance and Responsiveness.....26
Table 8. Non-functional Requirement: Scalability of Application Infrastructure.....26
Table 9. Non-functional Requirement: Comprehensive Security Measures.....27
Table 10. Non-functional Requirement: User-Centric Design and Usability.....27
Table 11. Non-functional Requirement: System Reliability and High Availability.....28
Table 12. Non-functional Requirement: Maintainability and Extensibility of the Codebase...28

ABSTRACT

A methodical approach is necessary to address the problems found in the "IHateMoney" system in an efficient manner. Identifying the precise nature of the issue is the first step in determining if it has to do with functionality issues, security flaws, or performance deterioration. This include auditing code and security, examining system performance data, and evaluating application logs. For prompt mitigation, rapid steps like reverting recent upgrades or boosting server resources will be taken. In order to better handle load and complicated procedures, we may need to improve security measures, optimize back-end processes, and potentially rethink the system architecture in order to find a long-term solution. Before being fully implemented, these solutions will undergo extensive testing. Following deployment, key performance indicators and user input will be used to regularly assess the efficiency of these solutions, ensuring that the system not only resolves the present issue but is also strengthened against future ones. This all-encompassing strategy guarantees both user delight and ongoing system integrity.

CHAPTER 1

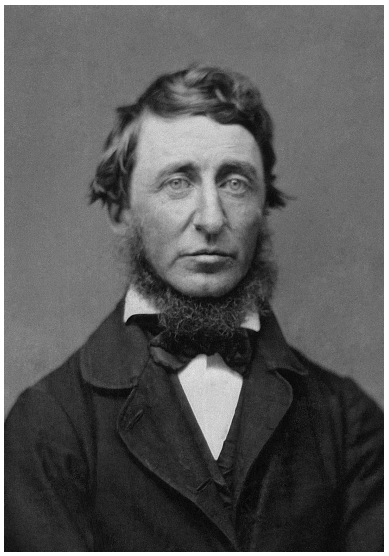
INTRODUCTION

1.1. Motivation

We believe that the relentless pursuit of money in a capitalist society often leads to stress, anxiety, and an unending cycle of consumption that leaves many feeling unfulfilled. We despise money's power over people's lives, dictating their choices and often leading to debt, financial insecurity, and an unhealthy work-life balance. Wealth inequality, economic anxiety, consumerism, lack of financial literacy, job insecurity, and rising living costs are pervasive money-related problems people face. Our mission is to help individuals take control of their finances by providing the tools and resources they need to manage their money effectively, empowering them to track spending, create budgets, and achieve financial peace of mind.

1.2. Problem Statement

In today's capitalist society, the relentless pursuit of money often leads to significant stress, anxiety, and a sense of unfulfillment. Many people are trapped in a consumption cycle, burdened by debt, and facing financial insecurity. Wealth inequality continues to widen, exacerbating the challenges for those struggling to achieve economic stability. The pervasive culture of consumerism drives individuals to spend beyond their means, while a lack of financial literacy leaves many ill-equipped to manage their finances effectively. Additionally, job insecurity and rising living costs add to the economic pressures individuals and families face. As Henry David Thoreau aptly stated, "Wealth is the ability to fully experience life," yet for many, financial strain prevents them from truly living.



David Henry Thoreau (1817-1862)

American naturalist, essayist, poet, and philosopher. A leading transcendentalist, he is best known for his book Walden, a reflection upon simple living in natural surroundings.

Mahatma Gandhi (1869-1948)

Indian lawyer, anti-colonial nationalist, and political ethicist who employed nonviolent resistance to lead the successful campaign for India's independence from British rule.

Figure 1. *Some inspirations*

Our web application aims to address these issues by providing a comprehensive solution that empowers individuals to take control of their finances. Our app offers tools for tracking spending, creating budgets, and making informed financial decisions, helping users overcome financial anxiety and achieve greater economic security. We believe as Mahatma Gandhi said, "The world has enough for everyone's need, but not enough for everyone's greed." By promoting financial literacy and responsible money management, we strive to help individuals find balance and peace in their financial lives, enabling them to focus on what truly matters.

1.3. Scope and Objectives

1.3.1. Scope

This report delineates the comprehensive scope and objectives of our spending management web application, designed to empower users to track, analyze, and optimize their financial expenditures efficiently. The application addresses the growing need for individuals and businesses to gain better control over their finances through intuitive digital tools. By leveraging modern web technologies, we aim to provide a seamless and user-friendly experience that enhances financial transparency and decision-making.

Figure 2. *Popular finance management apps*

Our approach to developing a spending management web application draws inspiration from popular money-tracking apps' success and user-focused features such as Mint, YNAB, Personal Capital, PocketGuard, Acorns, and Wally. These applications have set a high standard in the industry by offering intuitive interfaces, comprehensive expense-tracking functionalities, proactive budgeting tools, and insightful financial analyses. By studying their methodologies and listening to user feedback, we aim to incorporate the best practices and innovative features resonating with users, ensuring our application meets and exceeds expectations in helping individuals and businesses achieve greater financial control and transparency.

1.3.2. Objectives

At the core of our spending management web application lies the objective of delivering a user-centric interface that simplifies expense tracking. Our foremost goal is to ensure users can effortlessly input, categorize, and monitor their expenditures in real time. This includes comprehensive features for categorizing expenses across domains such as daily essentials, utilities, leisure, and more, enabling a holistic view of financial habits. Moreover, the application will empower users with tools for effectively setting and managing budgets.

These features will include personalized budget planning capabilities with alerts and notifications to inform users of budget thresholds and potential overspending, promoting proactive financial management.

A pivotal aspect of our application is its robust data visualization capabilities. Users will gain insights into spending patterns and trends through dynamic charts, graphs, and customizable reports. This visualization not only aids in understanding where money is being allocated but also facilitates strategic decision-making for future financial planning. Security is paramount; thus, stringent measures will be implemented to safeguard user data. This includes encryption protocols, secure authentication processes, and adherence to global data protection regulations to instill confidence in our users regarding the privacy and confidentiality of their financial information. Additionally, the application will support seamless integration with external financial accounts and services, enabling automatic transaction imports and reconciliation for enhanced accuracy and convenience. Ultimately, our spending management web application aims to be a trusted ally in fostering financial literacy, control, and empowerment among its users, contributing to their overall economic well-being.

1.4. Assumptions and Solutions

1.4.1. Assumptions

We develop our spending management web application under several vital assumptions based on industry trends, user behaviors, and technological advancements. Firstly, we assume a growing demand for digital tools that simplify and enhance personal and business financial management. This assumption is supported by the increasing adoption of mobile banking, budgeting apps, and digital wallets, indicating a shift towards convenient, real-time financial monitoring and planning.

Secondly, we assume users prioritize user-friendly interfaces and seamless experiences when choosing financial management tools. Drawing from successful apps in the market, such as Mint and YNAB, we anticipate that intuitive design, ease of navigation, and precise data visualization will be critical factors in user adoption and retention.

Thirdly, we assume robust security measures are needed to protect sensitive financial data. With heightened concerns about data privacy and cybersecurity threats, we prioritize

implementing encryption, secure authentication protocols, and compliance with data protection regulations to build trust and confidence among our users.

Lastly, we assume that integration capabilities with external financial accounts and services will enhance the utility of our application. Seamless connectivity with banks, credit cards, and investment platforms, similar to what Personal Capital and Acorns offer, will provide users with a consolidated view of their finances and streamline transaction tracking and reconciliation processes.

These assumptions guide our development strategy. We aim to create a comprehensive and reliable spending management tool that effectively meets modern users' evolving needs and expectations in managing their finances.

1.4.2. Solutions

In addressing the assumptions outlined for our spending management web application, we propose a solution that integrates innovative features and best practices from successful money-tracking apps while catering to our target users' specific needs and preferences.

Firstly, our solution will prioritize user experience with a clean, intuitive interface that facilitates effortless expense input, categorization, and analysis. Drawing inspiration from apps like Mint and YNAB, we will implement customizable budgeting tools and real-time expense tracking features to empower users to understand and control their finances effectively.

Secondly, our solution will employ state-of-the-art encryption methods, secure authentication mechanisms, and strict adherence to data protection regulations to address the security assumption. This approach ensures that user data remains confidential and protected from unauthorized access or breaches, fostering trust and reliability among our user base.

Thirdly, our solution will emphasize robust integration capabilities, enabling seamless connectivity with users' bank accounts, credit cards, and other financial services. Similar to Personal Capital and Acorns, this integration will facilitate automatic transaction imports, accurate financial reporting, and holistic financial insights, simplifying the management of multiple financial accounts and enhancing overall user convenience.

Additionally, our solution will incorporate dynamic data visualization tools such as interactive charts, graphs, and personalized financial reports. These features, inspired by apps

like PocketGuard and Wally, will provide users with clear insights into their spending patterns, savings goals progress, and overall financial health, empowering informed decision-making and proactive financial management.

By aligning our solution with these critical aspects derived from successful money-tracking apps, we aim to deliver a comprehensive spending management tool that meets and exceeds user expectations, effectively driving user engagement and satisfaction in managing their financial well-being.

1.5. Structure of report

The report is divided into the following sections: Abstract, Introduction, Related Work and Comparison, Methodology, Implementation and Results, Discussion and Evaluation, Conclusion and Future Work, and References.

- Introduction: This section defines the problem, describes the background that motivated the study, and explains the objectives and parameters of the inquiry.
- Related Work and Comparison: This study compares and contrasts the technologies and approaches of currently in-use apps with those of this work.
- Methodology: This section explains the tactics for design and development, along with the instruments and reasoning used. Additionally, it uses diagrams to clarify the system's design and requirements.
- Implementation and Results: This section describes the structure and configuration of the built system using several use cases. Videos or pictures illustrate the results.
- Discussion and Evaluation: Analyze the system's effectiveness and discuss the implementation's successes and failures based on the results.
- Conclusion and Future Work: The thesis concludes by reviewing the initial motivation, highlighting the key findings, and discussing the implications. It also outlines potential improvements and extensions to the project.
- References: A list of all the sources used in the study that are acknowledged for helping to shape the thesis.

CHAPTER 2

RELATED WORK & COMPARISON

2.1. Related Work

This section compares and contrasts You Need a Budget (YNAB), PocketGuard, and Mint, three well-known budgeting programs. Every tool has special features designed to increase savings, maximize spending, and promote financial literacy.

2.1.1. Mint

Figure 3. *Mint*

Mint, developed by Intuit, the company behind QuickBooks and TurboTax, is a free, web-based tool that integrates various aspects of personal finance management into one platform. It offers the following detailed features:

- Automatic Synchronization: Mint connects to virtually all US financial institutions to automatically download and categorize transactions. This includes banks, credit card companies, loan services, and investment accounts.
- Budget Creation and Alerts: Users can set custom budgets for different spending categories. Mint alerts users when they are approaching or exceeding these budgets.
- Financial Goals Setting: Mint allows users to set specific financial goals, such as paying off debt or saving for a down payment, and tracks progress towards these goals.
- Credit Score Monitoring: This service provides free access to your credit score, along with explanations of the factors affecting it and suggestions for improvement.
- Bill Payment Tracker: Mint also tracks your bills and subscriptions, sending reminders to avoid late payments and helping manage monthly expenses efficiently.

- Security: Mint uses multi-factor authentication and strong encryption protocols to secure user data, ensuring safety and privacy.

2.1.2. You Need a Budget (YNAB)

Figure 4. *YANB*

YNAB is not just a tool but a budgeting philosophy that encourages a proactive approach to managing money. It is based around four core rules which guide its functionality:

- Give Every Dollar a Job: YNAB encourages users to assign every dollar of income a specific purpose, whether for bills, savings, or discretionary spending, which helps prevent wasteful expenditures.
- Embrace Your True Expenses breaks down larger, less frequent expenses into manageable monthly allocations so users are prepared financially for them when they do occur.
- Roll With The Punches: YNAB is flexible, allowing users to adjust their budgets as needed throughout the month. This helps manage unexpected expenses without disrupting the financial plan.
- Age Your Money: Aims to teach users to live on income from the previous month, building a buffer of 30 days' expenses to increase financial stability.
- Educational Resources and Support: YNAB offers various educational materials and live workshops to improve financial literacy and budgeting skills.
- Subscription Fee: YNAB is not free; it charges an annual subscription fee, which includes full access to all tools and educational resources.

2.1.3. PocketGuard

Figure 5. *PocketGuard*

PocketGuard simplifies budget management by focusing on an easily understandable overview of your financial situation, making it particularly attractive for beginners or those overwhelmed by more complex systems.

- In My Pocket: Calculates how much money you have available after accounting for committed expenses, goals, and savings, simplifying spending decisions.
- Bill Tracking and Negotiation: Automatically identifies bills and subscriptions and provides insights into where users might save money, offering tools for renegotiating or canceling unnecessary subscriptions.
- Savings Goals: Integrates goal setting and tracking within the app, allowing users to see how saving money impacts their spending power.
- Bank-Level Security: This type of security uses industry-standard security features to protect user data, including encryption and biometric entry (such as fingerprint access).
- Free vs. Plus Versions: PocketGuard is available in both a free version, which offers basic functionalities, and a Plus version, which offers advanced features such as detailed budgeting options and the ability to export data.

2.2. Comparison

We'll list key features and criteria often considered important for budgeting tools. This table will help visualize the differences and similarities among these applications.

Feature/Criteria	Mint	YNAB	PocketGuard
Cost	Free	Subscription ($84/year)	Free & Plus versions
Budgeting Philosophy	Automated tracking	Proactive budgeting	Simplified budgeting
User Interface	Comprehensive dashboard	User-friendly, requires learning	Simple and intuitive
Automatic Sync	Yes	Yes	Yes
Bill Tracking	Yes	No	Yes
Financial Goals Tracking	Yes	Yes	Limited
Investment Tracking	Yes	No	No

Table 1. *Comparison between existing tools*

Many complimentary services, such as credit monitoring, budget formulation, and automated transaction classification, are available via Mint. However, its intricate and full dashboard may be intimidating to novice users and have a more challenging learning curve.

You Need a Budget (YNAB) offers a comprehensive library of instructional courses in exchange for an annual membership fee. YNAB encourages a proactive approach to budgeting. The idea behind YNAB is to give each dollar a task, which encourages financial discipline but also calls for a commitment to sticking to a budget that not everyone can afford.

With its user-friendly design, PocketGuard streamlines the personal financial management process and concentrates on giving consumers a clear view of their spare money once all obligations have been met. This method is very appealing to novices or those who would like a simple financial summary.

Through a smart integration of these current platforms' qualities and resolution of their shortcomings, "IHateMoney" can establish a distinct niche within the budgeting tool industry. This approach would provide a wide range of users with anything from simple budget management to sophisticated financial planning, all while maintaining a complete and user-friendly service.

2.3. Background

Creating efficient and user-friendly budgeting software has become essential to support users' improved financial management in today's financial technology environment. Utilizing a wide range of frameworks and technologies, "IHateMoney" aims to maximize user experience and functionality. An overview of the technological foundation for the application is given in this part, along with information on the functions and interactions of the many technologies used in its architecture.

2.3.1. Front-End Technologies

The foundation of web development, HTML, CSS, and JavaScript, are used in the "IHateMoney" front end. The basis for text, links, and data tables is laid down in the online content using HTML (HyperText Markup Language). Cascading Style Sheets, or CSS, improves the visual appearance by enabling complex theming and responsive designs that work well on various devices. Complex user interactions, dynamic information updates, and form validations are just a few of the interactive features made possible by JavaScript.

Figure 6. *Front-end Technologies*

Vaadin, a web application framework that smoothly combines with TypeScript, a superset of JavaScript, further improves the front end. TypeScript features strong typing,

object-oriented programming, and compiled error checking, enhancing the code's resilience and maintainability. Vaadin's usage of web components streamlines the development process. It lessens browser burden by enabling developers to create reusable user interface components and handle user state more effectively on the server side.

2.3.2. Back-End Technologies

Java and Spring Boot form the core of the back end. Java's platform independence, robustness, and object-oriented capabilities are ideal for building reliable and scalable server-side applications. Spring Boot, a framework derived from the Spring ecosystem, simplifies the configuration and deployment of Spring applications. It provides embedded server options like Tomcat and Jetty, eliminating the need for external server configurations and speeding up the deployment process.

Figure 7. *Back-end Technologies*

Spring Boot's extensive suite of libraries and quick setup for data access, security, and MVC architectures make it particularly valuable for rapid development cycles in enterprise settings. Its auto-configuration features intelligently load and configure application beans based on the context, reducing the need for manual setup.

2.3.3. Database Management

Figure 8. *MySQL Database Management*

MySQL is the relational database management system for "IHateMoney." Known for its reliability and robustness, MySQL supports complex queries, large datasets, and concurrent database access, which are critical for handling the financial data of "IHateMoney." MySQL's structured query language allows for precise and flexible interaction with the data, ensuring data integrity through foreign keys, transactions, and lock-based controls.

2.3.4. Authentication and Security

Figure 9. *Spring Security*

Security within "IHateMoney" is paramount, especially considering the sensitivity of financial data. Spring Security protects application access by robustly integrating authentication and authorization mechanisms. It supports various authentication protocols, including Spring Security and traditional form-based authentication. Spring Security's method-level security ensures that operations are performed only by authorized users, and its CSRF (Cross-Site Request Forgery) protection defends against common web vulnerabilities.

2.3.5. Data Visualization

Figure 10. *Visulaize Tools*

Data visualization in "IHateMoney" is managed through Highcharts and integrated with Vaadin. Highcharts provides various customizable chart options, such as pie charts, bar graphs, and line charts, which are instrumental in displaying financial data in an easily digestible format. The Vaadin integration allows these charts to be seamlessly embedded within the web application, providing the end-user with real-time updates and interactive capabilities.

2.3.6. Deployment

Figure 11. *Microsoft Azure*

Deployment on Microsoft Azure leverages Azure's cloud infrastructure to ensure that "IHateMoney" remains highly available, scalable, and secure. Azure provides tools for automating deployment, managing scalability through Azure Scale Sets, and securing applications with Azure Active Directory and network security groups. The platform's global infrastructure ensures low latency and high performance, which is critical for the user experience in financial applications.

CHAPTER 3

METHODOLOGY

This chapter has two parts. The first section provides the functional and non-functional requirements for building our system. The changes to the system's structure are then further explained, ensuring that every optimal solution is carried out in full.

3.1. Requirement analysis

Before we can begin any action, we must identify all of the system's needs. The functional and non-functional criteria for my new system are listed below.

3.1.1. Functional requirements

Name	User Authentication System
Summary	Implement a secure user authentication system that enables users to register, log in, and securely access their accounts.
Rationale	Ensures that only authenticated users can access their financial data, protecting privacy and enhancing security.
Requirements	• Utilize Spring Security for authentication mechanisms. • Passwords must be hashed and securely stored. • Provide functionality for password recovery and account verification via email.

Table 2. *Functional Requirement: User Authentication System*

Name	Budget Management Interface
Summary	A dynamic user interface allows users to create, modify, and track budgets with various financial categories.
Rationale	Facilitates effective financial planning and monitoring, enabling users to maintain control over their financial health.
Requirements	• Allow users to set budget limits for different categories. • Implement real-time updates to reflect changes in the budget. • Provide visualizations of budget allocations using Vaadin/Highcharts.

Table 3. *Functional Requirement: Budget Management Interface*

Name	Transaction Recording and Categorization
Summary	System for users to log and categorize financial transactions linked to specific budgets and categories.
Rationale	Critical for accurate financial tracking and budget adherence, providing insights into spending habits.
Requirements	• Support manual entry and import of transaction data. • Transactions should include detailed metadata such as amount, date, description, and category. • Automatic categorization of transactions based on predefined rules or past behaviour.

Table 4. *Functional Requirement: Transaction Recording and Categorization*

Name	Advanced Financial Reporting and Analytics
Summary	Tools to generate comprehensive reports and analytics on user finances, including spending patterns and budget compliance.
Rationale	Provides valuable insights into financial behavior, aiding users in making informed decisions.
Requirements	• Generate monthly, quarterly, and annual financial reports. • Offer interactive charts and graphs to visualize data trends. • Allow users to customize reports based on specific parameters and dates.

Table 5. *Functional Requirement: Advanced Financial Reporting and Analytics*

Name	Automated Data Backup and Recovery System
Summary	The system will regularly back up user data and provide a reliable restoration method in case of loss.
Rationale	Ensures data integrity and reliability, safeguarding user information against accidental loss or corruption.
Requirements	• Scheduled daily backups of all user data. • Secure storage of backups in a separate physical location or cloud storage. • Quick and reliable recovery processes to restore user data from backups when necessary.

Table 6. *Functional Requirement: Automated Data Backup and Recovery System*

3.1.2. Non-functional requirements

Name	High Performance and Responsiveness
Summary	Ensure that the application maintains high performance and responsiveness under varying load conditions.
Rationale	To provide a smooth and efficient user experience even during peak usage times, preventing slowdowns or downtime
Requirements	• The application should handle multiple simultaneous user requests without degradation in performance. • Response times for user interactions should not exceed 2 seconds. • Optimize backend processes and database queries for maximum efficiency.

Table 7. *Non-functional Requirement: High Performance and Responsiveness*

Name	Scalability of Application Infrastructure
Summary	The system architecture must support easy scaling to increase user and data growth.
Rationale	To ensure the application can grow without significant changes to the infrastructure, supporting more users and data over time.
Requirements	• Implement microservices architecture or modular components where feasible to facilitate scaling. • Utilize cloud services that allow for dynamic resource allocation. • Perform regular load testing to guide scaling decisions.

Table 8. *Non-functional Requirement: Scalability of Application Infrastructure*

Name	Comprehensive Security Measures
Summary	Implement comprehensive security measures to protect user data and prevent unauthorized access.
Rationale	To safeguard sensitive financial information and comply with data protection regulations, ensuring user trust.
Requirements	• All data transmissions must be encrypted using SSL/TLS. • Implement OWASP security best practices in the development lifecycle. • Conduct periodic security audits and vulnerability assessments.

Table 9. *Non-functional Requirement: Comprehensive Security Measures*

Name	User-Centric Design and Usability
Summary	The application should feature an intuitive, easy-to-navigate user interface suitable for users with varying levels of technical proficiency.
Rationale	Enhances user satisfaction and engagement, reducing the learning curve for new users.
Requirements	• Conduct usability testing phases to refine interface design. • The interface should be accessible on various devices and screen sizes, ensuring mobile responsiveness. • Provide tooltips, help documentation, and onboarding tutorials for users.

Table 10. *Non-functional Requirement: User-Centric Design and Usability*

Name	System Reliability and High Availability
Summary	Ensure the application is reliable and always available with minimal downtime.
Rationale	Critical for maintaining user trust and satisfaction, particularly for a financial application where access needs are often time-sensitive.
Requirements	• Implement failover mechanisms and redundancy in critical components of the infrastructure. • The system should have an uptime of 99.9%. • Employ monitoring tools to detect and resolve issues proactively.

Table 11. *Non-functional Requirement: System Reliability and High Availability*

Name	Maintainability and Extensibility of the Codebase
Summary	The codebase should be well-documented, clean, and organized to facilitate easy updates, maintenance, and feature extensions.
Rationale	Ensures that the application can evolve with minimal technical debt and that new developers can onboard quickly.
Requirements	• Follow industry-standard coding practices and guidelines. • Ensure that all code is reviewed for clarity and maintainability before integration. • Documentation must be up-to-date and include system architecture, codebase, and API references.

Table 12. *Non-functional Requirement: Maintainability and Extensibility of the Codebase*

3.1.3. Use Case diagram

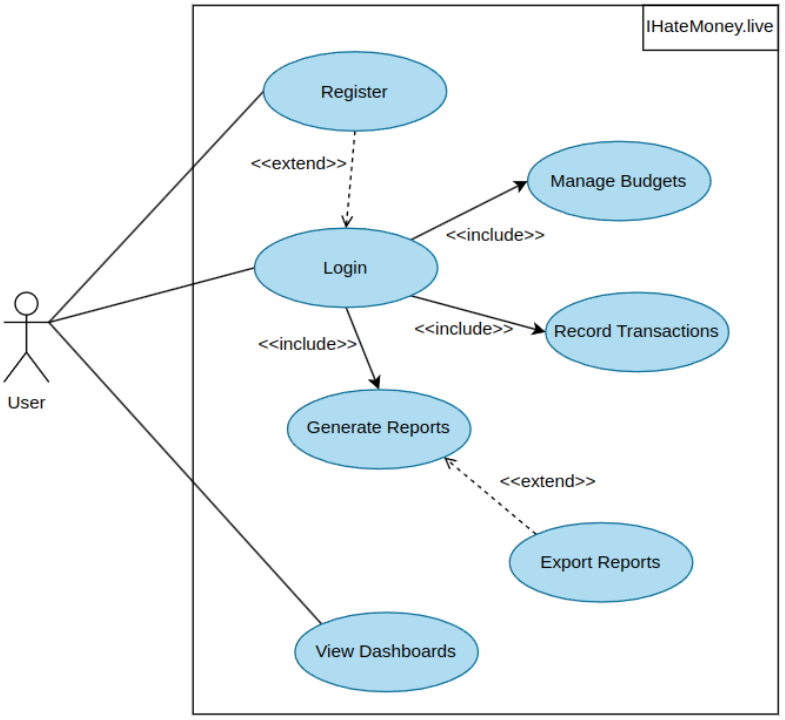


Figure 12. *User Use Case Diagram*

The use case diagram illustrates user interactions with the "IHateMoney.live" application. Each use case represents a different function the application provides. Here's a breakdown of the user interactions as depicted in the diagram:

- Register: This use case allows the user to create a new account in the system. It is an entry point for new users to access the application's functionalities. This use case is connected to "Login" via an "extend" relationship, indicating that registration can lead directly to logging in under certain conditions or provide additional options during registration.
- Login: The central use case where the user must authenticate themselves to access their personalized settings and data related to budgeting. Once logged in, the user can engage in several activities, which are required ("include" relationship) for other actions:
    - Manage Budgets: Users can create, modify, view, and delete budgets. This function is essential for setting up financial planning parameters.
    - Record Transactions: This allows users to enter transactions against their budgets, which is crucial for tracking spending and maintaining financial records.

    - Generate Reports: Users can generate various financial reports based on their budgets and transactions, providing insights into their financial health.
    - View Dashboards: This use case provides a visual overview or dashboard of the user's financial data, likely displaying key metrics, charts, and summaries.
- Generate Reports: In addition to providing direct financial reports, this use case extends to "Export Reports," indicating that users can export the generated reports into various formats for external use, such as printing or analysis in other tools. The "extend" relationship suggests that exporting reports is optional and contingent upon user needs.
- Export Reports: As an extension of "Generate Reports," this represents a specialized function where the user can take the reports created by the application and export them to different formats (e.g., PDF, Excel).

The diagram uses solid lines with arrows to denote "include" relationships, signifying that the base use case (Log in) action includes the connected use cases as part of its operation. The dashed lines with arrows depict "extend" relationships, indicating optional or conditional functionalities that extend the base use case capabilities.

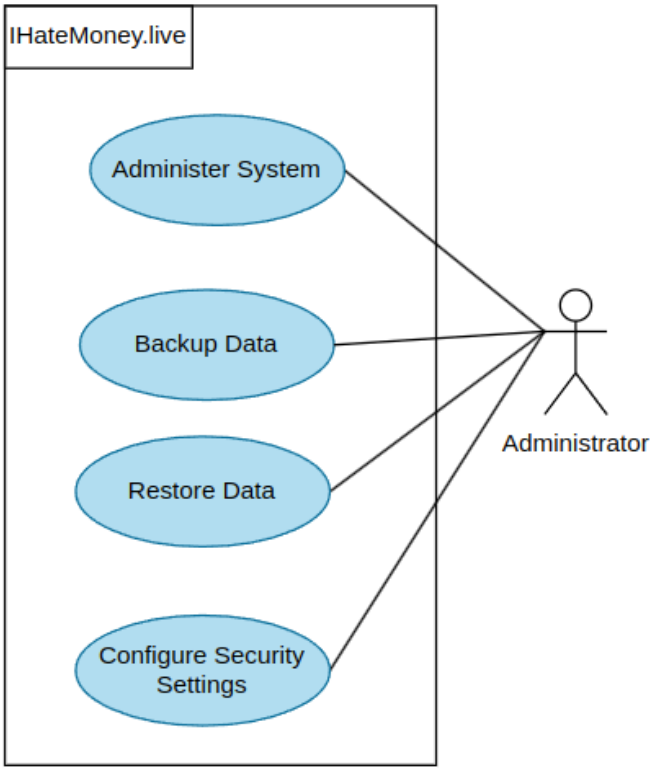


Figure 13. *Admin Use Case Diagram*

The use case diagram submitted for the "IHateMoney.live" application describes the features the Administrator position may access. Below is a description of each use case that the graphic shows:

- Manage System: This generic use case shows that the Administrator can control and direct the operation of the application system as a whole. This might include duties such as monitoring system performance, controlling user accounts, upgrading or changing software components, and performing routine maintenance to guarantee the program functions properly. This use case is a general category with many system management tasks.
- Data Backup: This use case allows the administrator to carry out data backups. Data backups are essential to the program because they guarantee restoring all user and financial data if lost due to system failures, corruption, or other unanticipated problems. Because backups are so important to data management and security, administrators are usually the only ones with the authority to start and maintain them.
- Recover Data: This feature, complementary to the Backup Data use case, allows the Administrator to recover data from earlier backups. In recovery cases, this is essential when the application's data integrity is jeopardized. The ability to recover data offers a safety net for the application's operational continuity by ensuring less downtime and data loss.
- Configure Security Settings: This use case entails adjusting and customizing the application's security parameters and protocols. To safeguard the program against unwanted access and possible security risks, this may include controlling encryption settings, setting up firewalls, establishing user authentication parameters, and implementing other security procedures.

These use cases are directly related to the Administrator, suggesting that only people with administrative capabilities can perform these duties. As the diagram makes clear, the "IHateMoney.live" application's operational integrity, security, and dependability are all critically dependent on administrators.

Overall, the use case diagrams highlight the tasks and responsibilities assigned to the Administrator and User while concisely capturing the functionality necessary for the application's reliable administration and user.

3.2. System Design

The architecture, deriving from the AbstractEntity class, prioritizes modularity and reuse while guaranteeing the centralization of common functionality such as timestamping and ID management. This structure supports the DRY (Don't Repeat Yourself) principle, which is essential for maintaining clear, effective, and error-free code.

3.2.1. Activity Diagram

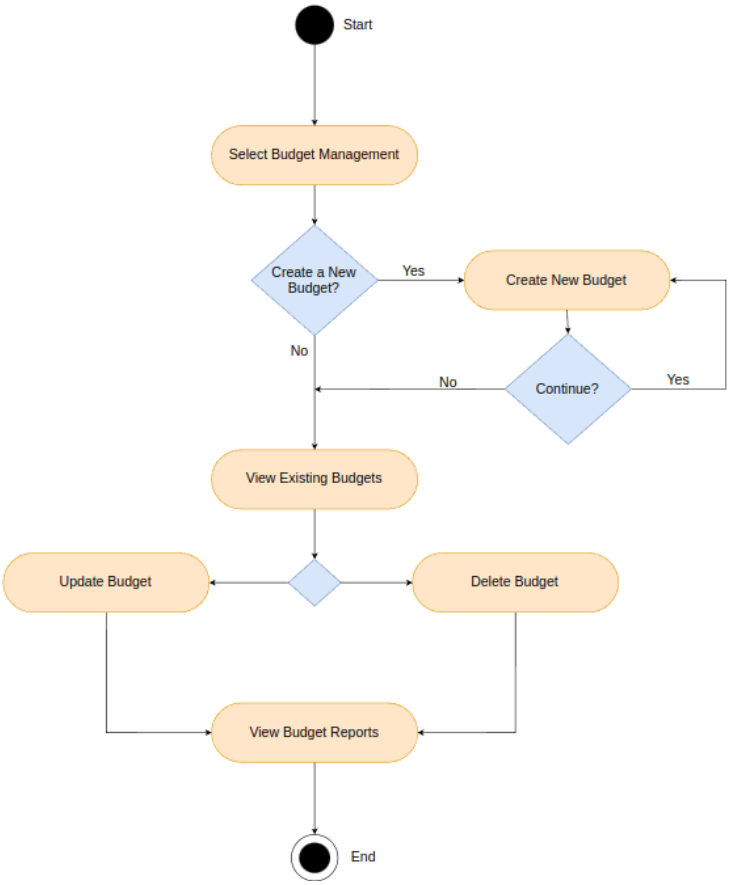


Figure 14. *Activity Diagram*

The diagram starts at the initiation of the budget management process and details the steps a user might take to manage budgets, including creating, viewing, updating, and deleting budgets and viewing reports related to those budgets.

1. Start: The process begins when a user selects the "Budget Management" option in the application.
2. Decision Point: Create a New Budget.
    - If "Yes," the user proceeds to "Create New Budget." This activity likely involves entering details such as budget name, amount, start and end dates, and expenditure categories.
    - If "No," the process moves to "View Existing Budgets."

3. Create New Budget:
    - After creating a new budget, there's a decision point labelled "Continue?" which implies that the user can proceed with further actions or finish the budget creation process.
    - If "Yes," the process loops back, allowing the user to either create another new budget or view existing ones.
    - If "No," it progresses to "View Existing Budgets."
4. View Existing Budgets:
    - This step is a central node that connects to multiple activities. Users can select a budget to view and then update or delete it.
    - The flow from here is conditional based on user input, which directs users to either "Update Budget" or "Delete Budget."
5. Update Budget:
    - In this step, the user modifies the details of an existing budget. Modifications could include changing the budget amount, duration, or associated categories.
    - Following updates, the process leads to "View Budget Reports," suggesting that updates might often be reviewed in the context of budget performance reports.
6. Delete Budget:
    - This allows the user to remove a budget from the system entirely.
    - Post-deletion, the workflow also directs to "View Budget Reports," perhaps to confirm that the budget has been successfully deleted or to review the impacts on overall financial planning.
7. View Budget Reports:
    - This activity seems to be a common final step in various workflow branches, highlighting its importance. Viewing reports likely provides comprehensive insights into budget performance, spending trends, and financial forecasting.
8. End: The process concludes, marking the end of the user’s session in budget management.

With decision points representing potential user alternatives, this activity diagram successfully depicts a user-centric process in a budget management system. Because of the design's emphasis on adaptability, users can easily move between various budget management activities in a single session. This configuration improves the user experience by offering a thorough and rational method for managing personal finances.

3.2.2. Class Diagram

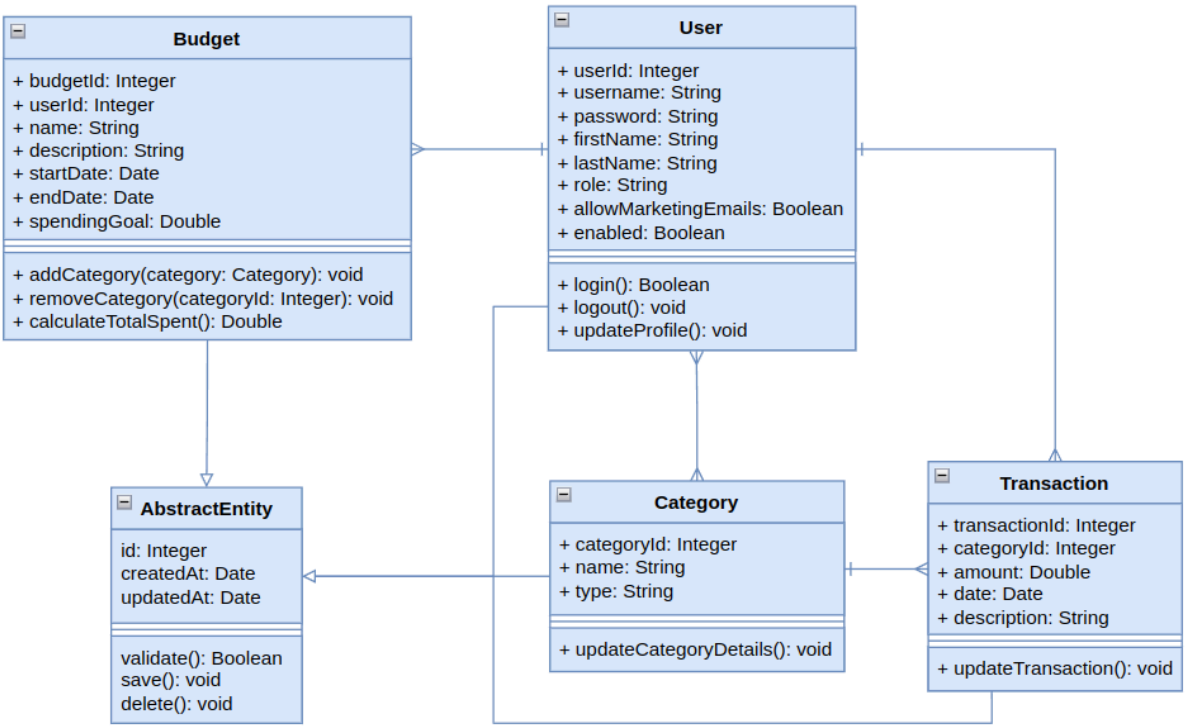


Figure 15. *Class diagram*

The class diagram illustrates the object-oriented structure for the "IHateMoney" application, detailing how entities like User, Budget, Category, and Transaction are designed and interact within the system. Each of these classes inherits from an AbstractEntity class, which standardizes common attributes and methods across all entities. Here's a detailed analysis of each class represented in the diagram:

1. AbstractEntity: This is the base class from which all other entities inherit, ensuring that common properties and methods are shared across the system.

   Attributes:

   - id: A unique identifier for each instance of an entity.
   - createdAt: The date and time when the entity was created.
   - updatedAt: The date and time when the entity was last updated.

   Methods:

   - validate(): Checks if the entity's current state is valid.
   - save(): Saves or updates the entity in the database.
   - delete(): Deletes the entity from the database.

2. User: Represents a system user and encapsulates all user-related information.

   Attributes:

   - Inherits id, createdAt, and updatedAt from AbstractEntity.
   - username: The user's login identifier.
   - password: Securely stored password for authentication.
   - firstName and lastName: The user's full name.
   - role: Defines the user's role within the system (e.g., admin, standard user).
   - allowMarketingEmails: Indicates whether the user has opted in to receive marketing emails.
   - enabled: Boolean flag indicating whether the user's account is active.

   Methods:

   - login(): Authenticates the user against the database.
   - logout(): Ends the user session.
   - updateProfile(): Allows the user to update their profile information.

3. Budget: Handles the financial planning aspects by tracking budgets.

   Attributes:

   - Inherits base attributes from AbstractEntity.
   - userId: References the User class to associate the budget with a specific user.
   - name: The name or title of the budget.
   - description: A brief description of what the budget is for.
   - startDate and endDate: The time frame for the budget.
   - spendingGoal: The financial target set for the budget period.

   Methods:

   - addCategory(Category): Adds a new category to the budget.
   - removeCategory(categoryId): Removes a category from the budget.
   - calculateTotalSpent(): Calculates the total amount spent under this budget.

4. Category: Organizes transactions into manageable groups.

   Attributes:

   - Inherits base attributes from AbstractEntity.
   - name: The name of the category.
   - type: A descriptor of the category type, helping to classify transactions further.

Methods:

- updateCategoryDetails(): Updates the attributes of a category.

5. Transaction: Tracks individual financial transactions.

   Attributes:

   - Inherits base attributes from AbstractEntity.
   - categoryId: Links the transaction to a specific category.
   - amount: The monetary value of the transaction.
   - date: The date the transaction occurred.
   - description: Details about the transaction.

   Methods:

   - updateTransaction(): Allows updating the details of the transaction.

This class diagram efficiently organizes the basic entities needed to handle the application's financial information. By using inheritance from AbstractEntity, the system ensures that all entities implement necessary methods and attributes in the same way, improving maintainability and eliminating repetition.

3.2.3. Entity Relationship Diagram

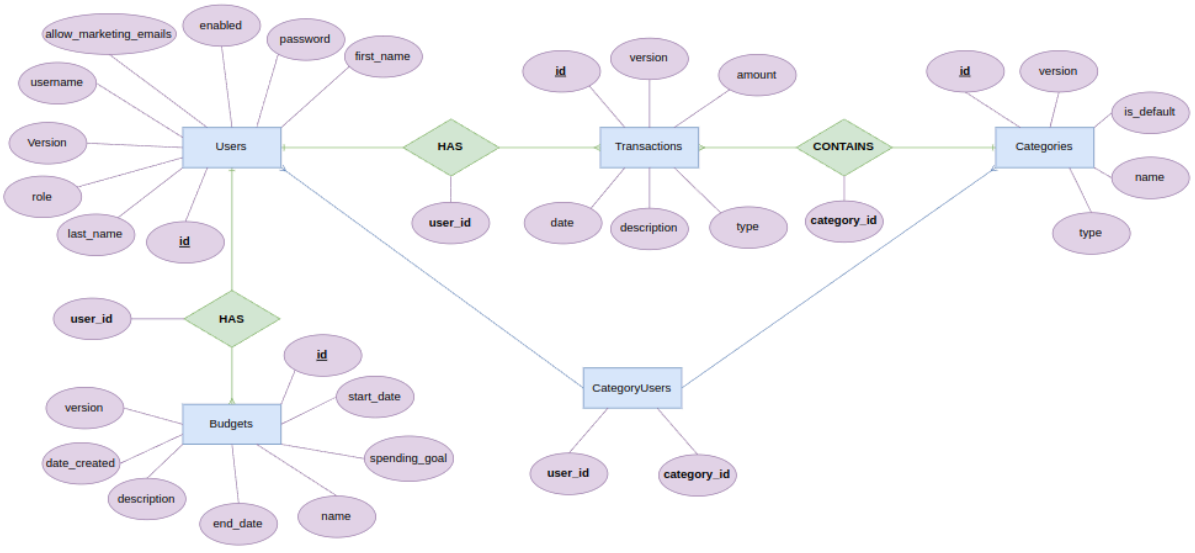


Figure 16. *ER Diagram*

The Entity-Relationship Diagram (ERD) represents the database schema for a financial management system, detailing how different entities, such as Users, Budgets, Transactions, and Categories, are related.

Users' attributes:

- id: A unique identifier for each user.
- username: The user's login name.
- password: The user's password for authentication.
- first_name and last_name: The user's names.
- role: Defines the user's role within the system (e.g., admin, standard user).
- allow_marketing_emails: Boolean indicating whether the user has opted to receive marketing emails.
- enabled: Boolean indicates whether the user's account is active.
- version: A system-generated version number for handling data consistency during updates.

Budgets' attributes:

- id: A unique identifier for each budget.
- user_id: Foreign key linking the budget to a specific user.
- name: The name or title of the budget.
- description: A brief description of the budget.
- start_date and end_date: Define the duration of the budget.
- spending_goal: The financial target for the budget period.
- date_created: The date when the budget was created.
- version: Similar to Users for data consistency.

Transactions' attributes:

- id: A unique identifier for each transaction.
- user_id: Foreign key linking the transaction to a user (optional, not always present depending on the system design).
- category_id: Foreign key linking the transaction to a specific category.
- amount: The monetary value of the transaction.
- date: The date of the transaction.
- description: Details about the transaction.
- type: The nature or type of the transaction (e.g., debit, credit).
- version: For data versioning.

Categories' attributes:

- id: A unique identifier for each category.
- name: The name of the category.
- type: A descriptor of the category type, helping to classify transactions further.

- is_default: Boolean indicating whether the category is a default system category.
- version: Used for maintaining data integrity.

CategoryUsers: This join table was created to manage the many-to-many relationship between Categories and Users.

- user_id: Foreign key that references Users.
- category_id: Foreign key that references Categories.

Relationships:

- Users HAS Budgets: This is a one-to-many relationship. Each user can have multiple budgets, but each is associated with only one user.
- Users HAS Transactions: Optionally, a one-to-many relationship suggests that not all transactions need to be directly linked to a user.
- Transactions CONTAINS Categories: Many-to-many relationships via CategoryUsers, indicating that transactions can involve multiple categories and vice versa.
- CategoryUsers: This join table supports the many-to-many relationship, associating categories with multiple users and vice versa.

The ERD efficiently organizes the relational structure required to enable a sophisticated financial management system. It includes normalization to minimize redundancy (via foreign keys and ID fields) and features to enable comprehensive record management (such as a version for optimistic concurrency control). This schema supports robust application design that can handle various financial duties, from basic budget management to intricate transaction monitoring across several categories and users.

CHAPTER 4

IMPLEMENT AND RESULTS

This section describes the implementation and result of the "IHateMoney" system, which is pivotal for managing financial data across various entities such as Users, Budgets, Transactions, and Categories. The design utilizes a relational database model to ensure data integrity, facilitate complex queries, and support efficient data management practices.

4.1. Implement

4.1.1. Implement View

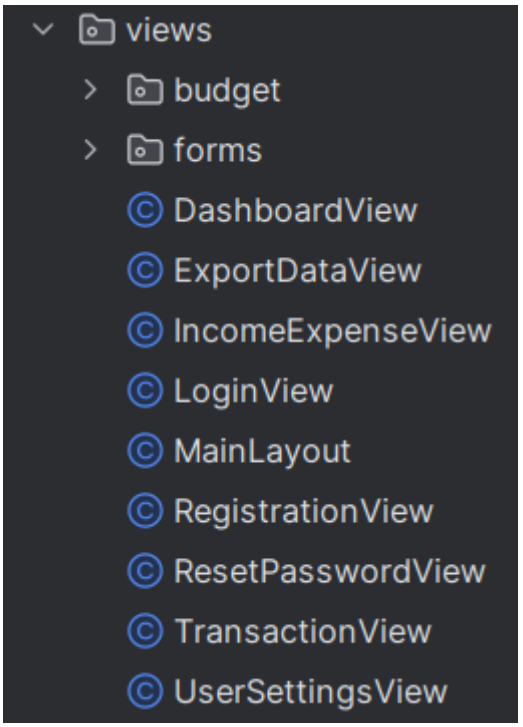


Figure 17. *View Directory*

The views directory is organized into several Java classes, each representing a different view or screen within the "IHateMoney" application. This structure is essential for a Vaadin-based application as it facilitates the modular design of user interfaces, making it easier to manage and update individual components of the application. Here's a breakdown of each view:

1. DashboardView

Purpose: This view likely serves as the main dashboard for users, providing a summary or an overview of their financial status, recent transactions, budgets, or other key metrics.

2. ExportDataView

Purpose: Handles the functionality to export user data, possibly including transaction histories, budget reports, or other financial data. This could support various formats like CSV, PDF, or Excel.

3. IncomeExpenseView

Purpose: Dedicated to managing and displaying information specifically related to income and expenses. This view helps users track their financial inflows and outflows, potentially offering tools for categorization and analysis.

4. LoginView

Purpose: Manages user authentication. This is the interface where users log into their accounts, incorporating fields for username and password and possibly options for password recovery or two-factor authentication.

5. MainLayout

Purpose: Acts as the main template or layout for the application, defining the common design elements and navigation components that are consistent across different views.

6. RegistrationView

Purpose: Provides a user interface for account creation, where new users can sign up by entering their details, such as name, email, and password.

7. ResetPasswordView

Purpose: Allows users to reset their passwords, likely including security questions or email verification steps to authenticate user identity before permitting password changes.

8. TransactionView

Purpose: Offers a detailed interface for managing transactions. This could include features for adding, editing or deleting transactions, along with filters or search capabilities to navigate through transaction records.

9. UserSettingsView

Purpose: Enables users to customize settings or update their profile information. This might include preferences for notifications, account details, or security settings.

4.1.2. Implement Data

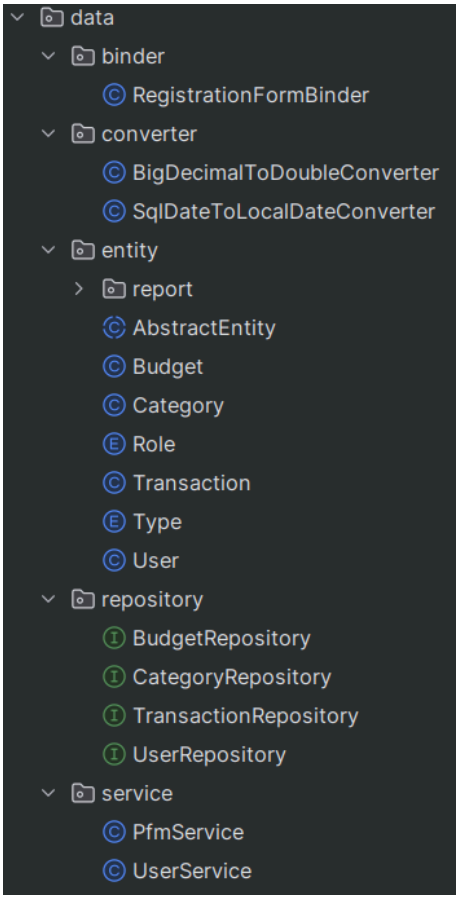


Figure 18. *Data Directory*

The data package is crucial for handling all data-related functionalities within the application, including database entity definitions, data access layers, and utility classes for data conversion and form binding.

1. **binder:**

- RegistrationFormBinder: This class manages the binding of form data to and from Java objects related to user registration. Binding classes simplify handling form submissions by automatically populating object fields from form fields.

2. **converter**

- BigDecimalToDoubleConverter: Converts BigDecimal values to Double, probably used for calculations or data transformations where precise decimal data from the database needs to be converted to a more commonly used data type in Java.

- SqlDateToLocalDateConverter: Converts SQL Date objects to Java 8 LocalDate objects. This is useful for applications that utilize modern date and time APIs in places where legacy SQL data types are retrieved from databases.

3. **entity:** Entities represent tables in the database. Each entity class corresponds to a table schema definition.

- AbstractEntity A base class providing common attributes like id, createdAt, and updatedAt and methods that other entity classes inherit.
- Budget, Category, Role, Transaction, Type, User: These classes define the schema for their respective tables. Attributes in these classes correspond to columns in the database. For example, the User might include a username, password, role, etc.

4. **report:** This might be a specialized sub-package or class dealing with generating or managing reports derived from the entity data.
5. **repository:** Repositories abstract the data layer, providing a collection-like interface for accessing domain objects.

- BudgetRepository, CategoryRepository, TransactionRepository, UserRepository: These interfaces extend Spring Data JPA's JpaRepository, allowing for CRUD operations and custom queries to be defined for handling the data of budgets, categories, transactions, and users, respectively.

6. **service:** Services contain business logic, calling on repositories to interact with the database.

- PfmService (Personal Financial Management Service): Handles complex operations and transactions related to personal financial management.
- UserService: Manages user-related operations such as authentication, profile management, and possibly role-based access controls.

4.1.3. Implement Security

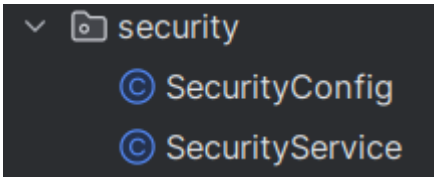


Figure 19. *Security directory*

1. SecurityConfig:

Purpose: This class is typically used to configure various aspects of security in the application. It extends one of the Spring Security configuration adapters, often WebSecurityConfigurerAdapter, which provides methods to configure aspects like which URL paths are secured, which are publicly accessible, configure custom login pages, and set up form-based or HTTP Basic authentication.

Functionalities:

- Authentication Manager Configuration: It configures authentication mechanisms, specifying how the users are loaded and the password encoding.
- Authorization Rules: Defines which roles or authorities are required to access specific resources. For example, you might restrict certain APIs to users with an ADMIN role.
- HTTP Security Configuration: Set up rules for CSRF protection, session management, and rules for which requests need to be authenticated and which do not.
- Custom Security Expressions: Additional complex security rules can be implemented as needed.

```
@Override  Nhathuy1305
protected void configure(HttpSecurity http) throws Exception {
    super.configure(http);

    setLoginView(http, LoginView.class);    // Set the custom login v
    http.csrf().disable();
    http
            .formLogin() FormLoginConfigurer<HttpSecurity>
            .loginPage("/login")
            .loginProcessingUrl(LOGIN_PROCESSING_URL)
            .failureUrl(LOGIN_FAILURE_URL)
            .defaultSuccessUrl( defaultSuccessUrl: "/", alwaysUse: true)
            .permitAll()
            .and() HttpSecurity
            .logout() LogoutConfigurer<HttpSecurity>
            .logoutSuccessUrl("/login")
            .permitAll()
            .and() HttpSecurity
            .exceptionHandling() ExceptionHandlingConfigurer<HttpSecurity>
            .accessDeniedPage( accessDeniedUrl: "/access-denied");

    http.headers().frameOptions().disable();
}
```

Figure 20. *Customize the security settings*

2. SecurityService

Purpose: This class likely acts as a service layer for security, providing a high-level abstraction over the security operations such as login, logout, and perhaps session management or even user details retrieval.

Functionalities:

- Load User Details: Might interface with UserDetailsService to load user details from the database. This is crucial for authenticating users and building the authentication object.
- Password Management: Could offer services like password encryption and comparison using password encoders provided by Spring Security.
- User Authentication: Handles the logic to authenticate a user against the stored user details and current security configurations.

4.2. Results

Starting with the Login Page:

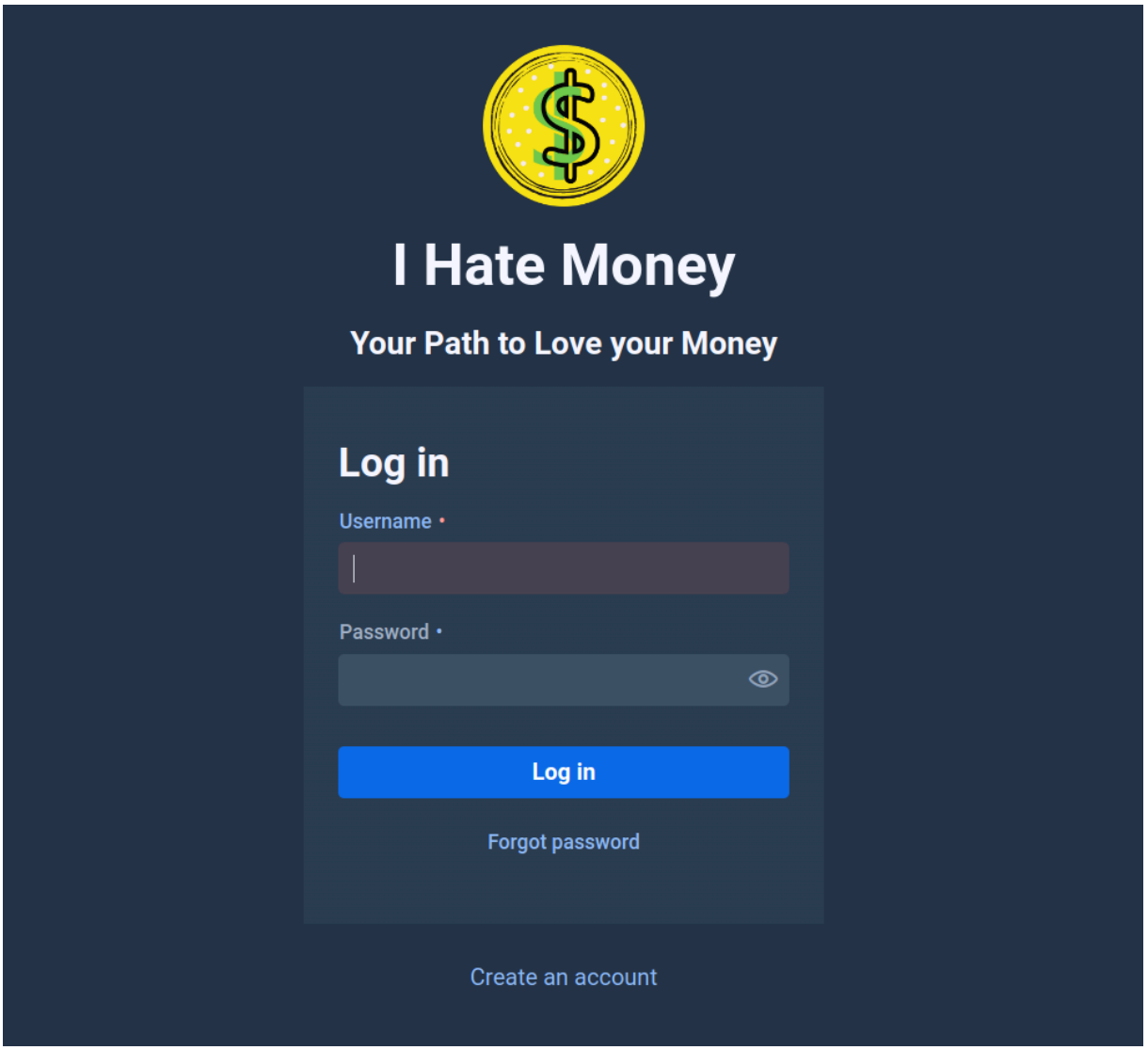


Figure 21. *Login Page*

If you encounter difficulties logging in, click the "Forgot Password" button. This action triggers an email to your registered account containing a reset link. Please note that this link will expire after 5 minutes for security purposes.

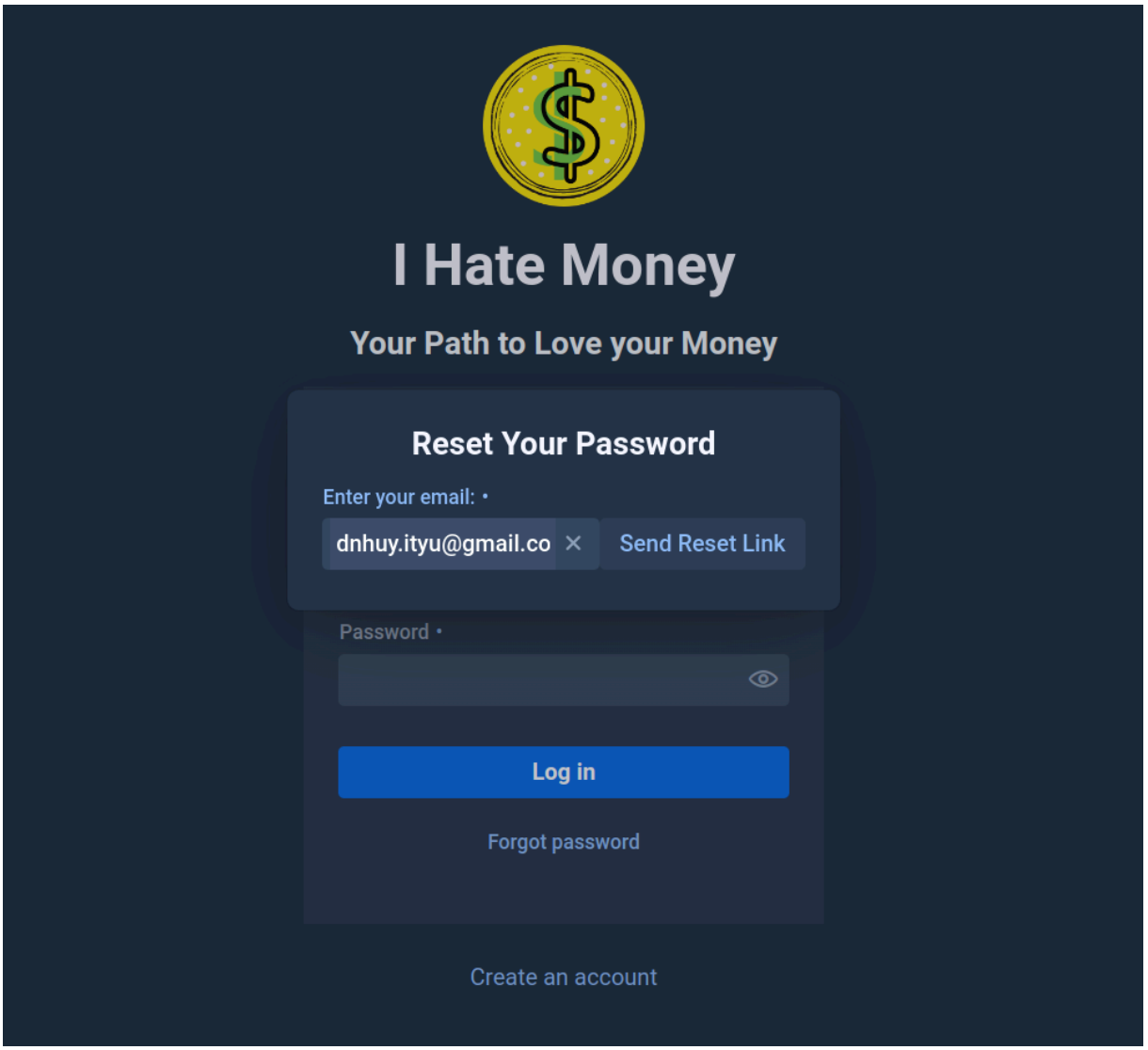


Figure 22. *Forgot Password page*

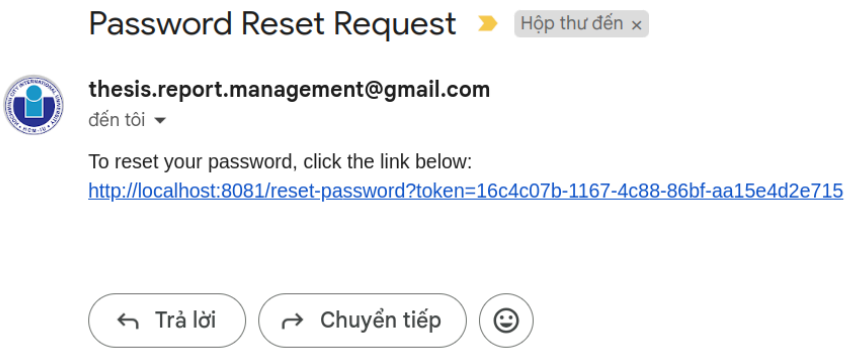


Figure 23. *Sample Email Received*

Here is what you received after click the link:

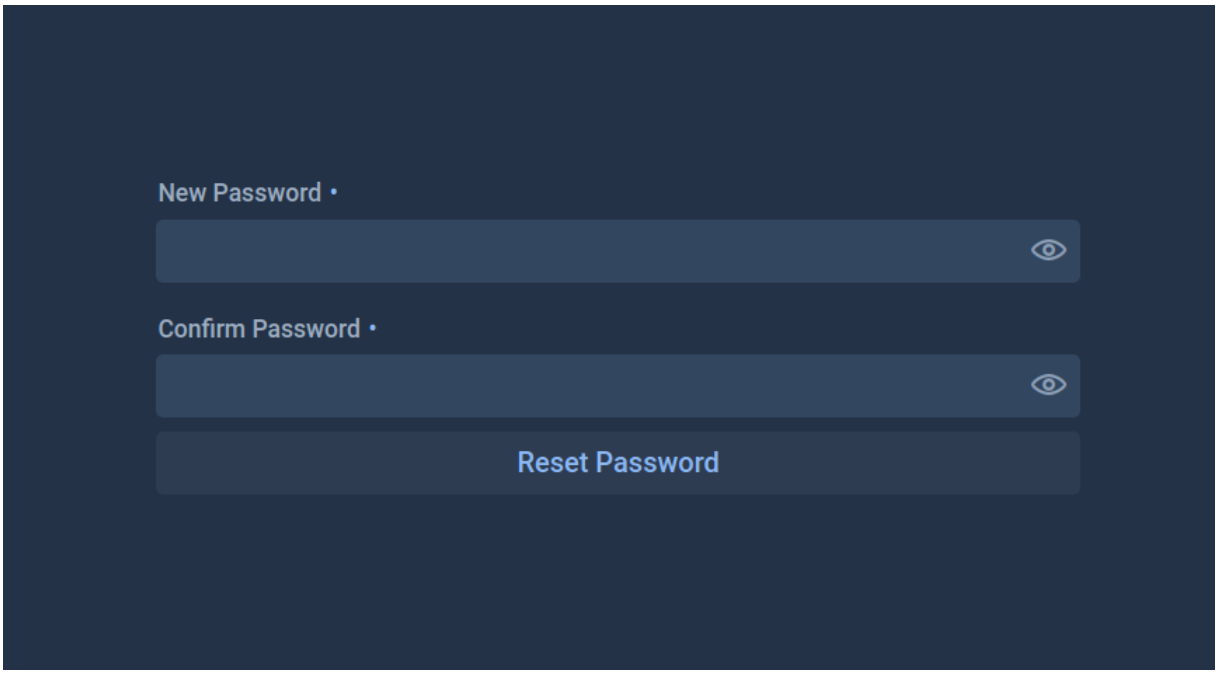


Figure 24. *Renew Password page*

If you do not have an existing account, simply select the "Create an Account" button to get started with the registration process:

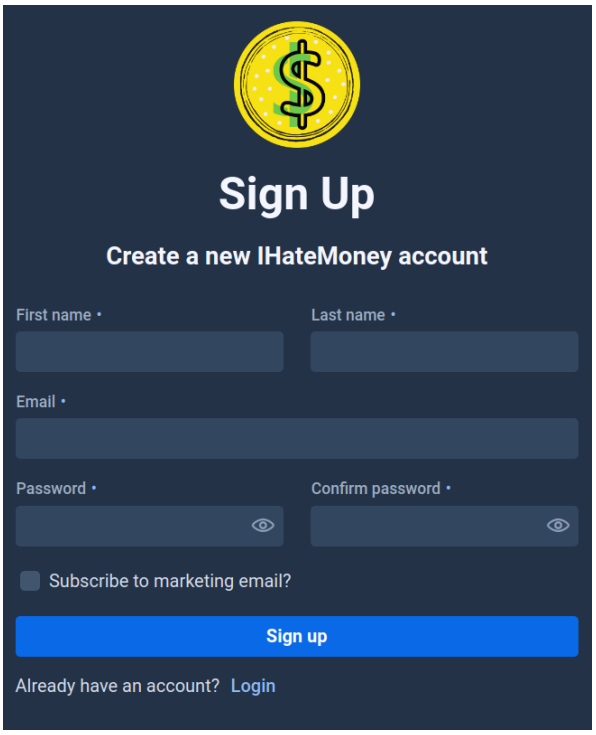


Figure 25. *Register page*

Once logged in, you will be directed to the dashboard. This is your home base for managing your financial activities within the application.

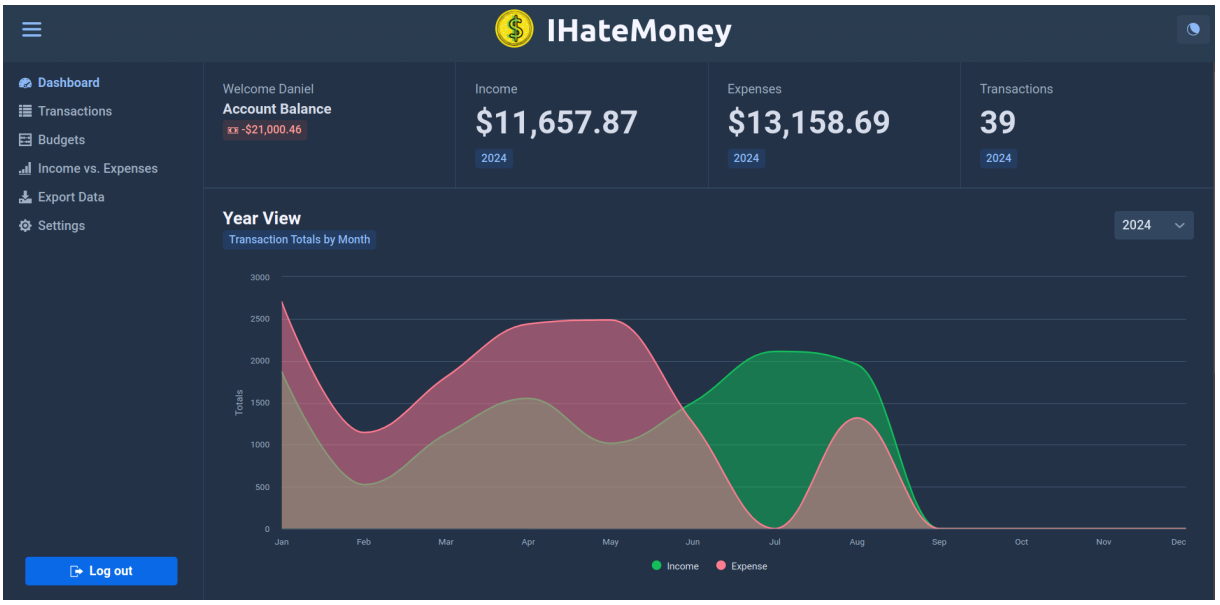


Figure 26. *Dashboard view*

In the Transactions section, you have the flexibility to search, add, or delete transactions. This feature helps you keep track of your financial movements effectively.

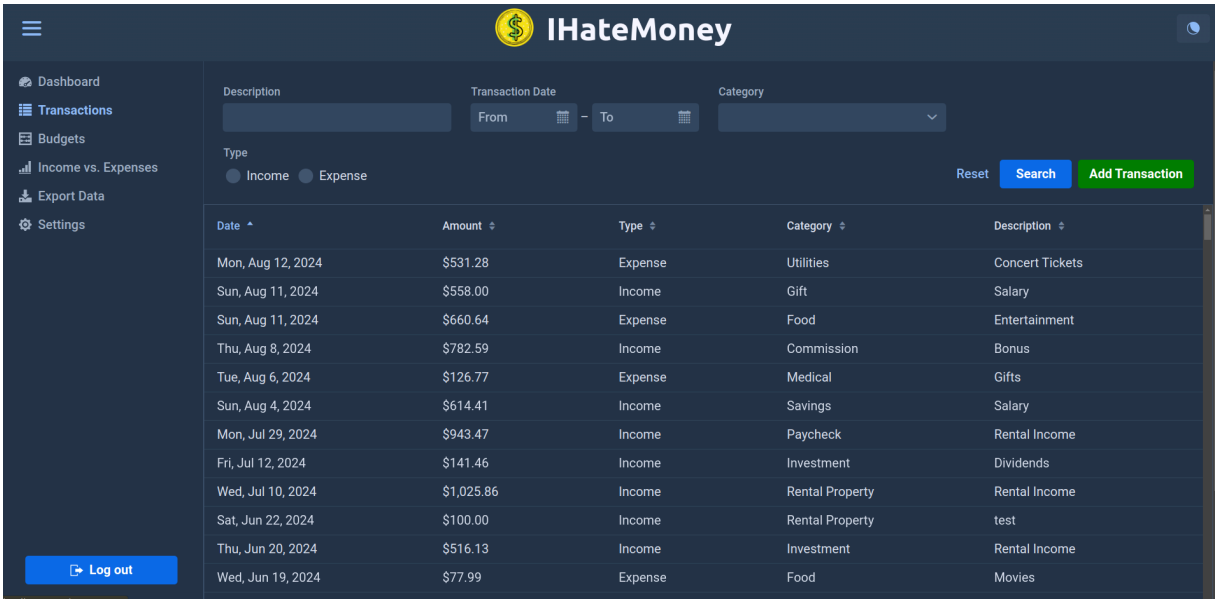


Figure 27. *Transactions view*

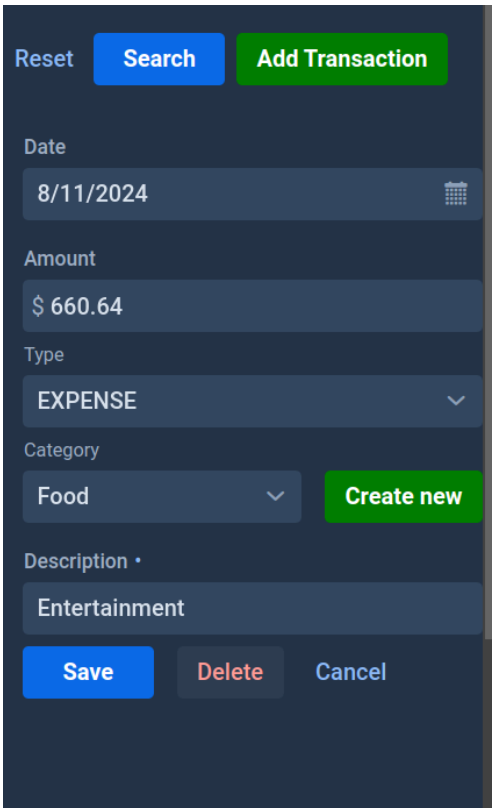


Figure 28. *Update a Transaction*

The Budget section allows you to add new budget plans or delete existing ones, enabling precise financial management tailored to your needs.

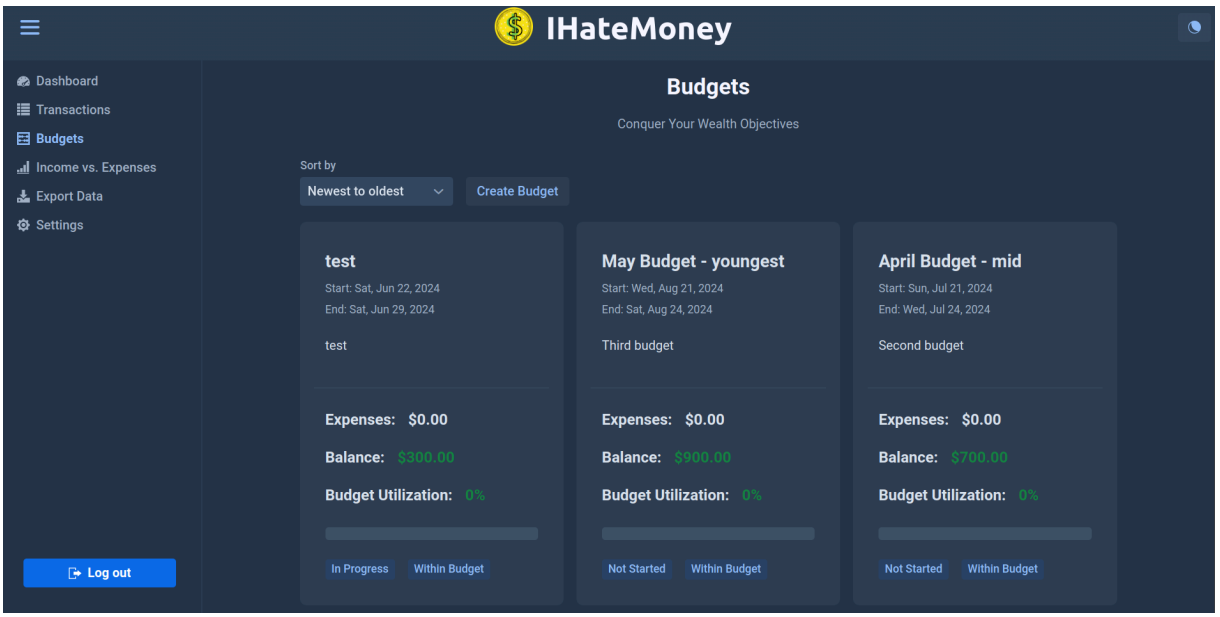


Figure 29. *Budgets view*

The "Income vs. Expenses" view provides a comprehensive overview of your incoming and outgoing financial transactions, offering valuable insights into your financial health.

Figure 30. *Income vs. Expenses view*

Use the Export Data section to download your financial data in CSV format. This tool is especially useful for backing up data or for analysis purposes outside the application.

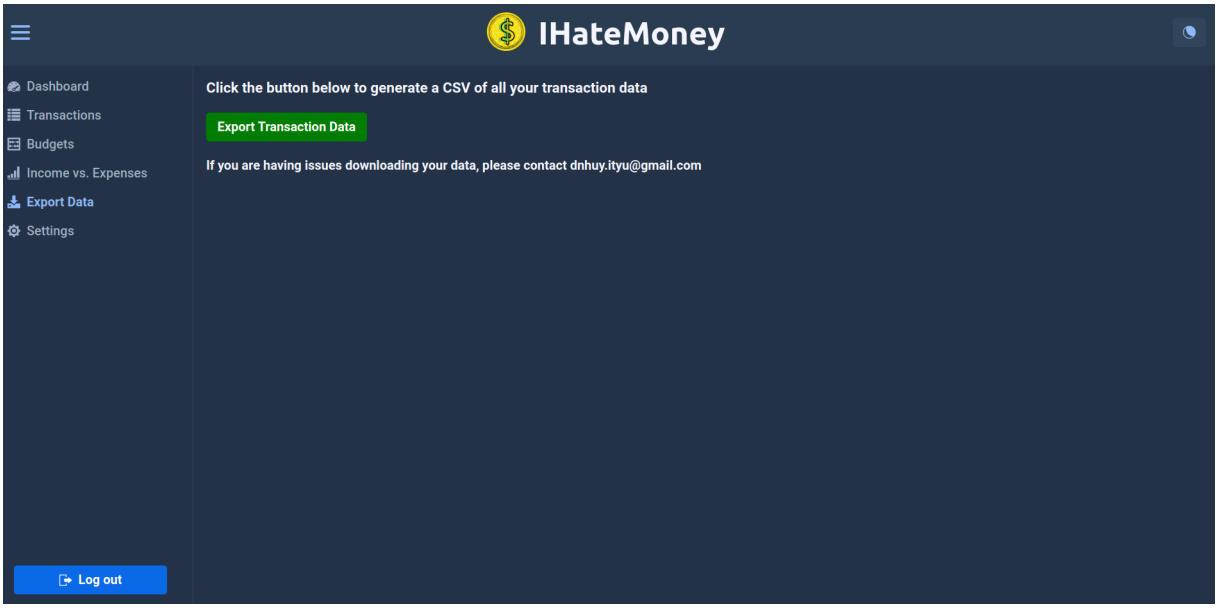


Figure 31. *Export Data view*

The Settings section is designed to help you personalize your application experience. Here, you can:

- Change the application's theme to suit your preference.
- Update personal information to keep your profile current.
- Access advanced settings for actions like deleting all your data.
- Find contact information for administrative support.

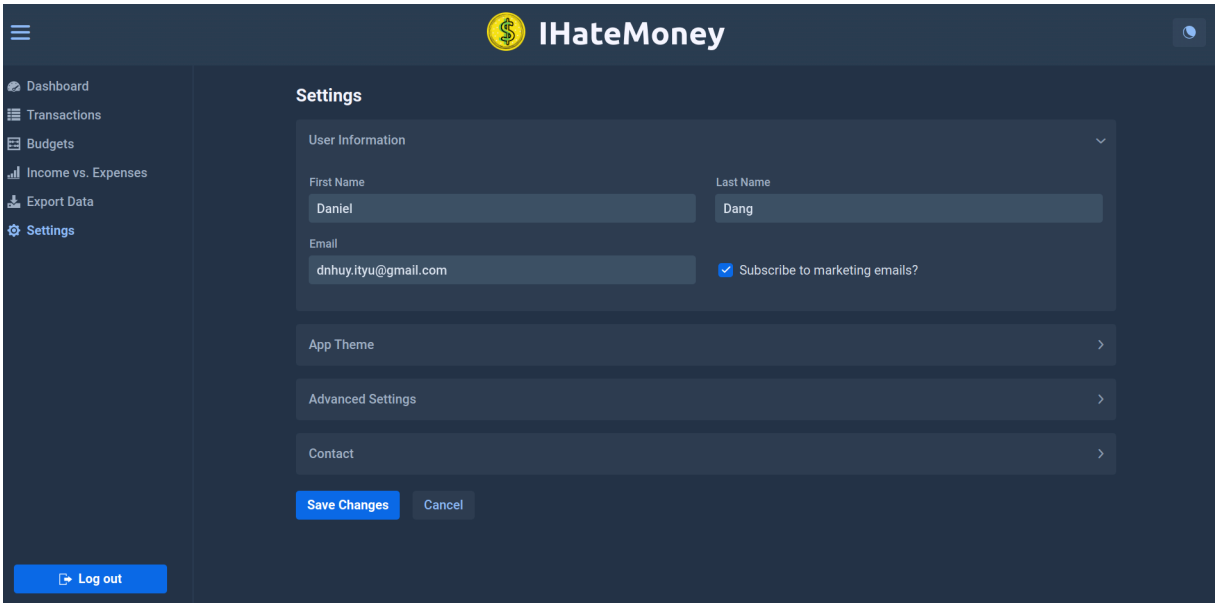


Figure 32. *Settings view*

When you're ready to exit, you can log out securely by clicking on the logout button.

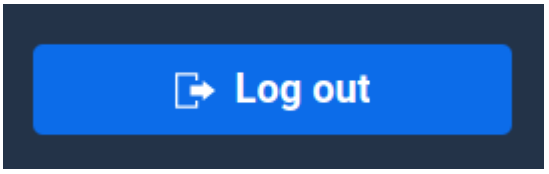


Figure 33. *Log out button*

CHAPTER 5

DISCUSSION AND EVALUATION

5.1. Discussion

Our web app dedicated to money management offers several distinct advantages that cater to modern users' needs. Chief among these is its unparalleled convenience. The app eliminates the constraints of traditional methods reliant on physical records by allowing users to access their financial information anytime and anywhere with an internet connection. This convenience allows for seamless updates and real-time monitoring of economic activities, empowering users to stay on top of their finances effortlessly.

Moreover, the app's capability to deliver real-time updates is a significant advantage over conventional financial management approaches. Unlike manual record-keeping or static spreadsheets, our web app ensures that users receive immediate notifications and insights into their transaction history, account balances, and investment performance. This real-time data enhances financial transparency and enables users to make informed decisions promptly, fostering better financial habits and planning.

Another compelling advantage lies in the app's automation and integration features. The app simplifies the complexities of financial management by automating tasks such as expense categorization, bill reminders, and budget tracking. Integration with bank accounts and credit cards further streamlines processes, allowing for seamless data synchronization and a comprehensive view of one's financial landscape. This integration saves time and reduces the likelihood of manual errors, ensuring more excellent financial planning and analysis accuracy.

However, despite these advantages, it's crucial to acknowledge certain limitations of our money management web app. One notable disadvantage includes concerns over data security and privacy. As with any digital platform dealing with sensitive financial information, ensuring robust security measures against cyber threats and unauthorized access is paramount. Addressing these concerns requires continuous investment in cutting-edge encryption technologies and stringent privacy policies to safeguard users' data effectively.

Furthermore, while the app promotes financial discipline through automation and real-time insights, it may only cater to some users' unique financial needs and preferences.

Some individuals prefer more personalized financial advice or comprehensive wealth management services that go beyond the capabilities of a single web application. Balancing these considerations with ongoing user feedback and technological advancements remains crucial to refining and expanding the app's functionalities to serve a diverse range of users better and effectively.

5.2. Comparison

Several key differences and advantages appear when comparing our money management web app with other well-known apps. Our app strongly emphasizes user interface and ease of use, striving to provide a seamless and intuitive experience for users at every level of financial management proficiency. Unlike some apps that may overwhelm users with many features, our interface is designed to be clean and straightforward, focusing on essential tools that empower users without unnecessary complexity. This simplicity ensures that users can quickly grasp and utilize our app's functionalities without a steep learning curve, setting it apart from more feature-heavy competitors like Mint or YNAB (You Need A Budget).

Our app stands out for its robust automation and integration capabilities in terms of automation and integration. Like Mint, it offers automated expense tracking, budget monitoring, and bill reminders, providing users real-time updates and insights into their financial health. Integration with various financial accounts, including bank accounts and credit cards, further enhances convenience by centralizing financial data in one accessible platform. This integration streamlines the management of finances, offering users a comprehensive view of their financial situation without switching between multiple apps or platforms.

However, while our app excels in simplicity and integration, it may need help with advanced financial planning features compared to competitors like Personal Capital. Apps like Personal Capital offer sophisticated investment tracking, retirement planning tools, and personalized financial advice, catering to users seeking comprehensive wealth management solutions beyond basic budgeting and expense tracking. For users requiring more in-depth financial analysis and advisory services, these advanced features may present a compelling reason to opt for alternative apps specializing in wealth management and investment planning.

Ultimately, our money management web app distinguishes itself through its user-friendly interface, automation capabilities, and seamless integration, making it an ideal choice for individuals and families looking to simplify their day-to-day financial management tasks effectively. As we continue to evolve and enhance our app based on user feedback and technological advancements, we aim to bridge gaps and deliver even more value to users seeking reliable and accessible financial management solutions.

5.3. Evaluation

In conclusion, our money management web app offers a compelling balance of simplicity, automation, and integration compared to other popular apps. Its user-friendly interface ensures accessibility for users of varying financial management skills, promoting ease of use without overwhelming complexity. Like Mint, the app's robust automation features streamline everyday financial tasks such as expense tracking and budget management. In contrast, integrating multiple financial accounts enhances convenience by centralizing financial information.

However, there are considerations when evaluating our app against competitors like Personal Capital, which offers advanced wealth management and investment planning tools. Our app may cater less extensively to users seeking sophisticated investment tracking or personalized financial advice beyond basic budgeting. For such users, alternative apps specializing in comprehensive wealth management may better suit their needs.

Nevertheless, our app remains a robust choice for individuals and families looking to effectively streamline their financial management practices. By enhancing user experience, expanding integration capabilities, and responding to user feedback, we aim to continuously improve and provide more excellent value in simplifying and optimizing personal finance management.

CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1. Conclusion

With the use of technology, the "IHateMoney" app has enabled people to manage their personal finances more effectively. Through the integration of essential functions like transaction management, budget monitoring, and detailed financial reporting, the platform caters to a wide range of customer requirements. Users' financial literacy and decision-making skills have been greatly improved by the application's user-friendly design and real-time information. Our unwavering dedication to enhancing and broadening these resources is what keeps us at the forefront of financial technology as we continue to develop.

6.2. Future work

Short-Term Goals (Next 6-12 Months):

- Feature Enhancements: We plan to introduce customizable and savable budget templates to provide users with more flexibility in managing their finances according to personal or business needs.
- Performance Improvements: Efforts will be made to optimize database queries to enhance application speed and implement load balancing strategies to ensure smooth performance during peak usage times.
- Security Upgrades: We will upgrade our encryption methods for data storage and conduct regular security audits and compliance checks to fortify data protection and user privacy.

Long-Term Goals (1-3 Years):

- Scalability: The application will transition to cloud services that offer dynamic scaling capabilities to handle traffic spikes efficiently. This move will ensure that our infrastructure can support a growing user base without compromising on performance.
- Market Expansion: We aim to launch multilingual support to make our application accessible to non-English speaking users globally. Additionally, partnerships with financial institutions will be explored to offer integrated services, enhancing the value provided to our users.

- Advanced Analytical Tools: Development of predictive analytics features is on the horizon, with the goal of forecasting user spending and savings. These tools will provide users with insights that can help preempt financial issues and optimize savings strategies.

REFERENCES

[1] H. N. Odle‐Dusseau, R. A. Matthews, and J. H. Wayne, “Employees’ financial insecurity and health: The underlying role of stress and work–family conflict appraisals,” J. Occup. Organ. Psychol., vol. 91, no. 3, pp. 546–568, Sep. 2018, doi: 10.1111/joop.12216.

APPENDIX