STREAMLINING PERSONAL FINANCE – AN INTEGRATED SYSTEM FOR AUTOMATED TRANSACTION ENTRY AND ENHANCED VISUALIZATION

By

HARYISH E

2232MCA0058 – 67222200076

A PROJECT REPORT

Submitted to the

FACULTY OF INFORMATION AND COMMUNICATION ENGINEERING

In a partial fulfillment for award of the degree

Of

MASTER OF COMPUTER APPLICATIONS

CENTRE FOR DISTANCE EDUCATION

ANNA UNIVERSITY

CHENNAI 600 025

December, 2024

STREAMLINING PERSONAL FINANCE – AN INTEGRATED SYSTEM FOR AUTOMATED TRANSACTION ENTRY AND ENHANCED VISUALIZATION

By

HARYISH E

ROLL NO.: 2232MCA0058

REGISTER NO.: 67222200076

A PROJECT REPORT

Submitted to the

FACULTY OF INFORMATION AND COMMUNICATION ENGINEERING

In a partial fulfillment for award of the degree

Of

MASTER OF COMPUTER APPLICATIONS

CENTRE FOR DISTANCE EDUCATION

ANNA UNIVERSITY

CHENNAI 600 025

December, 2024

BONAFIDE CERTIFICATE

Certified that the project report titled “STREAMLINING PERSONAL FINANCE – AN INTEGRATED SYSTEM FOR AUTOMATED TRANSACTION ENTRY AND ENHANCED VISUALIZATION” is the Bonafide work of Mr. HARYISH E who carried out the work under my supervision. Certified further that to the best of my knowledge the work reported herein does not form part of any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

Signature of Student

Mr. Haryish E

MCA

Roll No.: 2232MCA0058

Register No.: 67222200076

Signature of Guide

Dr. S. Bose

Professor

Department of Computer Science and Engineering

CEG Campus

Anna University

Chennai - 25

CERTIFICATE OF VIVA-VOCE EXAMINATION

This is to certify that Mr. HARYISH E (Roll No: 2232MCA0058; Register No: 67222200076) has been subjected to Viva-voce-Examination on…… at the Study Centre "**Centre for Distance Education, Anna University**".

Internal Examiner	External Examiner
Name:	**Name:**
Designation:	Designation:
Address:	Address:

Coordinator Study Centre:

ACKNOWLEDGEMENT

First and foremost, I would like to thank Dr. P. NIRMAL KUMAR, Professor and Additional Director, Centre for Distance Education, Anna University, Chennai for his constant motivation throughout the course.

I am most grateful to Dr. S. BOSE, Professor, Department of Computer Science and Engineering, CEG Campus, Anna University, Chennai, for his guidance and constant supervision as well as for providing necessary information throughout the completion of this project.

I also thank my review committee member Dr. S. PRADEEP, Assistant Professor, Centre for Distance Education, Anna University, Chennai for his moral support.

I am also grateful to my parents, friends and colleagues for their timely support and constant words of encouragement. Last but not the least; I extend my thanks to The Almighty.

(HARYISH E.)

ABSTRACT

Managing personal and professional expenses efficiently is a critical challenge for individuals and freelancers alike. Traditional methods of tracking expenses, such as manual data entry and calculations or the use of predefined templates in applications like MS Excel, often fall short of addressing specific user needs. Freelancers face the tedious task of segregating personal and professional transactions, a process prone to errors and inefficiency.

This project introduces a robust and user-centric expense tracker, developed using the Django framework, to address these challenges. By automating expense tracking and categorization, the system empowers users to manage their finances seamlessly. Features include the ability to track income and expenses on a weekly, monthly, and yearly basis, with data visualizations such as pie charts to enhance financial insights. The system also aims to extend functionality by automatically classifying transactions as personal or professional, enabling small business owners to analyse gains and losses effectively.

The solution leverages the scalability and efficiency of Python Django, chosen for its lightweight architecture and its compatibility with machine learning tools for future enhancements. Unlike existing systems, which primarily focus on improving user interfaces for manual entry, this project emphasizes automation and integration with external platforms like e-commerce systems, enabling direct transaction entries. This minimizes manual effort, reduces time and resource costs, and provides a scalable foundation for future innovations in financial tracking. By addressing these real-world issues, this Streamline Personal Finance offers a comprehensive and efficient solution for modern financial management.

திட்ட சுருக்கம்

தனிநபர்கள் மற்றும் ஃப்ரீலான்சர்களுக்கு (*freelancers*) தனிப்பட்ட மற்றும் தொழில்முறை செலவுகளை நிர்வகிக்க குறித்த சவாலாக உள்ளது. பாரம்பரிய செலவுக் கண்காணிப்பு முறைகள், போன்று கையேடு தரவுகள் அல்லது MS Excel டெம்ப்ளேட்கள் (*templates*), ஃப்ரீலான்சர்களின் (*freelancers*) தனித்துவமான தேவைகளை பூர்த்தி செய்ய முடியவில்லை. குறிப்பாக, செலவுகளை தனிப்பட்ட மற்றும் தொழில்முறையாக பிரித்துப் பதிவுசெய்வது சிரமமானதாகவும் பிழைகளுக்கு இடமளிக்கக்கூடியதாகவும் உள்ளது.

இந்த Django வடிவமைப்பின் (*framework*) செலவுக் கண்காணிப்பு அமைப்பு தானியங்கிகரிப்பு (*automation*) மூலம் இந்த சவால்களை தீர்க்கிறது. இது வாரம், மாதம், வருட அளவிலான வருமானம் மற்றும் செலவுகளை பை சார்ட் (*pie charts*) போன்ற தரவுக் காட்சிகளுடன் காண்பிக்க உதவுகிறது. மேலும், பரிவர்த்தனைகளை (*transactions*) தனிப்பட்டது அல்லது தொழில்முறையாக வகைப்படுத்தி சிறு தொழில்முறையினருக்கு (*small business owners*) அவர்களுடைய வருமான-நட்டத்தை எளிதாக கண்காணிக்க உதவுகிறது.

Python Django-வின் எடை குறைந்த கட்டமைப்பும், மெஷின் லெர்னிங் (*machine learning*) கருவிகளுடன் இணங்கும் திறனும், இதனை பாதுகாப்பான மற்றும் விரிவுபடுத்தக்கூடியதாக மாற்றுகின்றன. தற்போதைய அமைப்புகள் முதன்மையாக கையேடு தரவுகளின் உள்ளீட்டை மேம்படுத்துவதில் கவனம் செலுத்துகின்றன, ஆனால் இந்த அமைப்பு தானியங்கி செயலும் e-commerce (*e-commerce*) தளங்களுடன் ஒருங்கிணைப்பையும் கொண்டு நேரடியாக பரிவர்த்தனைகளை பதிவு செய்யும் திறனை வழங்குகிறது. இது நேரமும் வளங்களும் மிச்சப்படுத்தி நவீன நிதி மேலாண்மைக்கு (*financial management*) சிறந்த தீர்வாக செயல்படுகிறது.

TABLE OF CONTENT

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

1.1. OVERVIEW

This project introduces an automated Streamline Personal Finance designed to address the inefficiencies of traditional manual methods and provide a modern financial management solution. Built using the Django framework, the system caters to individuals and freelancers, helping them efficiently track and segregate personal and professional income and expenses.

Key features include user registration, secure data storage, automated categorization of transactions, and comprehensive tracking on weekly, monthly, and yearly scales. Users gain actionable insights through visualizations like bar graphs and pie charts, enabling quick financial decision-making. Unlike existing solutions, which focus only on manual entry improvements, this system emphasizes automation to minimize errors, save time, and enhance accuracy, helping individuals and small business owners in their budgets.

Utilizing Django's lightweight and it architecture, the project is future-ready, allowing integration with machine learning tools for advanced analytics and external platforms like e-commerce systems for direct transaction entries. By automating routine processes and providing secure, user-friendly interfaces, this solution significantly enhances efficiency, accuracy, and user experience in expense management.

1.2. LITERATURE SURVEY

The literature survey analyses expense tracking, financial management, and automation systems, identifying limitations and improvement opportunities. It forms the basis for the proposed Streamline Personal Finance system, enhancing user experience and automation.

P. Thanapal and Mohammed Yasmeen Patel [1] developed an **Income and Expense Tracker** aimed at manual income and expense tracking. The system allowed users to input transaction details manually and offered basic categorization. While it proved effective for

individual users, it lacked advanced features such as automated data entry, email synchronization, or real-time visualization.

Ma Liqian, Tang Siyu, and J. Black Michael [2] explored a **Customized Multi-person Tracker** that utilized machine learning and deep learning for tracking multiple entities simultaneously. Although their work demonstrated high scalability and accuracy, its application was limited to tracking people rather than financial transactions, making it indirectly relevant to the domain of automated expense tracking.

R. Kukade et al. [3] proposed a **TrackEZ Expense Tracker** focusing on simplifying expense management by categorizing transactions and generating reports. However, the system required significant user input and manual data entry, limiting its scalability and usability in real-world scenarios. The proposed project overcomes these limitations by introducing automated data parsing and synchronization features.

J. L. Yeo et al. [4] presented a **Time Scheduling and Finance Management Toolkit** for university students. The system provided basic finance and scheduling assistance but lacked integration with modern e-commerce or banking systems. This system inspired the incorporation of time-based analytics in the proposed system for tracking financial trends over daily, monthly, and yearly periods.

P. Bhatele et al. [5] discussed a financial system emphasizing automation through email synchronization and data filtration. Their work laid a strong foundation for transaction tracking, but their reliance on external APIs presented challenges in terms of security and feasibility. The proposed project addresses these gaps by simulating API environments through mocked services while ensuring data accuracy.

A. Tamizhselvi et al. [6] introduced **Financial and Future Expense Prediction Models** leveraging frequent patterns and machine learning. Their approach focused on predicting spending trends using past data, demonstrating over 80% accuracy. While their work was promising, it required extensive datasets and computational power, which may not be feasible for small-scale implementations. This aspect has been considered for future work in the proposed project.

Bhujang et al. [7] designed an **Expense Tracker and Budget Planner** with a focus on budget planning and manual entry for expense tracking. While useful for individual users, it lacked automation or integration capabilities, limiting its scope for professional use. The proposed project builds on these findings by incorporating automated transaction classification and synchronization.

1.2.1. FUTURE DIRECTIONS

The literature points to several areas for enhancement, such as real-time API integrations for email and banking systems, machine learning-based automated transaction categorization, and advanced predictive analytics for financial planning. These advancements would address existing limitations and provide a unified, scalable financial management solution.

1.2.2. GOALS OF PRESENT INVESTIGATION

The above literature survey highlights significant advancements in financial management systems. However, the following limitations in existing systems necessitate further investigation:

1. Lack of automated transaction entry from email systems or e-commerce platforms.
2. Absence of real-time synchronization and consolidated financial overview for users.
3. Limited scalability and manual effort involved in categorizing and analyzing transactions.

To address these gaps, the proposed system introduces automated data parsing, email synchronization, and dynamic visualizations, providing users with a comprehensive and secure financial management solution.

1.2.3. COMPARISON TABLE

The below Table 1.1 exhibits how expense tracking and visualization implemented in terms of traditional method (paper pen/MS Excel), Existing Systems (Previously implemented approaches) and Proposed System. This helps us in analyzing strength and weaknesses of this project.

Criteria	Traditional Methods	Existing Systems	Proposed System
Automation Level	Manual entry	Semi-automated	Fully automated
Data Security	Low	Medium	High
User Interface	Complex	Moderate	User-friendly
Integration Capabilities	None	Limited	Extensive
Cost	Low	Medium	High
Scalability	Low	Medium	High
User Support	None	Basic	Comprehensive
Data Accuracy	Prone to errors	Improved but inconsistent	High accuracy with automation
Time Efficiency	Time-consuming	Moderate	Highly efficient
Visualization Tools	None	Basic charts	Advanced visualizations (e.g., pie charts, bar graphs)
Transaction Categorization	Manual	Limited automation	Automated with NLP and templates (not in current scope)

Table 1.1 Comparison table of Expense Tracking System in Traditional, External and Proposed methods

1.2.4. CONCLUSION

This literature survey reveals that while many existing systems for expense tracking offer basic functionalities, most still rely heavily on manual data entry. The proposed system aims to improve upon these by automating the categorization of transactions and integrating with external platforms like e-commerce and banking systems. By leveraging machine learning and predictive analytics, the proposed system will offer a more advanced and efficient solution for modern expense tracking.

1.3. EXISTING AND PROPOSED SYSTEM

The existing systems for tracking expenses primarily focus on manual data entry or limited automation. These systems are designed to help users categorize their transactions and visualize their expenses. While some systems provide visual aids such as pie charts and budgeting tools, they lack advanced features like automated data integration and predictive analysis.

Common Features on Existing Systems:

1. **Manual Data Entry**: Most systems rely on users manually inputting transaction details, which is time-consuming and prone to human error.
2. **Expense Categorization**: Users can assign categories to their expenses, but this often requires manual effort.
3. **Budget Management**: Some tools allow users to set budgets and compare them with actual spending.
4. **Visualization Tools**: Basic graphical representations, such as pie charts, are commonly used to provide insights into spending habits.

Limitations of the Existing System:

1. **Lack of Automation**: Manual data entry is tedious and limits efficiency.
2. **Integration Challenges**: Existing systems rarely integrate with banking or e-commerce platforms to automate transaction entries.
3. **Inconsistency**: Manual processes often result in inconsistent data due to errors or oversight.
4. **Limited User-Specific Features**: Existing tools don't cater to niche user groups like freelancers, who need to segregate personal and professional expenses.
5. **Scalability Issues**: Most systems are not built to handle large datasets or adapt to future requirements, such as predictive analytics.

The proposed system is a digital Streamline Personal Finance built using the Django framework, designed to overcome the limitations of existing systems. It introduces

automation, categorization, and advanced features tailored to individuals and freelancers, ensuring efficient and secure financial management.

Key Features of the Proposed System:

1. **Automation**: Automatically categorizes transactions as personal or professional, reducing manual effort.
2. **Integration**: Direct integration with e-commerce platforms and banking systems for streamlined transaction entries.
3. **Secure Data Management**: Robust authentication ensures secure storage and retrieval of user data.
4. **Advanced Visualization**: Provides graphical insights into income and expenses on various time scales.
5. **Scalability**: Supports future enhancements like predictive analytics and machine learning insights.
6. **User-Centric Design**: Intuitive interface tailored for individuals and freelancers.

Advantages of the Proposed System:

1. **Time Efficiency**: Automation saves significant time by reducing manual data entry.
2. **Enhanced Accuracy**: Minimizes human intervention, ensuring higher data consistency.
3. **Customization for Freelancers**: Segregates personal and professional finances, addressing a critical need.
4. **Seamless Integration**: Real-time transaction integration with external platforms improves user experience.
5. **Future-Ready**: Adaptable for advanced analytics and evolving financial tracking needs.

1.4. OBJECTIVE AND SCOPE OF THE PROJECT

The primary **objective** of this project is to develop an **automated Streamline Personal Finance** using the **Django framework**, designed to streamline the process of tracking and managing **personal** and **professional finances**. The system aims to provide a

secure, **user-friendly platform** for individuals and freelancers, enabling them to **track**, **visualize**, and **analyse** their financial data effectively.

The key objectives include:

1. **Automation of Expense Tracking**: Replace **manual tracking** methods with an **automated system** to ensure **accuracy** and reduce time spent on **data entry**.
2. **Secure Data Management**: Implement **robust data storage mechanisms** to protect **user financial information**, ensuring **privacy** and **security**.
3. **Visualization of Financial Data**: Provide intuitive **visualizations**, such as **pie charts**, to help users understand **income** and **expense patterns** over **weekly**, **monthly**, and **yearly** periods.
4. **Simplified Financial Analysis**: Offer tools that enable users to **segregate personal** and **professional transactions**, aiding in effective **financial planning** and **management**.
5. **Enhanced User Experience**: Design a **user-friendly interface** for seamless data entry, retrieval, and financial analysis.
6. **System Integration**: Lay the groundwork for future features, such as **automatic transaction categorization** and **integration with e-commerce platforms** for direct **transaction entry**.

By addressing the inefficiencies of traditional systems and leveraging the capabilities of **Django**, this project provides a solid foundation for an **automated**, **secure**, and **scalable financial management solution**.

1.5.ORGANIZATION OF THE REPORT

This report systematically covers the process of conceptualizing, planning, designing, developing, and testing the Streamline Personal Finance system. It outlines each phase of the software development lifecycle in an object-oriented environment, providing a clear view of the project's scope and execution. Below is an overview of the chapters:

Chapter 1: Requirement - This chapter outlines the functional and non-functional requirements of the expense tracker, detailing its goals, performance expectations, and constraints, in line with IEEE 830-1998 SRS standards.

Chapter 2: Specific Requirement - This section delves into the technical requirements, including UML, DFD, and ER diagrams, and details on external interfaces, logical databases, and performance attributes.

Chapter 3: Design and Test Plan - This chapter covers the system's architectural design and testing strategies, including database schemas, user interface layouts, and backend structures. It also details the testing plan, which includes unit, integration, and user acceptance testing.

Chapter 4: Implementation, QA, and Results - This section describes the development of system modules using Django, covering user registration, expense entry, and data visualization. It emphasizes quality assurance processes like debugging, security, and performance optimization, and evaluates system functionality and user feedback.

Chapter 5: Conclusion - The report concludes by summarizing the project's achievements, highlighting how it bridges the gap between manual and automated financial tracking. It addresses challenges like email synchronization and transaction filtering and suggests future enhancements such as live API integration and machine learning-based categorization.

CHAPTER 2
REQUIREMENT SPECIFICATION

The "**Streamlining Personal Finance"** system automates transaction tracking by reading emails and extracting transaction details. It connects to the user's email account to retrieve relevant emails, such as receipts and purchase confirmations. Using predefined templates and natural language processing, the system identifies key information like transaction amounts, dates, merchant names, and categories. This data is then automatically entered into a secure, centralized transaction database. Users can view, manage, and visualize their financial information through graphical representations like pie charts, providing insights into spending patterns and budget allocations. This integration reduces manual entry, minimizes errors, and ensures accurate, real-time updates to financial records, enhancing the user experience.

Here's a more concise and point-based explanation on how actually we expect the system to work:

- **Email Integration:** The system connects to the user's email account to retrieve emails containing transaction details (e.g., receipts, purchase confirmations, or bank statements).
- **Transaction Extraction:** The system scans these emails using predefined templates and natural language processing to identify and extract key transaction details such as:
    - Transaction amount
    - Date of transaction
    - Merchant name
    - Transaction category
- **Database Recording:** The extracted transaction details are automatically added to the **transaction database** for secure storage.
- **Viewing and Management:** Users can view and manage their transactions from the system, with all data automatically updated and organized.
- **Data Visualization:** The system offers visual representations of financial data (e.g., **pie charts**) to help users track spending patterns and see their budget breakdown.

This method automates the process of transaction entry, reducing errors, and ensuring accurate financial records.

2.1. OVERVIEW

The software addresses common financial management issues for individuals and freelancers by automatically extracting transaction details from emails, such as receipts and purchase confirmations, eliminating the need for manual entry and ensuring accurate, up-to-date financial records. Transactions are securely stored in a database, where users can view and manage them. The system includes visualization tools like pie charts to help users understand spending patterns and budget allocations, providing real-time financial health updates. Designed with a simple, intuitive interface, it allows easy registration, login, and data management, while ensuring secure data storage. This automated solution offers a clear, comprehensive, and visual overview of finances, enhancing efficiency and accuracy.

2.1.1. PRODUCT PERSPECTIVE

The **"Streamlining Personal Finance"** system is designed to automate personal and professional expense tracking, improving accuracy, ease of use, and overall financial management. This section outlines the various interfaces and constraints the system will operate under, from both a software and hardware perspective, as well as the end-to-end process it will follow.

System Interfaces:

The system will interact with **external services** to collect transaction data. It will:

- **Email Integration**: The system will fetch transaction details from user email accounts. It will need access to email providers (such as Gmail, Outlook) through their APIs to extract transactional data like receipts and invoices.
- **Bank Integration**: Future iterations of the system may integrate with **banking APIs** to automatically fetch transaction data from the user's bank account, further minimizing manual data entry.
- **E-Commerce Integration**: For freelancers, the system will allow integration with e-commerce platforms like **PayPal**, **Stripe**, or **Amazon** to directly retrieve transaction records related to business activities.

However, as there is unfeasibility or may not open to public environment in interacting with direct external services APIs like Banking APIs, PayPal Amazon, Gmail and Outlook, we are building the testing environment to exhibit how it is working by developing the following system interactions,

- **Email Integration**: For this we are building a mocked / un-real environment / gateway which implements under local server to send and receive SMS to the user (configured with mock email ID)
- **Data Filtration**: We require a system where it will convert the extracted emails (mocked) into required data using REGEX module in python and utilizing in inserting to transactions database for record and visualization.
- **E-commerce Integration**: For this system, we develop the system that will contain transaction entry required to fill and authentication to map which user this transaction belongs to and so it will send to relevant users.

User Interface (UI):

The user interface will be **web-based**, designed to be intuitive and easy to use:

- **Dashboard**: Users will interact with a dashboard showing visualized financial data like pie charts for income and expenses.
- **Transaction Entry**: A simple form-based interface will allow users to manually enter transactions when needed.
- **Data Visualizations**: Pie charts and reports will visually display income and expense distributions, with the ability to filter by categories, dates, and more.
- **E-commerce Biling service:** Building the frontend application for entering the transaction details, emphasizing the common approach that user purchases something via online or in-store.
- **Authentication**: Secure login and registration screens will ensure that users' financial data is protected.

Hardware Interfaces:

The system will run on standard **client-server architecture**:

- The user will need a **PC, laptop, or mobile device** to access the web app.

- The system does not have specific hardware requirements beyond these common devices and internet access.

Software Interfaces:

The software system will interact with several key technologies:

- **Django Framework**: The core framework used to develop the application, ensuring rapid backend development, security, and scalability.
- **Frontend Technologies**: HTML, CSS, and JavaScript (with libraries like Bootstrap or React) will be used for the user interface to ensure responsiveness and interactivity.
- **Database**: The backend will use **SQL (SQLite in this case)** to store transaction data, user details, and other necessary information.

Communication Interfaces:

The system will facilitate communication between the user and the server through standard **HTTP/HTTPS protocols**:

- **REST APIs** will be used for data exchange between the frontend and backend (e.g., to retrieve transaction data or visualize spending reports).
- **Email Communication**: The system will be able to send users notifications regarding updates, transaction summaries, or authentication activities (password reset, etc.).

Memory Constraints:

The system will operate efficiently within typical **memory limits**:

- The database will be optimized for scalability but will not require excessive memory or storage space, as financial records are relatively small in size.
- **Frontend memory usage** will be kept minimal, as the web app will only process a limited amount of real-time data (such as charts or recent transactions).

Operations:

The system will be designed to handle the following operations:

- **Data Entry**: The system will support both manual entry and automated transaction entry via email and bank APIs.
- **Data Processing**: Transaction data will be automatically classified (eventually) and stored in the database for later viewing and visualization.
- **User Authentication**: The system will handle user registrations, login sessions, and ensure the security of users' personal and financial data.
- **Report Generation**: Users can generate financial reports based on specific filters (e.g., monthly, yearly, personal, or business).

Site Adaptation Requirements:

The system will be designed to run across various platforms, ensuring accessibility and usability:

- **Browser Compatibility**: The system will be accessible via **Chrome, Firefox, Safari**, and other modern browsers.
- **Responsive Design**: The user interface will be responsive to fit different screen sizes and resolutions, enabling access from desktops, tablets, and mobile devices.
- **Security Adaptations**: The system will implement encryption protocols to ensure that sensitive data is protected from unauthorized access.

2.1.2. PRODUCT FUNCTION

The product offers several key functionalities to help users manage their expenses and transactions efficiently. Users, whether individuals or freelancers, can register, log in securely, and access their personalized dashboard. The system allows for manual entry of transactions, particularly for expenses not captured through email synchronization, and automatically synchronizes and extracts transaction details from the user’s email to ensure comprehensive expense tracking. It maintains a unified list of all transactions, both manual and auto synced, for easy review and analysis.

Data visualizations, such as pie charts and summaries, provide users with insights into their financial activity, supporting filtering and categorization of transactions based on parameters like date, type, and source. Merchants can input purchase and billing details

directly into the system, associating them with user accounts, and transactions are authenticated by verifying the user's credentials against their existing records.

The system extracts transaction-related information from user emails using email provider APIs, filters and parses the extracted data to identify and store relevant transaction details in the database, mapped to the appropriate user account. Administrators can monitor and manage user accounts within the system and can delete user accounts upon request, ensuring compliance with user privacy requirements.

2.1.3. USER CHARACTERISTICS

The system is designed for diverse user groups with varying technical expertise and needs. Below are the defined user characteristics for the system:

1. **Individual Users or Freelancers:**
    - **Goals:**
        - Track income and expenses with minimal manual effort.
        - Visualize financial activity to manage budgets effectively.
    - **Needs:**
        - Secure login and data access.
        - User-friendly interfaces for manual and automated transaction management.
        - Accurate data synchronization from email accounts.
2. **Merchant Retailers:**
    - **Goals:**
        - Provide seamless transaction entry for customer purchases.
        - Notify customers of completed transactions via email.
    - **Needs:**
        - Reliable transaction recording mapped to user accounts.
        - Authentication features to validate customer details during transactions.
3. **System Administrators:**
    - **Goals:**
        - Monitor and manage system users and their activities.

        - Ensure system security and integrity.
    - **Needs:**
        - Tools for user account monitoring, updates, and deletions.
        - Administrative interfaces to handle user requests securely.
4. **Mocked System Developers and Testers (for development purposes):**
    - **Goals:**
        - Validate the functionality of mocked external interfaces like email extraction and transaction entry.
        - Ensure data integrity and user flow during testing.
    - **Needs:**
        - Access to test environments replicating email, e-commerce, and transaction scenarios.
        - Debugging tools to resolve issues during development and testing.

2.1.4. OPERATING ENVIRONMENT

The system is designed to operate in a modern web-based environment, requiring both client and server-side components. Below are the specifications of its operating environment:

1. **Client Environment:**
    - **Browser Compatibility:** Supports modern browsers like Google Chrome, Mozilla Firefox, Microsoft Edge, and Safari.
    - **Device Compatibility:** Accessible on desktops, laptops for better UX.
    - **Network Requirement:** Requires stable internet connectivity for real-time synchronization and system interactions.
2. **Server Environment:**
    - **Hosting Platform:** Can be deployed on cloud-based platforms (e.g., AWS, Heroku, or Azure) or local servers for development/testing.
    - **Backend Framework:** Python Django framework for robust and scalable application development.
    - **Database Management:** SQLite to store and manipulate transaction and user data.

2.1.5. CONSTRAINTS

System Constraints:

The application operates as a mocked system, limiting real-time interaction with external services (e.g., banking APIs or live email systems). Resource-intensive operations (e.g., email extraction and data processing) may introduce delays on low-performance servers.

Technological Constraints:

Requires Python Django expertise for development and maintenance. This is Limited by the capabilities of the REGEX module for data filtration, which may require enhancements for complex parsing scenarios.

Security Constraints:

All mocked data and interactions are confined to a controlled testing environment, ensuring no sensitive data is used or exposed. User authentication relies on secure credential storage and management techniques, adhering to basic security best practices.

This detailed consideration of the operating environment and constraints ensures the system operates efficiently while staying within its defined scope.

2.2. SPECIFIC REQUIREMENT

The system gives the detailed specification of all the features, their performance and other requirements of the system. With these specifications, we will be able to give complete view of the system.

2.2.1. EXTERNAL INTERFACES

The system interacts with several interfaces to provide its core functionalities. These include:

- **Hardware Interface:**
    - Requires standard computing devices like desktops, laptops with internet connectivity. No additional hardware is required as all operations are software-based.

- **Software Interface:**
    - **Backend Framework:** Python Django for handling business logic and data management.
    - **Database:** SQLite for storing transaction and user data.
    - **Front-End Technologies:** HTML, CSS, JavaScript for building the UI.
    - **Mock Servers:** Simulate email and e-commerce transactions to validate system capabilities.
- **Communication Interface:**
    - Supports HTTP/HTTPS protocols for secure data transfer between client and server. Mocked APIs are utilized for testing email extraction and transaction processing.

2.2.2. USE CASES

Actors:

User**,** Admin

Use Cases:

1. **User Registration**
    - **Description**: Allows a new user to sign up by providing necessary details like email, username, and password.
    - **Precondition**: User has not registered previously.
    - **Postcondition**: User is successfully registered and can log in.
2. **User Login**
    - **Description**: Enables an existing user to log in using their username/email and password. This use case appeared in Figure 2.1.
    - **Precondition**: User has already registered and has valid credentials.
    - **Postcondition**: User is authenticated and redirected to the dashboard.
3. **Add Transaction**
    - **Description**: Enables a user to add a new financial transaction, including details like amount, category, and date. This appeared on Figure 2.1.
    - **Precondition**: User is logged in and has access to the transaction management page.

- **Postcondition**: New transaction is added and visible in the transaction list.

4. **View Transactions**
    - **Description**: Displays a list of all the user's financial transactions, including date, category, and amount. This appeared on 2.1
    - **Precondition**: User is logged in.
    - **Postcondition**: User can view their transaction history.
5. **Generate Analytics**
    - **Description**: Generates visual reports (e.g., pie charts, bar graphs) to analyze spending patterns and transaction categories.
    - **Precondition**: User has transactions recorded in the system.
    - **Postcondition**: Visualizations are rendered and displayed to the user.
6. **Admin Manage Users**
    - **Description**: Allows an admin to manage user accounts, including viewing, editing, and deactivating accounts. Admin and his management happen as per use case diagram Figure 2.1.and Figure 2.2.
    - **Precondition**: User is an admin with login credentials.
    - **Postcondition**: Admin can view and manage user accounts.
7. **Filter Transaction Data**
    - **Description**: Allows an admin to view and manage user transactions, including editing or deleting records. The process is mentioned in use case diagram Figure 2.2.
    - **Precondition**: User is an admin with login credentials.
    - **Postcondition**: Admin can modify transaction data for users.
8. **Purchase entry**
    - **Description:** Allows users or merchant vendors to enter the transaction details. This is Framed as use case diagram named as “Merchant Application” displayed in Figure 2.3.
    - **Postcondition:** The system can parse the transaction data to mocked IMI system for email notification.
9. **Mock Email Service**

- **Description:** Simulates email interactions for testing purposes, mimicking merchant email notifications. This appears in Figure 2.3 and the detailed flow are framed in use case Figure 2.2 as Email Integration Systems.
- **Precondition:** Mock email service must be configured.
- **Postcondition:** Simulated transactions are successfully logged for testing.

2.2.2.1. USE CASE DIAGRAM

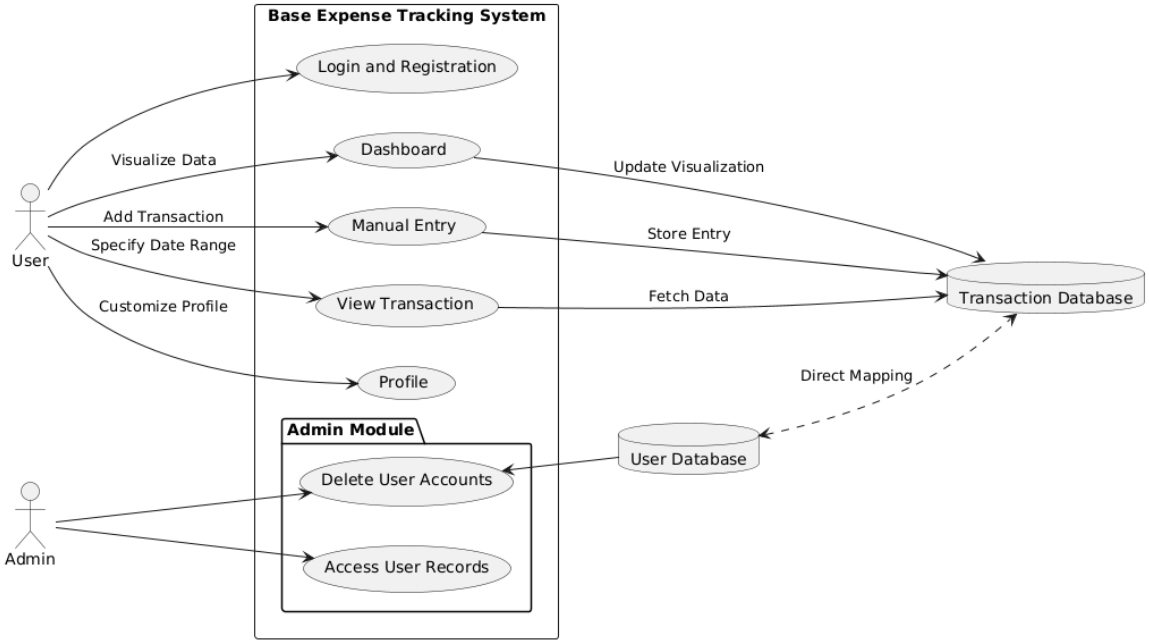

Figure 2.1 Use Case Diagram for Base Expense Tracking Application

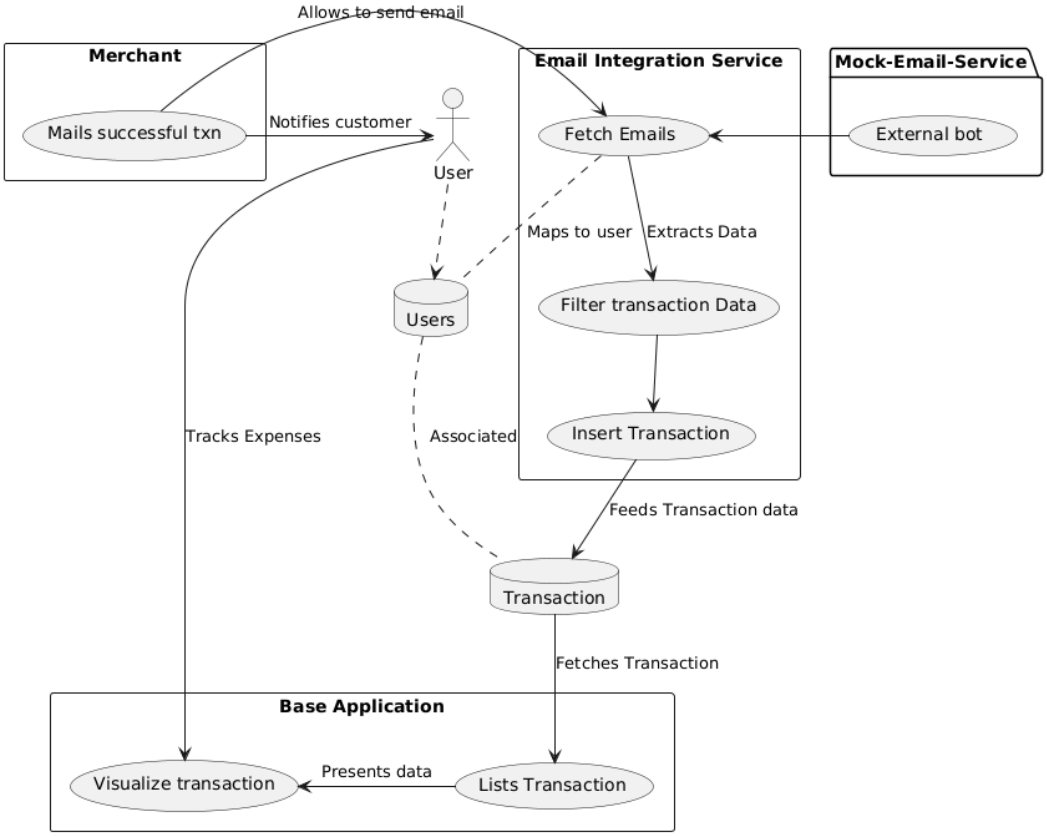

Figure 2.2 Use case Diagram for Email Integration System

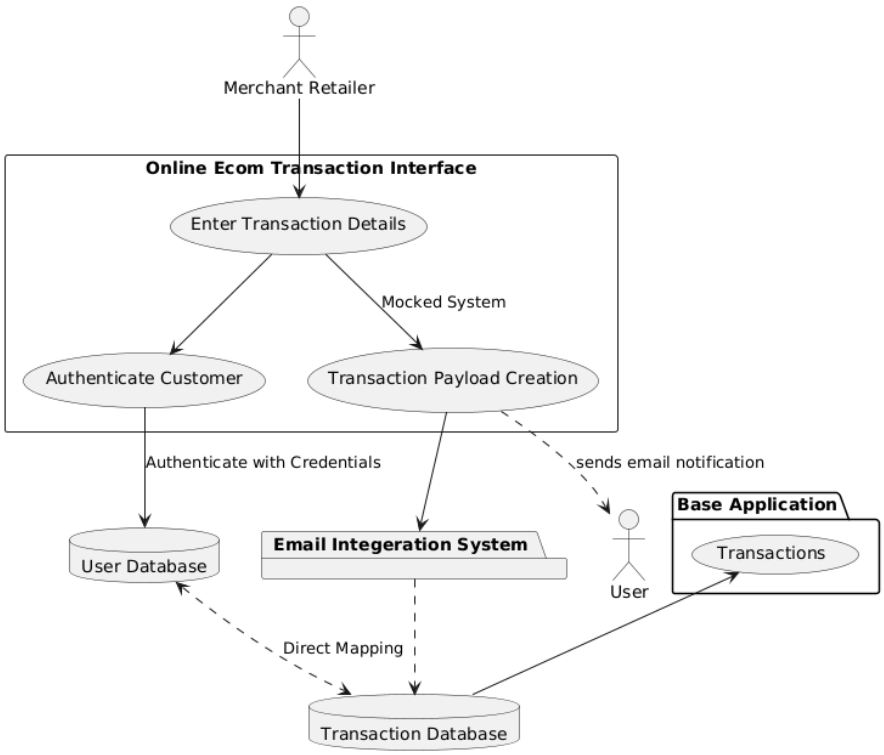

Figure 2.3 Use Case Diagram for Merchant Application

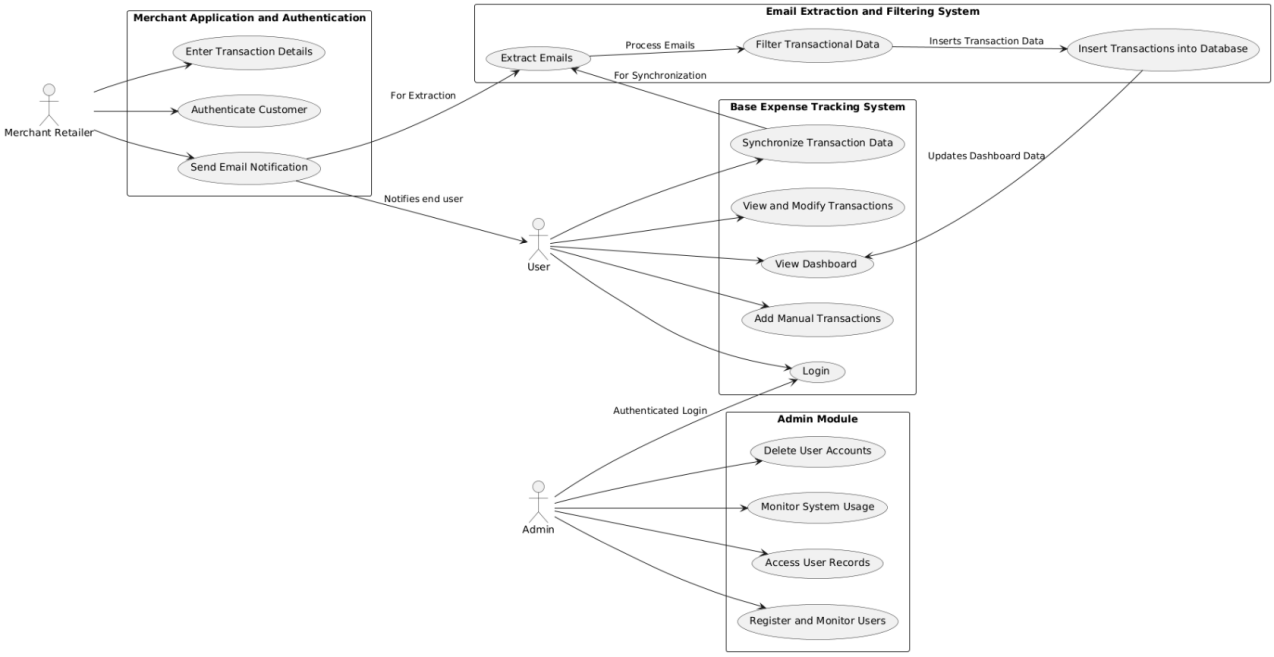

Figure 2.4 Use Case Diagram for Overall System at high level

2.2.3. DATA FLOW AND ER DIAGRAMS

This section presents a comprehensive visual representation of the system's design, showcasing how data and processes interact across different modules. The diagrams include:

- Entity-Relationship (ER) Diagram
- Data Flow Diagram (DFD)

2.2.3.1.DATA FLOW DIAGRAM

The DFD [18] drawn in the Figure 2.5, illustrates the flow of data between external entities (users, merchants, and email systems) and the internal components of the system. Key processes such as transaction management, email extraction, and user authentication are highlighted, showing how data is input, processed, and stored in the relevant databases.

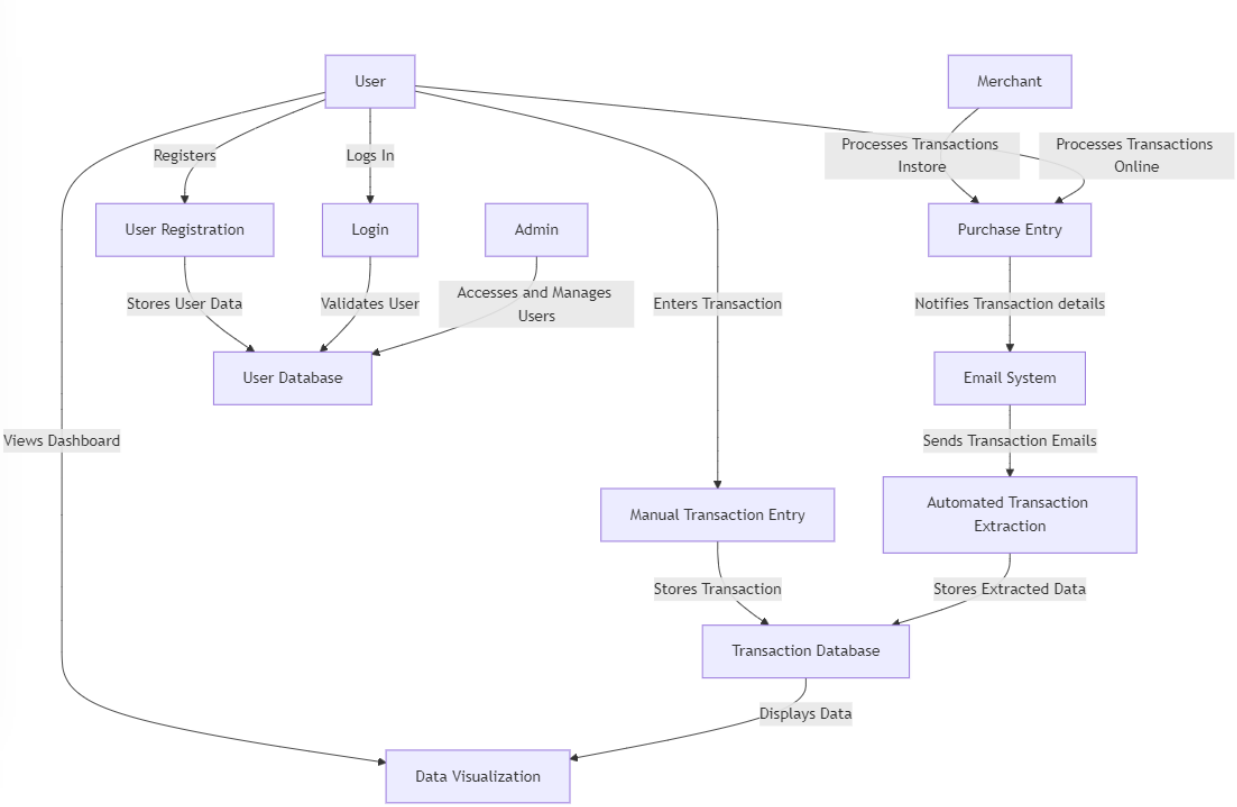

Figure 2.5 Data flow diagram for entire application

2.2.3.2. ER DIAGRAM

The ER diagram [6] models the database structure, defining the relationships between entities like *Users*, *Transactions*, and *Merchants*. It ensures data normalization and provides a clear blueprint for database schema implementation.

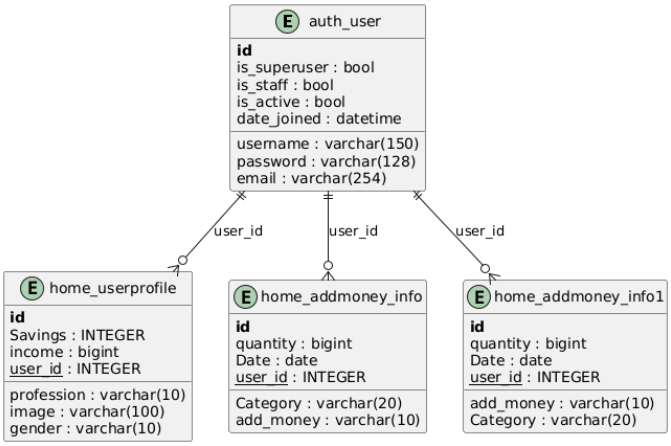

Figure 2.6 ER Diagram for Commutation to Base Application

2.2.4. UML DIAGRAMS

- **Use Case Diagrams:** Define user interactions with the system, covering use cases (mentioned in Figure 2.1, Figure 2.2, Figure 2.3, Figure 2.4) such as logging in, managing transactions, and merchant operations.
- **Class Diagrams:** Figure 10 Depict the system's object-oriented design, including core classes like *User*, *Transaction*, *Merchant*, and their attributes, methods, and relationships.
- **Sequence Diagrams:** Figure 2.7, 2.8, 2.9 Illustrate the step-by-step flow of processes such as email extraction and transaction synchronization, ensuring smooth communication between components.

2.2.4.1. SEQUENCE DIAGRAMS

Base Expense Tracking Application: As mentioned in diagram Figure 2.7, User logs in and interacts with the system to add transactions and synchronize data. The Synchronization Service fetches transaction data from Email System after user synchronization.

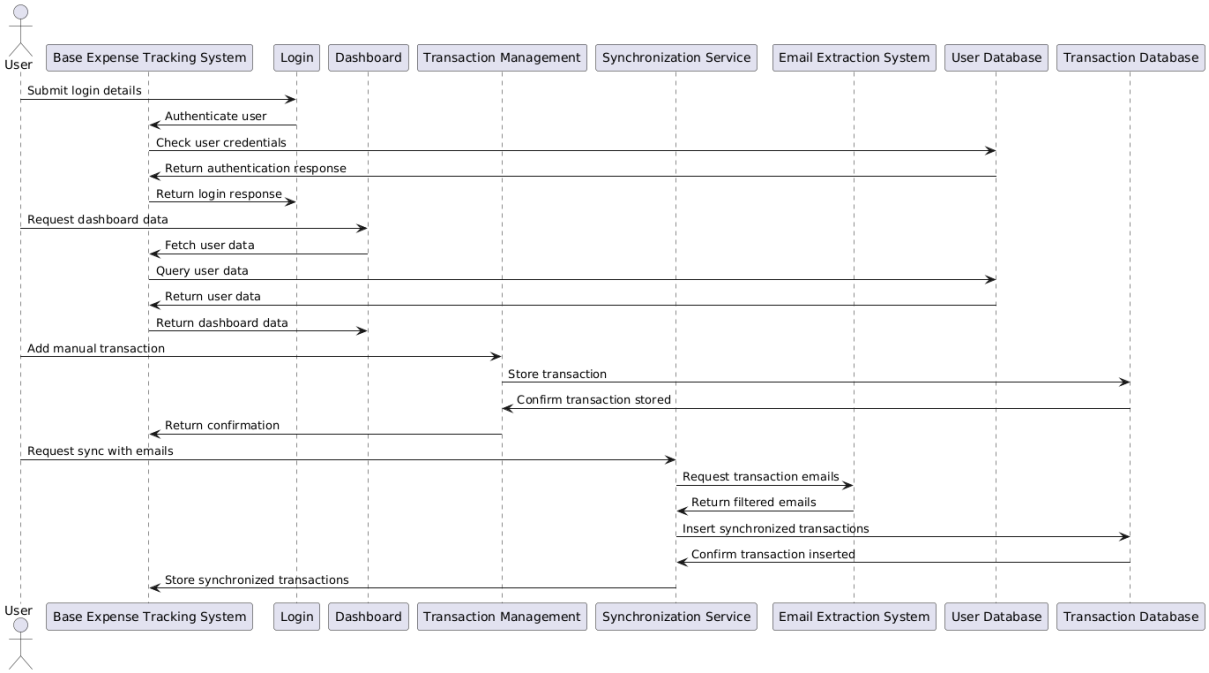

Figure 2.7 Sequence Diagram for E2E flow of Base Expense Tracker

Merchant Application and Authentication: The **Merchant** enters transaction details, and the system verifies the customer using **Authentication**. Once validated, the transaction is stored, and a notification email is sent to the customer. This entire process explicitly mentioned in Figure 2.8.

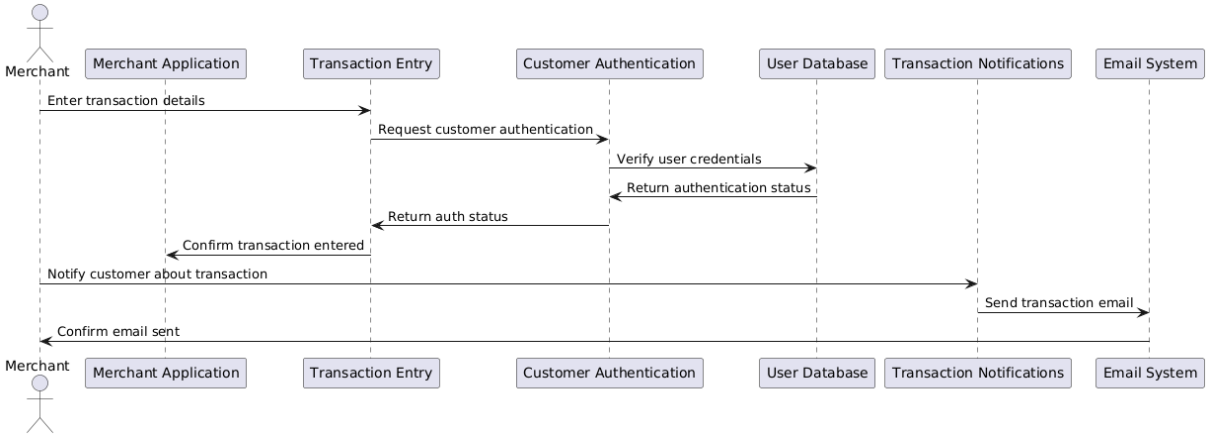

Figure 2.8 Sequence Diagram for E2E flow for Merchant application

Email Extraction, Filtering, and Database Interaction: The **Email System** provides raw transaction emails, which are extracted and filtered to store transaction data in the **Transaction Database**, and it maps transactions to users in the **User Database**. This is framed as sequence diagram at below Figure 2.9.

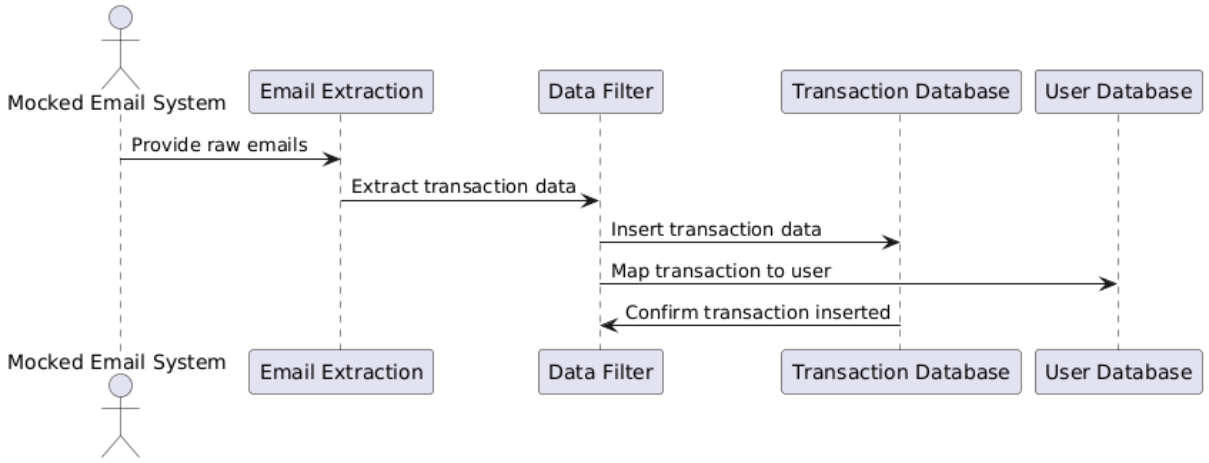

Figure 2.9 Sequence Diagram for Emal Extraction, Filtering and Database Interaction

2.2.4.2._CLASS DIAGRAMS

The diagram Figure 2.10 above is a **class diagram** for an expense tracker system, illustrating key entities like User, UserProfile, Merchant, Transaction, TransactionDatabase, and EmailSystem. It shows their attributes and relationships, such as users making transactions, merchants processing them, and the email system extracting transaction data for storage. This design emphasizes managing users, transactions, and integrating email data for financial insights.

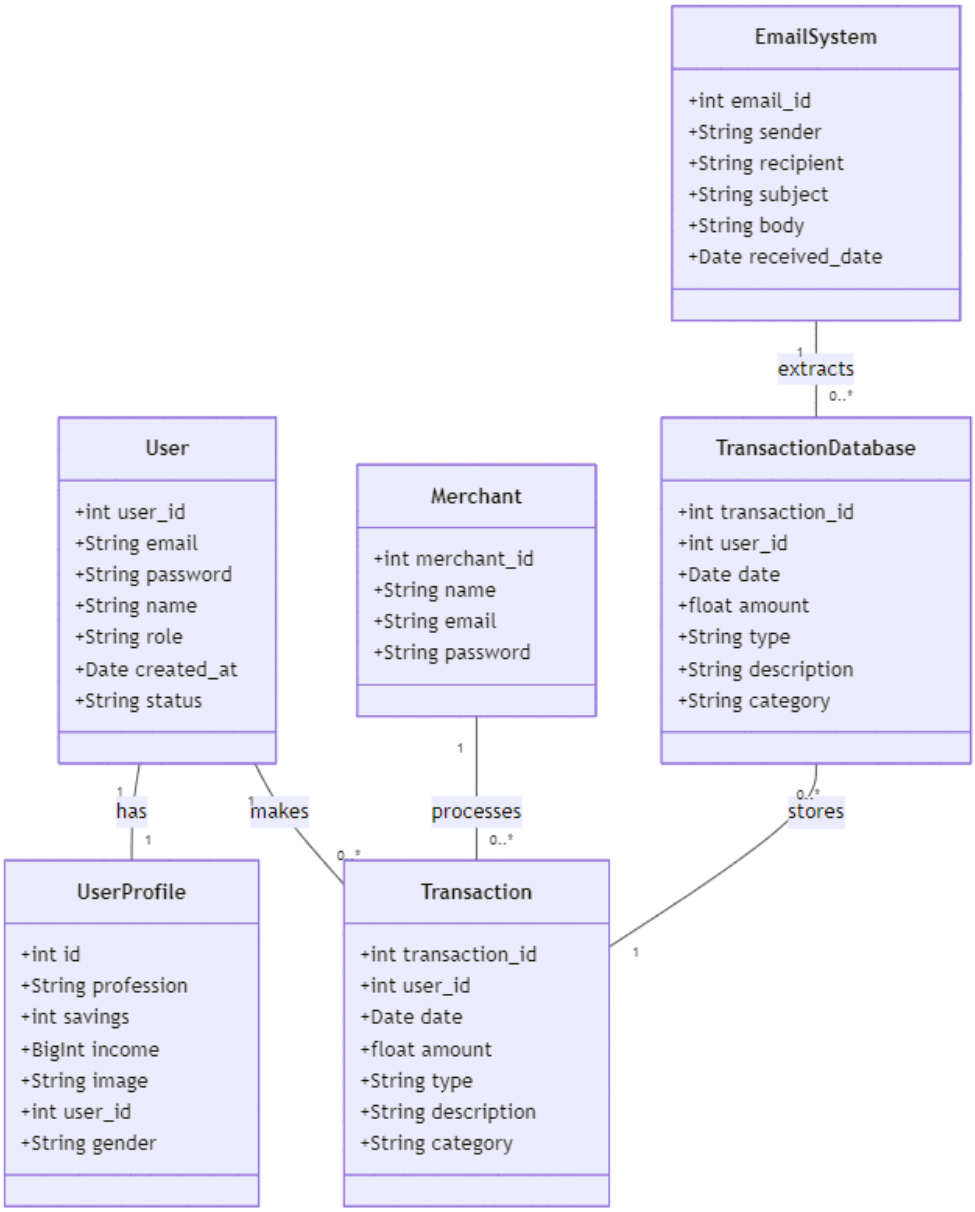

Figure 2.10 Class Diagram for developing Streamline Personal Finance (high level)

2.2.5. PERFORMANCE REQUIREMENT

The performance requirements outline the expected efficiency, speed, and responsiveness of the system. Below are the key performance requirements for this project:

Requirement	Description	Metric
Login and Authentication	System should authenticate user credentials and provide access to the dashboard.	< 2 seconds
Transaction Data Fetching	Retrieve and display transaction data (manual or email-based) on the dashboard.	< 5 seconds
Transaction Entry	Manual transaction entries should be saved and reflected in the system.	< 3 seconds
Email Synchronization	Synchronize and extract email data (mocked system).	< 5 seconds
Data Visualization	Render financial visualizations (charts, graphs) after loading data.	< 5 seconds
System Throughput	Concurrent users accessing the system or entering transactions.	100 concurrent users
Transaction Handling	Process and store transactions.	1,000 transactions/minute
Scalability	Ability to handle increasing user count and transactions.	Up to 10,000 concurrent users
Data Integrity and Accuracy	Ensure accuracy in data retrieval, storing, and displaying financial transactions.	100% accuracy
System Availability	Availability of the system during peak times (weekdays).	99.9% availability
Data Synchronization	Synchronize data between email extraction, transaction database, and UI.	< 3 seconds delay

Security	Secure data transmission (SSL/TLS), password hashing (bcrypt), and secure login protocols (e.g., OAuth).	High security standards
Resource Utilization	Optimize CPU and memory usage, especially for low-spec devices.	Efficient memory/CPU usage
Storage Optimization	Optimize transaction data storage for fast retrieval, indexing, and query optimization.	Optimized storage and indexing

Table 2.1 Performance Requirement for the project

This table should help in clearly defining and tracking the performance requirements for this project.

CHAPTER 3
SYSTEM DESIGN AND TEST PLAN

System Design refers to the architectural and functional design decisions that are made to define the structure and components of the software system. This includes high-level design (architecture) and detailed design (implementation-level decisions), addressing how the system is going to be implemented and how its components interact with each other.

The IEEE 830-1998 standard for System Design documentation outlines key areas that must be described to give a clear overview of the system's architecture and its components. Below is the detailed system design and test plan.

3.1. SYSTEM ARCHITECTURE

The purpose of framing system architecture is to describes the overall structure of the system, identifying key modules, components, and their relationships. The system architecture for this application depicted below Figure 3.1, includes major components like the **Base Expense Tracking System**, **Merchant Application**, **Email Extraction and Filtering System**, and how these interact with each other.

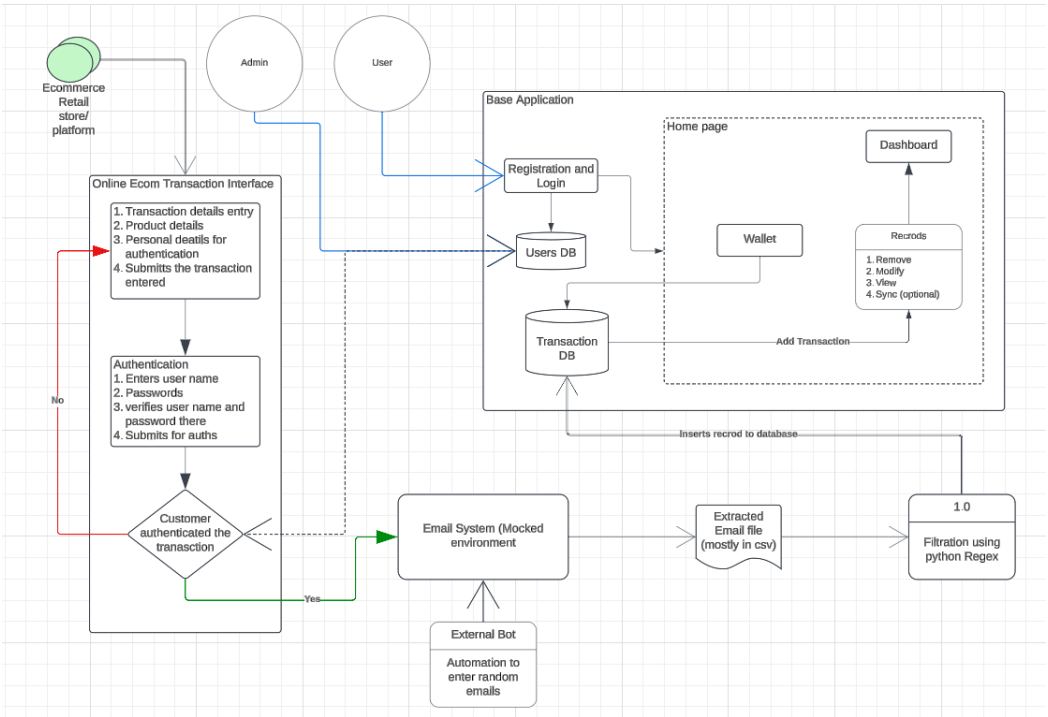

Figure 3.1. System Architecture: STREAMLINING PERSONAL FINANCIAL SYSTEMS

Let us categorize this architecture into 3 major segments. That explains and depicts in the image above,

1. **Base Application: (Expense tracking system)**

As per the below diagram, users have the direct interaction with this application which can proceed them the transaction list, visualizations via dashboard segment, and allows to enter the transactions manually. This entire application acts as a base system which steps in automated transactions.

i. Login and registration: It uses basic credential concept of registering and login to the web app

ii. Home page: After success login, user gets navigated to home page containing below sub sections.

   a. Dashboard: user interface for data visualization of transactions. It uses Pie chart for daily and weekly expenses to exhibit which kinds of spends are made by the user. Another visualization is bar Graph which emphasizes the usage of monthly budget showcasing how well we are increasing the change.

   b. Manual Entry: This is done when user wants to enter the manual transaction. The entry made here will be directly stored in Transaction Database.

   c. View Transaction: This provides the tabular view of transaction happened within user specified range.

   d. Profile: Enable customization of user profile.

2. **Online Ecom Transaction Interface**

This is the external component that doesn't depend on any system integrated within application environment. The application allows the mocked system where the merchant can enter the transaction details (if it is manual bill entry / in-store) or we are defining the payload of transaction details (if you view it as online shopping). However, in future scope, if we implement it in real-time environment, this system may be decommissioned.

Followed by entering details, it will ask for customer authentication, which it asks for User name and password of the Streamline Personal Finance that we develop, as we utilize this in testing / non-real environment.

3. **Email Extraction and data parsing systems:**

This is the vital part in the entire project that which can extracts all the email mentioned and then filters the emails based on relevant keywords such as Banker name, merchant name, Debit / Credit, Amount specified. It utilizes filtration method and Regex methods to convert the email content into tuple of relevant information (banker, merchantname, purchaseitem, debit/credit, amount) to the database for further procedure.

i) Emailing system: It is the gateway in which we use extended services for processing mocked emails which allows us to send mocked email and visualize email occurred to the account. We have end-connected with the Email Bot such that it will automate sending the email to the email id via the gateway mentioned. This is done in intention to activate the filtration process.
ii) Extraction: Using relevant methods in python, we extract all emails into CSV.
iii) Filtration: As discussed, entire filtration process and parsing data in tuple is responsible for this component.

3.2.INTERACTION DESIGN

This section covers the end-to-end workflow of application in each point of view for clear understanding on how each component mentioned in above architecture works. Hence readers are requested to get through the architecture and then proceed with this.

The below Table 3.1 summarizing the roles and steps for this project:

Role	**Step**	**Description**
User	Logging In	Access the application using email and password, then navigate to the home page.
	Adding Manual Transactions	Enter miscellaneous transactions not auto-detected by the system.
	Synchronizing Transactions	Sync with the mocked email system to retrieve and record transaction data.
	Viewing Dashboard	Access a user-friendly dashboard to visualize expenses via pie charts, graphs, and reports.
Admin	Logging In	Access the admin portal using secure credentials.

	Managing Users	Monitor user activities, view registered users, and delete accounts upon request.
	Monitoring Transactions	Audit transaction logs and ensure data accuracy across the system.
	Managing Merchants	Add/remove merchants and ensure compliance with system policies.
	Ensuring System Health	Perform backups, diagnostics, and address performance bottlenecks or vulnerabilities.
Merchant	Logging In	Log into the merchant portal using secure credentials.
	Recording Transactions	Input customer purchase details such as name, product details, and amount.
	Authenticating Customers	Verify customers by cross-referencing with the system's user database.
	Sending Notifications	Trigger email notifications to customers with transaction details (receipt, merchant name, etc.).
	Resolving Disputes	Assist customers in resolving issues related to notifications or transactions.

Table 3.1 Interaction Design Specifications

3.3.DATABASE DESIGN

The database design for the system ensures efficient data storage, retrieval, and interaction for all modules, including user management, transaction handling, and system operations. The design consists of two primary databases: **User Database** and **Transaction Database**.

Transaction Database (TxnDB)

The **Transaction Database** is designed to store all financial transactions, whether manually entered by users or synchronized from email data.

- **Structure**:
    - Contains records of transactions categorized by type (income or expense), date, amount, and description.

    - Links each transaction to the corresponding user through a **user ID**.
- **Key Attributes**:
    - transaction_id (Primary Key): Unique identifier for each transaction.
    - user_id (Foreign Key): Links the transaction to the respective user in the **User Database**.
    - date: Timestamp when the transaction occurred.
    - amount: The value of the transaction.
    - type: Defines whether the transaction is income or expense.
    - description: Additional information about the transaction (e.g., product details, merchant name).
    - category: Classification of the transaction (e.g., groceries, rent, utilities).
- **Features**:
    - Supports filtering and categorization of transactions for visualizations.
    - Ensures data accuracy and integrity through relationships with the **User Database**.

User Database (UserDB)

The **User Database** is the backbone for user management, storing essential user details and managing access to the system.

- **Structure**:
    - Contains user-specific data like login credentials, personal information, and preferences.
    - Provides ability to store profile details and pictures.
- **Key Attributes**:
    - user_id (Primary Key): Unique identifier for each user.
    - email: User's email, which is also used for synchronization with the mocked email system.
    - password: Securely hashed password for authentication.
    - name: Full name of the user.
    - role: Defines whether the user is an **individual user**, **merchant**, or **admin**.
    - created_at: Timestamp when the user registered.
    - status: Indicates if the account is active, suspended, or closed.

- **Features**:
    - Enables secure access through hashed passwords and role-based authentication.
    - Provides data linkage for transactions through the user_id attribute.

Relationship Between Transaction and User Databases

- **One-to-Many Relationship**:

Each user can have multiple transactions linked through the user_id.

- **Referential Integrity**:

Ensured by the user_id in the Transaction Database referencing the primary key in the User Database.

3.4. TEST PLAN

A **Test Plan** serves as a blueprint for the testing activities of a project, ensuring that the system functions as expected and meets the requirements. This section details the approach, scope, testing types, and responsibilities for validating the expense tracking system. the test plan provided adheres to the **IEEE 829-1998 (Standard for Software Test Documentation)**, which outlines the necessary components for a comprehensive test plan.

1. Test Scope

This test plan covers both functional and non-functional testing for the expense tracking system. The focus is on ensuring the system performs as expected, meeting the business requirements and user expectations. The testing includes validating all modules and their interactions, scalability, and user experience. The high-level scenarios defined as per the scope is mentioned in the Table 3.

2. Objectives

The main objective is to ensure the software functions as intended across all modules—user authentication, transaction management, email extraction, and merchant system. Integration between modules will be tested, and the system will be validated for performance and scalability under load. The test plan aims to provide thorough verification of both the front-end (user interface) and back-end (data processing and database interaction).

3. Types of Testing

Unit Testing

Unit testing focuses on testing the smallest units of functionality, such as individual functions, methods, or classes. In the context of this project, unit tests verify isolated components such as:

- **Transaction Entry Functionality**: Ensuring that the function for adding manual transactions works as expected.
- **Email Parsing**: Verifying that the email data extraction and filtering logic correctly identifies relevant transaction data and maps it to the user.
- **Database Interactions**: Ensuring the database updates correctly when transactions are added or modified.

Unit tests are conducted using mock data to isolate and test specific functionalities without dependency on other parts of the system.

Integration Testing

Integration testing verifies that different modules or components of the system work together as expected. This testing ensures that the interactions between components such as the email system, database, and transaction management are seamless. For instance:

- **Email to Database Integration**: Testing how the email extraction module communicates with the database to insert transaction data.
- **Merchant System and User Database**: Validating that the merchant application correctly authenticates users and updates user records in the database.
- **Transaction Synchronization**: Ensuring that the synchronization service pulls transaction data from the mocked email system and updates the database correctly.

Integration tests help to identify issues that arise when different parts of the system work together.

Smoke Testing

Smoke testing is the initial set of tests conducted on a new build to check if the critical functionalities work as expected. This testing provides early confidence that the build is

stable enough for further detailed testing. In this project, smoke testing covers the following critical functionalities:

- **Login and Registration**: Ensuring that users can log in with valid credentials.
- **Transaction Addition**: Verifying that users can add transactions through both manual entry and email synchronization.
- **Data Visualization**: Checking that the dashboard correctly displays transaction data, including pie charts for income and expenses. Smoke testing is performed after each build to confirm that the basic functionalities work before performing deeper testing.

Sanity Testing

Sanity testing focuses on validating specific changes made to the system after major updates or bug fixes. For instance, after the change from API-based direct transaction entry to email-based synchronization, sanity testing ensures that the new email integration works correctly without affecting existing functionalities. This change includes:

- **Email System Integration**: Verifying that the newly implemented email extraction system correctly maps data to the user database and that the transactions are synchronized as expected.
- **Transaction Handling**: Ensuring that manual transactions continue to work as expected alongside the newly introduced email-based synchronization.

Sanity testing is typically conducted after significant code changes to ensure that the modified system behaves as expected in the areas affected by those changes.

System Testing

System testing validates the complete and integrated system, ensuring that all components, both functional and non-functional, meet the specified requirements. This testing is comprehensive and covers end-to-end processes such as:

- **End-to-End Transaction Flow**: From entering transactions, synchronizing via email, to storing data in the transaction database, and updating the dashboard with visualizations.

- **Email Synchronization and Transaction Processing**: Ensuring that emails are correctly processed and the extracted transaction data is mapped and stored in the database.

System testing ensures that the system as a whole function correctly in all possible scenarios, simulating real-world usage.

Non-Functional Testing (NFT)

Non-functional testing measures the performance characteristics of the system, such as scalability, performance, and reliability. For this project, **scalability testing** will be the primary focus. Scalability testing ensures that the system can handle an increasing number of users and transactions without a drop in performance.

4. High Level Test Scenarios

Module	**Test Scenario**	**Type**
User Authentication	User can register with valid credentials.	Smoke, System
	Login fails with incorrect credentials.	Smoke, System
	Password reset functionality works as expected.	Sanity, System
Dashboard	Dashboard displays correct visualizations and transactions for the logged-in user.	Smoke, System
	Filters (category/date) display relevant results.	Sanity, System
Transaction Management	User can add a manual transaction.	Smoke, System
	Existing transactions can be modified or deleted.	Sanity, System
	Transactions are correctly categorized (e.g., income/expense).	System
Email Extraction	Transaction data is correctly extracted from mocked emails.	Smoke, System
	Filtered data is accurate and mapped to the correct user.	System

	Invalid email data is skipped without affecting the database.	Sanity, System
Merchant System	Merchant can enter transaction details for a user.	Smoke, System
	Authentication for the customer is validated during transaction entry.	System
	Email notification is sent after a transaction is recorded.	Sanity, System
Scalability	System supports 10,000 concurrent users with consistent performance.	Non-Functional (NFT)

Table 3.2 Test Scenarios: STREAMLINE PERSONAL FINANCE

CHAPTER 4
IMPLEMENTATION AND RESULT

4.1. DEVELOPMENT AND IMPLEMENTATION OVERVIEW

The **Financial Assistance** project is a Django-based web application designed to manage user finances. The application is divided into several key modules: user authentication, expense management, and analytics.

The project leverages Django's built-in features such as the User model for authentication and the Django ORM for database management. The core functionality includes user registration and login, transaction tracking, and data visualization.

- **User Authentication**: The users app handles user registration, login, and profile management. It uses Django's User model and includes forms to validate user inputs. This can be visualized as per Figure 4.1.
- **Expense Management**: The expenses app allows users to record and categorize financial transactions. Transactions are stored in the Transaction model, which includes fields such as amount, category, date, and user. Views are implemented to display transactions and add new ones as per Figure 4.2, Figure 4.3.
- **Analytics**: The analytics app generates visualizations of transaction data, such as pie charts and bar graphs, using libraries like matplotlib. This helps users analyse their spending patterns. The basic version is implemented as per Figure 4.5.
- **Admin Interface**: The Django admin interface is customized to allow administrators to manage user profiles, transactions, and other app data.
- **Routing**: URL routing is configured in each app's urls.py file, which integrates with the project's main urls.py to map URLs to views.

The system supports features like CSV export of transactions, secure user authentication, and responsive data visualization to help users manage their finances effectively.

4.1.1. DATABASE DESIGN

The database consists of several interrelated tables designed as per Figure 2.6 to handle user information, financial transactions, and profiles. Key design principles include normalization to eliminate redundancy and ensure data integrity.

Entity Definitions

1. **auth_user**: Stores authentication details for all system users. This Includes information like username, email, and account status. (defined in table 4.1)
2. **home_userprofile**: Extends user details with profile-specific data such as profession, savings, income, and gender. Links to the auth_user table via the user_id foreign key. (defined in table 4.2)
3. **home_addmoney_info**: Logs financial transactions, including amounts, categories, and dates. This Contains a foreign key user_id linking it to the auth_user table. (defined in table 4.3)

Schema Details

1. auth_user:

Attribute	Data Type	Constraint
id	INTEGER	Primary Key
username	VARCHAR(150)	Unique, Not Null
password	VARCHAR(128)	Not Null
email	VARCHAR(254)	
is_superuser	BOOLEAN	
is_staff	BOOLEAN	
is_active	BOOLEAN	
date_joined	DATETIME	

Table 4.1 schema details- AUTH_USER

2. home_userprofile:

Attribute	Data Type	Constraint
id	INTEGER	Primary Key
profession	VARCHAR(10)	
Savings	INTEGER	
income	BIGINT	
image	VARCHAR(100)	

user_id	INTEGER	Foreign Key (auth_user.id)
gender	VARCHAR(10)	

Table 4.2 Schema details - HOME_USERPROFILE

3. home_addmoney_info:

Attribute	**Data Type**	**Constraint**
id	INTEGER	Primary Key
Date	DATE	
Category	VARCHAR(20)	
user_id	INTEGER	Foreign Key (auth_user.id)
add_money	VARCHAR(10)	

Table 4.3 Schema details - HOME_ADDMONEY_INFO

Design Considerations

- **Normalization**: All tables are normalized to reduce data redundancy.
- **Scalability**: The schema design allows for future enhancements, such as adding new transaction types or user roles.
- **Integrity**: Relationships enforce data integrity through foreign key constraints.

4.1.2. SCREENSHOTS

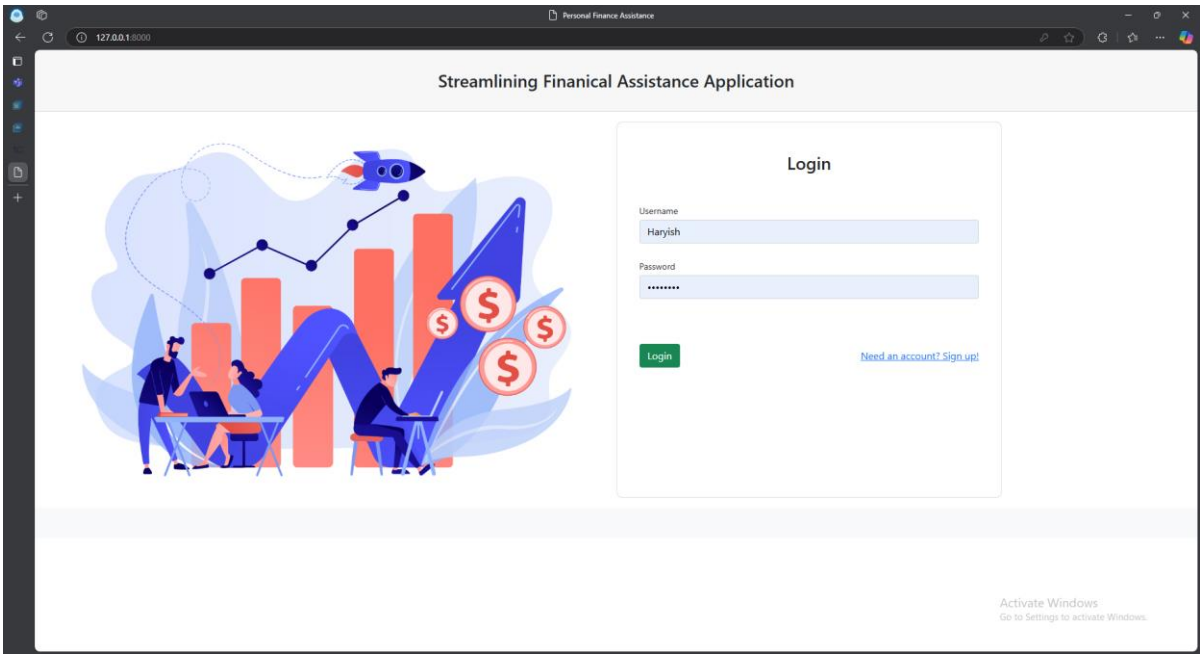

Figure 4.1 Login to Base Application

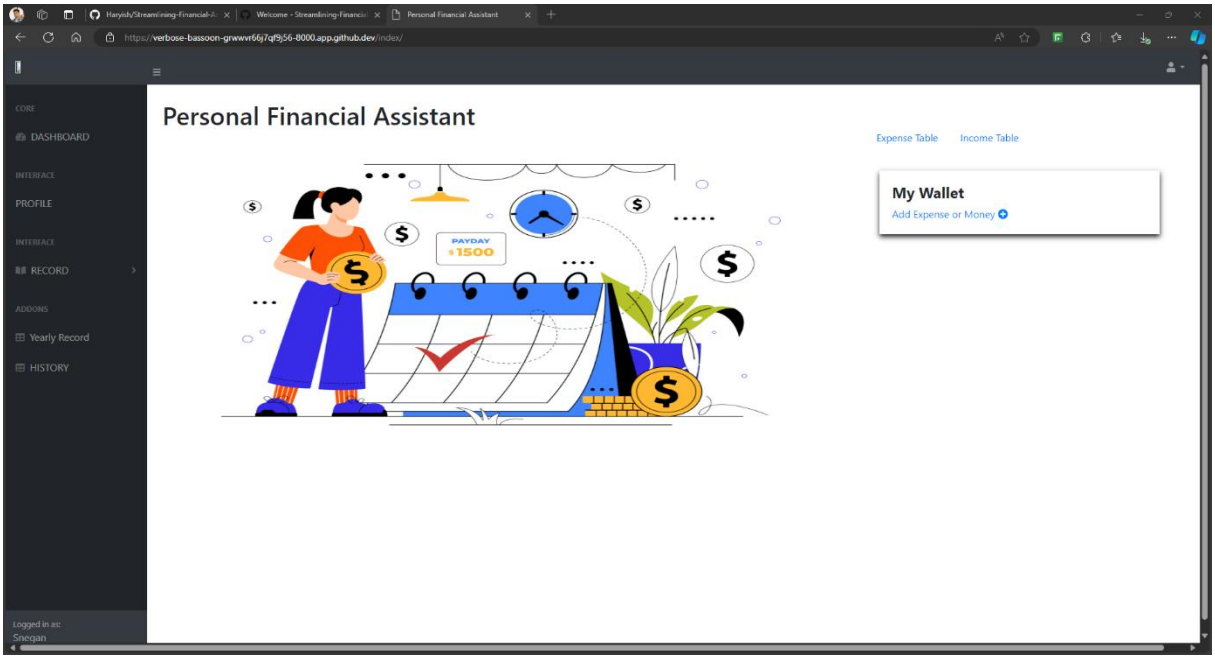

Figure 4.2 Home Page on Base Application

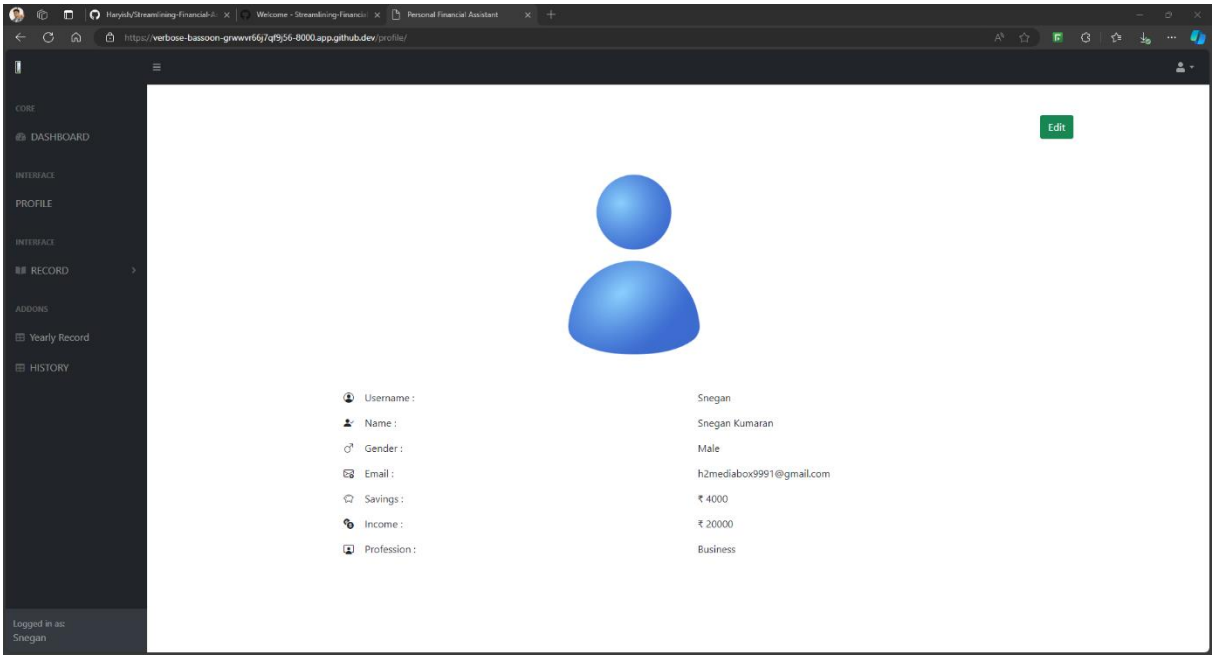

Figure 4.3 Profile to Base Application

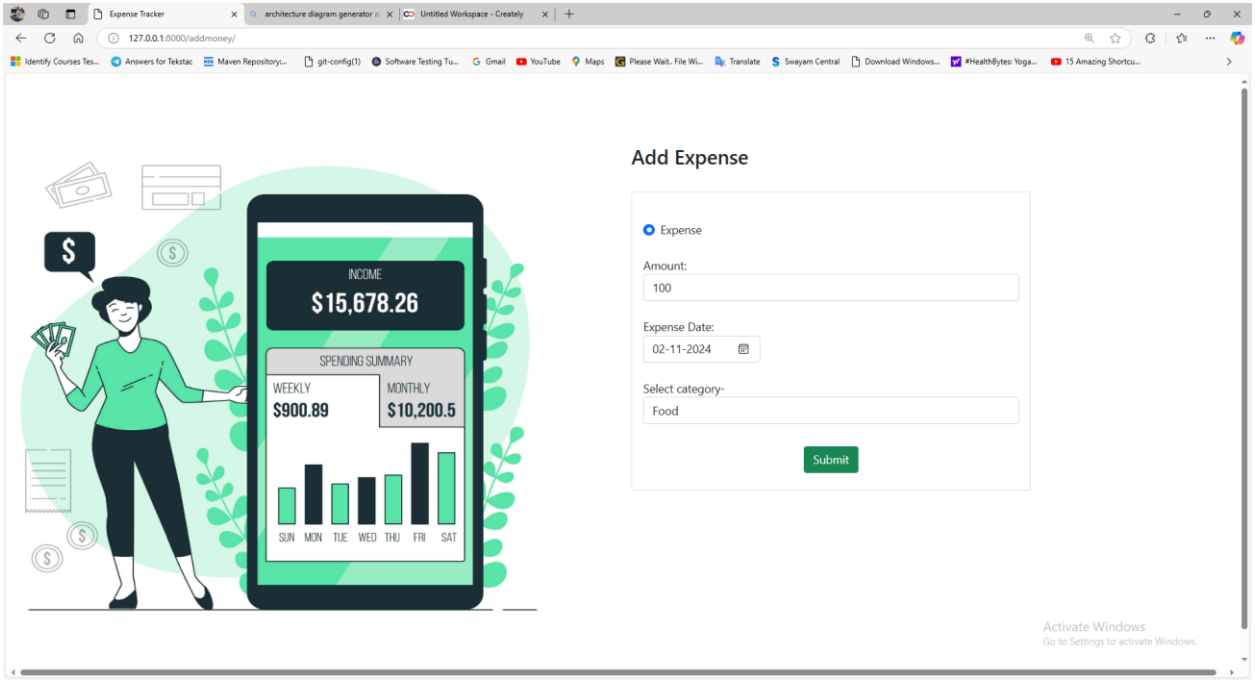

Figure 4.4 Transaction Manual Entry

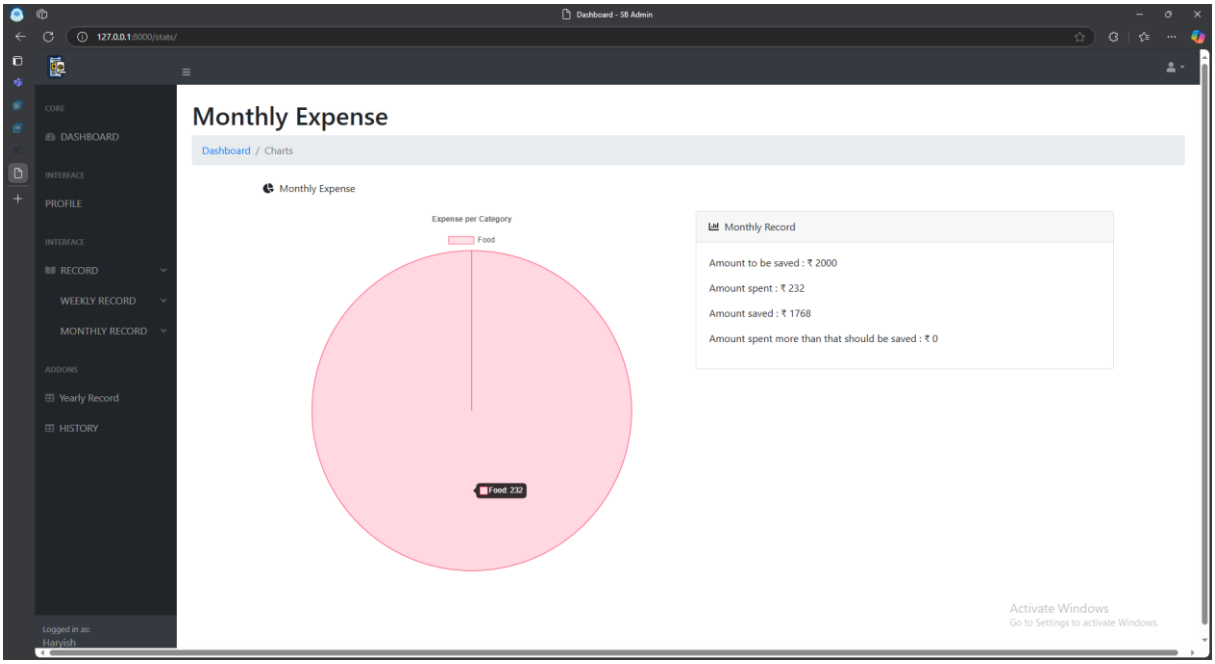

Figure 4.5 Data Visualization of Monthly Expenses

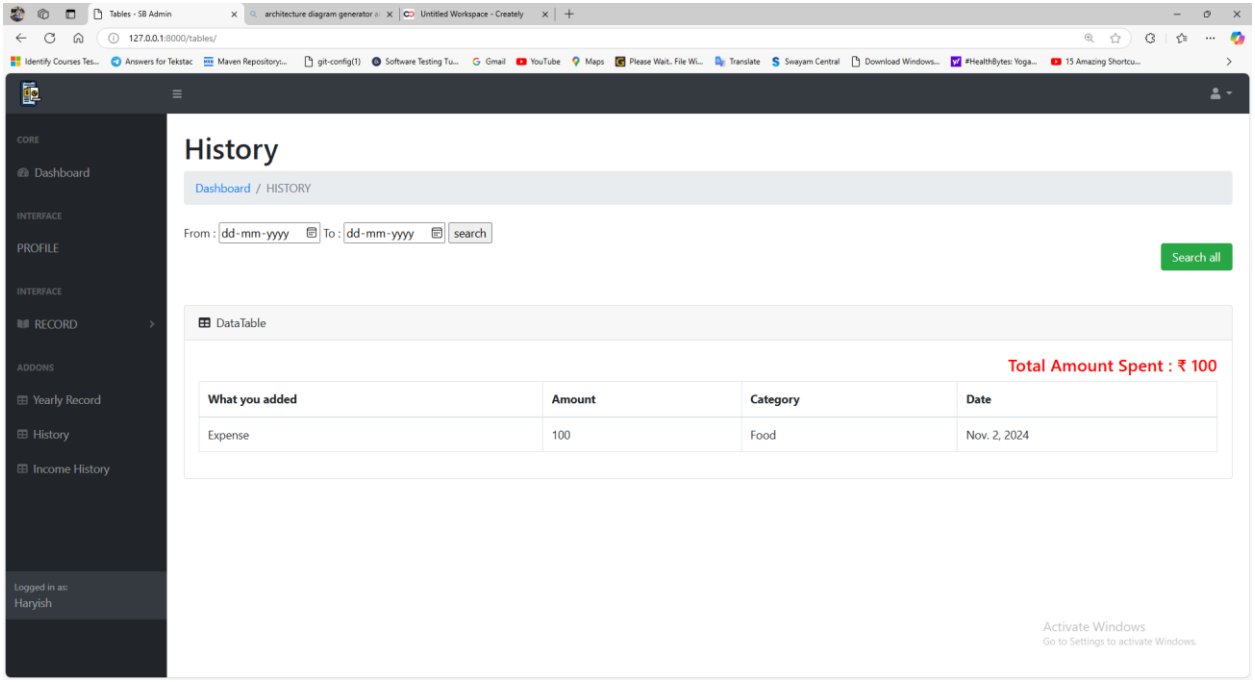

Figure 4.6. Transaction History - Existing Approach

4.2.OUTCOMES

The **Streamline Personal Finance** application has been successfully implemented, meeting all the key requirements and demonstrating the expected functionality and performance. Below are the results and observations for each major feature:

1. The system reliably authenticates users with valid credentials, granting access to their dashboard while displaying a clear error message for invalid attempts. Users can recover accounts via a secure email-based password reset link, ensuring usability and security.
2. Users can manually add transactions by specifying details like amount, date, and category, which are securely stored for review and analysis.
3. Transactions are displayed in an organized table with filters for categories and dates, enabling quick and efficient access to specific records.
4. Users can export their transaction data as CSV files for offline use, retaining accurate formatting and details for further analysis.
5. The system retrieves transaction-related emails using a mocked email service, ensuring seamless integration with external sources.

6. Transaction details such as amounts, merchants, and dates are accurately parsed from email content using regex, meeting the system’s data requirements.
7. Email-extracted transactions are seamlessly synced with the database, providing users with a unified and up-to-date transaction history.
8. Dynamic bar and pie charts visually represent categorized spending patterns, updating in real-time to enhance user insights.
9. Time-based analytics track spending trends over various periods, helping users identify habits, set goals, and make informed financial decisions.
10. Admins can deactivate or delete user accounts securely, ensuring compliance with data privacy regulations and user requests.
11. Usage logs and system metrics allow admins to monitor activity, track transactions, and address issues proactively for smooth operations.
12. The system processes transactions in real-time and renders visualizations for large datasets efficiently, maintaining high performance and user satisfaction.

This expanded section provides a detailed breakdown of how each feature is implemented and how it meets the project requirements. It also highlights the system's performance under realistic usage scenarios.

CHAPTER 5

IMPLEMENTATION AND RESULT

The successful completion of this project marks a significant step in delivering a software solution that addresses the practical needs of users in tracking and managing their financial transactions. The **Streamlining Personal Finance Assistant** has been designed and developed to simplify personal and professional financial management for individuals, freelancers, and small businesses. By integrating features like manual transaction entry, email-based transaction extraction, and robust visualization tools, the system offers a user-friendly platform for effective financial oversight.

5.1. RESULTS

The project achieved its goals, addressing both functional and non-functional requirements while overcoming significant challenges:

Accomplishments:

- The system allows manual transaction entry, email synchronization, and detailed financial summaries. Administrators monitor users, and merchants authenticate and manage transaction notifications.
- It supports 10,000 concurrent users and 1,000 transactions per minute, ensuring 99.9% availability through rigorous testing.
- Secure financial data handling is ensured with bcrypt and SSL/TLS protocols.
- A user-friendly dashboard enables users to track finances efficiently, while merchants can easily input and notify transactions.

Challenges Addressed

The project effectively overcame the following challenges:

- Transitioning from direct API integration to a mocked email gateway for extracting transaction data while maintaining security and feasibility.

- Designing a reliable data filtration mechanism using Python's REGEX module to parse and store transaction data accurately.

System Value

This system bridges the gap between manual financial management and automation. Freelancers and individual users benefit from an effective tool for expense tracking and financial trend analysis, empowering better decision-making. Merchants gain simplified customer notification processes, while administrators achieve enhanced user management capabilities.

Limitations

Despite its achievements, the system has certain constraints:

- It relies on mocked environments for email and e-commerce interactions, limiting its current use in real-world API integrations.
- Transaction categorization remains manual, though plans for future enhancements include machine learning-based automation.

5.2. SUMMARY

The **Streamlining Personal Finance Assistant** serves as a robust prototype that combines essential financial tracking features with the potential for future growth. It is a scalable, secure, and user-friendly solution that addresses the financial management needs of its target audience. With further development, it holds the promise of becoming a comprehensive tool that integrates seamlessly into users' financial ecosystems.

The project reflects a harmonious blend of practical implementation, rigorous testing, and thoughtful design, showcasing the potential for software solutions to simplify and enhance everyday financial processes.

5.3. FUTURE WORK

While the Streamlining Personal Finance Assistant addresses essential financial tracking needs, future enhancements will focus on transforming it into a fully intelligent and automated solution. Key advancements include:

1. **Real-Time API Integration**

Replacing mocked environments with live integrations, such as Gmail/Outlook for email synchronization, Plaid for banking data, and PayPal/Amazon for e-commerce transactions, will enhance accuracy and efficiency.

2. **Automated Categorization**

Incorporating machine learning models and NLP techniques will automate tagging of transactions, improving scalability and accuracy by analysing patterns and descriptions.

3. **Predictive Analytics**

Advanced analytics, including expense forecasting, anomaly detection, and personalized financial recommendations, will provide users actionable insights to optimize financial decisions.

REFERENCES

[1] P. Thanapal and M. Y. Patel, "Income and Expense Tracker," *Indian Journal of Science and Technology*, vol. 8, no. S2, pp. 118–122, Jan. 2015.

[2] M. Liqian, T. Siyu, and J. B. Michael, "Customized Multi-person Tracker," *Springer International Publishing Asian Conference on Computer Vision*, Perth, June 2019.

[3] R. Kukade, P. Bhatele, D. Mahajan, et al., "TrackEZ Expense Tracker," in *2023 4th International Conference for Emerging Technology (INCET)*, Belgaum, India, pp. 1–5, 2023, doi: 10.1109/INCET57972.2023.10170735.

[4] J. L. Yeo, P. S. JosephNg, K. A. Alezabi, et al., "Time Scheduling and Finance Management: University Student Survival Kit," in *2020 IEEE Student Conference on Research and Development (SCOReD)*, Batu Pahat, Malaysia, pp. 1–6, 2020, doi: 10.1109/SCOReD50371.2020.9250969.

[5] P. Bhatele, D. Mahajan, and N. Mahajan, "TrackEZ Expense Tracker," in *2023 4th International Conference for Emerging Technology (INCET)*, Belgaum, India, pp. 1–5, 2023, doi: 10.1109/INCET57972.2023.10170735.

[6] A. Tamizhselvi, M. Anbu, and K. R. Radhakrishnan, "Financial and Individual Future Expense Prediction Based on Frequent Patterns Using Micro Services," in *2022 Third International Conference on Intelligent Computing Instrumentation and Control Technologies (ICICICT)*, Kannur, India, pp. 532–536, 2022, doi: 10.1109/ICICICT54557.2022.9917982.

[7] B. D. Bhujang, P. V. Wendole, A. P. Thakare, et al., "Expense Tracker and Budget Planner," *International Journal of Advanced Research in Computer Science*, vol. 9, no. 2, pp. 120–130, 2020.

[8] IEEE Computer Society, *IEEE Recommended Practice for Software Requirements Specifications (IEEE 830-1998)*, 1998.

[9] Python Software Foundation, *Django Documentation (Version 4.2)*, available at: https://docs.djangoproject.com/.

[10] SQLite Global Development Group, *SQLite Documentation (Version 15)*, available at: https://www.sqlite.org/docs/.

[11] Google Developers, *Gmail API Documentation*, available at: https://developers.google.com/gmail/api.

[12] PlantUML Documentation, available at: https://plantuml.com/.

[13] Python.org, *Python Standard Library Documentation*, available at: https://docs.python.org/3/library/.

[14] W3C, *Web Content Accessibility Guidelines (WCAG)*, available at: https://www.w3.org/WAI/standards-guidelines/wcag/.

[15] TutorialsPoint, *SQLite Tutorial*, available at: https://www.tutorialspoint.com/sqlite/.

[16] PyTest Documentation, available at: https://docs.pytest.org/.

[17] R. Shetty, *Selenium Webdriver with PYTHON from Scratch + Frameworks*, Udemy Course, available at: https://www.udemy.com/course/selenium-webdriver-with-python-from-scratch-frameworks/.

[18] Mermaid Documentation, available at: https://mermaid-js.github.io/mermaid/.

Contents

Do not print