

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

By

HARYISH E

2232MCA0058 – 67222200076

A PROJECT REPORT

SUBMITTER 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

SUBMITTER 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

Name:

Designation:

Address:

External Examiner

Name:

Designation:

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 her 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, in particular, 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 CONTENTS

Chapter No.	Chapter Name	Page No.
	ABSTRACT	i
	TABLE OF CONTENTS	iii
	LIST OF TABLES	v
	LIST OF FIGURES	vi
1	INTRODUCTION	1
	1.1 OVERVIEW	1
	1.2 LITERATURE SURVEY	2
	1.3 EXISTING AND PROPOSED SYSTEM	5
	1.4 OBJECTIVE AND SCOPE OF THE PROJECT	7
	1.4.1 OBJECTIVE	7
	1.4.2 SCOPE	8
	1.5 ORGANIZATION OF THE REPORT	9
2	REQUIREMENT SPECIFICATION	11
	2.1 OVERVIEW	12
	2.1.1 PRODUCT PERSPECTIVE	13
	2.1.2 PRODUCT FUNCTION	16
	2.1.3 USER CHARACTERISTICS	18
	2.1.4 OPERATING ENVIRONMENT	19
	2.1.5 CONSTRAINTS	20
	2.2 SPECIFIC REQUIREMENT	20
	2.2.1 EXTERNAL INTERFACES	21
	2.2.2 USE CASES	22
	2.2.3 DATA FLOW AND ER DIAGRAMS	26
	2.2.4 UML DIAGRAMS	27
	2.2.5 PERFORMANCE REQUIREMENT	30

3	SYSTEM DESIGN AND TEST PLAN	32
	3.1 SYSTEM ARCHITECTURE	32
	3.2 INTERACTION DESIGN	34
	3.3 DATABASE DESIGN	35
	3.4 TEST PLAN	37
4	IMPLEMENTATION AND RESULT	43
	4.1 DEVELOPMENT AND IMPLEMENTATION OVERVIEW	43
	4.1.1 DATABASE DESIGN	44
	4.1.2 SCREENSHOTS	45
	4.2 OUTCOMES	48
5	CONCLUSION	50
	5.1 RESULTS	50
	5.2 SUMMARY	51
	5.3 FUTURE WORK	51
	REFERENCES	53
A	APPENDICES	55
	A.1 DETAILS OF PROPOSED ENHANCEMENTS	55
	A.2 SCOPE FOR FUTURE WORK	56

LIST OF TABLES

Table no.	Table name	Page no.
Table 1	Use cases for centric financial assistance portal	23
Table 2	Use cases for external systems to financial assistance portal	24
Table 3	Performance requirement for the project	31
Table 4	Interaction design specifications	35
Table 5	Test scenarios for all modules	42
Table 6	Schema details- auth_user	44
Table 7	Schema details - home_userprofile	45

LIST OF FIGURES

Figure No.	Title	Page no.
Figure 1	Use case diagram for base expense tracking application	24
Figure 2	Use case for email integration system	24
Figure 3	Use case diagram for merchant application	25
Figure 4	Use case diagram for overall system interacting with each other	25
Figure 5	Er diagram for commutation to base application	26
Figure 6	Data flow diagram for entire application	27
Figure 7	Sequence diagram for e2e flow of base expense tracker	28
Figure 8	Sequence diagram for e2e flow for merchant application	28
Figure 9	Sequence diagram for email extraction, filtering and database interaction	29
Figure 10	Class diagram for entire software (high level - tentative)	29
Figure 11	System architecture of streamlining personal financial systems	32
Figure 12	Login to base application	45
Figure 13	Home page on base application	46
Figure 14	Profile to base application	46
Figure 15	Transaction manual entry	47
Figure 16	Data visualization of monthly expenses	47
Figure 17	Transaction history - existing approach	48

1. INTRODUCTION

Efficient financial management is vital in personal and professional domains, yet traditional methods, such as manual data entry on paper or spreadsheets, remain time-consuming, error-prone, and insecure. Freelancers and small business owners face additional challenges, including the segregation of personal and professional expenses, hindering effective financial analysis. Current systems improve user interfaces for manual entry but often neglect automation, accuracy, and data consistency.

This project addresses these gaps by developing a digital Streamline Personal Finance using the Django framework. It automates expense tracking, ensuring secure data management and providing an intuitive interface for seamless use. The system supports essential features such as user registration, secure storage, and income-expense categorization. Users can track financial data on various timelines, with insights visualized using charts for better decision-making. Designed for scalability, the system supports freelancers and small business owners by enabling transaction classification into personal and professional categories.

Additionally, the project explores automation and integration possibilities with external platforms, such as e-commerce systems, to reduce manual effort and operational costs. By replacing outdated tracking methods, this solution offers a modern, user-centric platform for efficient, secure, and comprehensive financial management.

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 pie charts, enabling quick and accurate 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.

Leveraging Django's lightweight and scalable 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 provides an in-depth analysis of existing work and systems related to expense tracking, financial management, and automation, highlighting their limitations and identifying opportunities for improvement. By reviewing various papers, tools, and methodologies, this survey establishes the foundation for the proposed Streamline Personal Finance system, which aims to enhance user experience, automation, and accuracy. The following presents the detailed findings from key studies and systems in the field.

i. “Income and Expense Tracker”

Author: P. Thanapal and Mohammed Yasmeen Patel

Published in: Indian Journal of Science and Technology, 2015

Description:

This study focuses on the design and development of an income and Streamline Personal Finance aimed at helping individuals monitor their personal finances. The authors proposed a manual entry-based system where users input their income and expenses through a simple interface. It provided a basic categorization of expenses, such as groceries, transportation, etc. The system was useful for individuals who wanted to track their expenses but lacked advanced features like automation or data categorization beyond basic categories.

- **Limitations:** The major limitation of this study was its reliance on manual data entry, which is time-consuming and prone to errors. There was no advanced data analysis or automation.
- **Relevance to the Project:** This work laid the foundation for the idea of expense tracking but falls short in areas like automation, transaction categorization, and data security. The proposed system will enhance this by automating the categorization of expenses and improving security.

ii. “Customized Multi-person Tracker”

Author: Ma Liqian, Tang Siyu, and J. Black Michael

Published in: Springer International Publishing, 2019

Description:

This paper explored a multi-person Streamline Personal Finance designed for shared expenses among multiple individuals, such as group projects or family budgets. The system categorized expenses based on individuals and allowed for easy tracking and sharing of financial data. The system allowed multiple users to input their expenses, which were then split and visualized for better tracking.

- **Limitations:** The system's focus on multi-user interaction did not address the challenges of tracking professional expenses for freelancers, which is a key focus of the proposed project.
- **Relevance to the Project:** While the study focused on multi-user scenarios, it did not address the needs of freelancers or small business owners who need to distinguish between personal and professional expenses. The proposed system can build on this by providing features to classify transactions as either personal or professional.

iii. TrackEZ Expense Tracker

Authors: P. Bhatele, D. Mahajan, B. Mahajan, D. Mahajan, N. Mahajan, and P. Mahajan

Description:

The aim was to create an automated expense tracking system that categorizes and organizes income and expenses. TrackEZ uses a combination of manual entry and automated classification to track and visualize expenses. The system categorizes expenses and generates reports on spending habits. The system effectively organizes expenses but still requires some manual input for categorization, limiting its efficiency. It also generates simple visual reports, such as pie charts, to assist users in understanding their financial habits.

- **Limitations:** While automation is part of the system, it does not integrate with external platforms like banks or e-commerce sites, limiting the scope of transaction automation. The system still relies on manual categorization, which could be streamlined.
- **Relevance to the Project:** The proposed system will automate the categorization of expenses more effectively and integrate with external platforms, addressing the limitations of TrackEZ, where automation is partial and limited to data visualization.

iv. **Streamline Personal Finance and Budget Planner**

Authors: Bhujang, B. D., Wendole, P. V., Thakare, A. P., Ghodele, P. C., & Khan, S. W.

Description:

The purpose of this paper was to develop an Streamline Personal Finance and budget planner that helps individuals track and categorize their income and expenses. The system allows users to manually input their transactions and categorize them. The tracker helps users stay within a budget by comparing their actual expenses to their set budget. The system successfully helped users maintain their budget by providing visual insights and categorizing their expenses. However, the reliance on manual entry remains a major drawback.

- **Limitations:** The major limitation is the lack of automation, which requires the user to input all transactions manually. The system does not integrate with banking or e-commerce platforms, limiting its ability to automate transaction entries.
- **Relevance to the Project:** This work is relevant because it focuses on budgeting and expense categorization but lacks automation. The proposed system will address this

limitation by automating transaction entries and categorizing expenses, thus enhancing user experience.

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:** The system automatically categorizes transactions as personal or professional, minimizing manual effort.
2. **Integration:** Direct integration with e-commerce platforms and banking systems to streamline transaction entries and enhance accuracy.
3. **Secure Data Management:** A robust authentication system ensures secure storage and retrieval of user data.
4. **Advanced Visualization:** Graphical representations, such as pie charts and trend analysis, provide insights into income and expenses on weekly, monthly, and yearly bases.
5. **Scalability:** Designed with scalability in mind, the system supports future enhancements, such as predictive analytics and machine learning-based insights.
6. **User-Centric Design:** A simple and intuitive interface tailored to the needs of individuals and freelancers, enabling easy tracking and segregation of income and expenses.

Advantages of the Proposed System:

1. **Time Efficiency:** Automation reduces the need for manual data entry, saving users significant time.
2. **Enhanced Accuracy:** By minimizing human intervention, the system ensures higher data consistency and accuracy.
3. **Customization for Freelancers:** The ability to segregate personal and professional finances addresses a critical need for freelancers and small business owners.
4. **Seamless Integration:** Real-time transaction integration with external platforms eliminates redundancies and improves user experience.
5. **Future-Ready:** Incorporates the potential for advanced analytics, making it adaptable for evolving financial tracking needs.

1.4. OBJECTIVE AND SCOPE OF THE PROJECT

1.4.1. OBJECTIVE

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.

1.4.2. SCOPE

The **scope** of this project encompasses the development of a comprehensive **expense management system** for individuals and freelancers, addressing the limitations of traditional and existing solutions.

1. Core Features:

- **User authentication** through a **registration** and **login system**.
- **Tracking** of **income** and **expenses** with secure **data storage**.
- **Visualization** of financial data through dynamic **charts** for **weekly, monthly, and yearly trends**.

2. User Segmentation:

- Aimed primarily at **individuals** accessing the system directly through the **web app**.
- **Freelancers (treated same as individuals)** who need to track **income** and **expenses** related to their **clients** and **business activities**.

3. Ease of Use:

- A **lightweight solution** using the **Django framework**, ensuring **maintainability** and **scalability**.
- Simplified **entry** and **retrieval processes** for **financial data**.

4. Integration with E-commerce Platforms:

- **Automate transaction entries** by integrating with **e-commerce** and **banking platforms**.

5. Scalability:

- Enable the system to adapt to larger **datasets** and **user**

Future scope includes:

1. Automated Categorization:

- Develop **machine learning models** to **classify transactions** as **personal** or **professional**, reducing manual intervention.
- **bases**, making it suitable for **small businesses** and larger organizations.

2. Platform Expansion:

Extend support for **mobile applications** and **APIs** for third-party integrations.

3. Enhanced Financial Analytics:

- Provide insights into **gains, losses, and financial trends** to support **business decision-making**.

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 provides a comprehensive view of the functional and non-functional requirements of the expense tracker. It explains what the system is designed to achieve, performance expectations, and operating constraints, following the IEEE 830-1998 SRS standards for precise documentation.

Chapter 2: Specific Requirement

This section delves deeper into the analysis and technical requirements. It includes diagrams such as UML, DFD, and ER, as well as details about external interfaces, logical databases,

and performance attributes. These components adhere to IEEE 830-1998 guidelines, ensuring a robust foundation for system design and implementation.

Chapter 3: Design and Test Plan

This chapter details the system's architectural design and testing strategies. It discusses database schemas, user interface layouts, and backend structures. The testing plan encompasses unit testing, integration testing, and user acceptance testing, ensuring the system meets quality standards through structured evaluation and tools.

Chapter 4: Implementation, QA, and Results

This section outlines the development of system modules, including user registration, expense entry, and data visualization, with Django as the core framework. It also covers the quality assurance processes, emphasizing debugging, security, and performance optimization. The results evaluate system functionality, user feedback, and performance, confirming its alignment with project goals.

Chapter 5: Conclusion

The report concludes by summarizing the project's accomplishments and value. It highlights how the system met its objectives by bridging the gap between manual financial tracking and automation. While addressing challenges like email-based synchronization and transaction filtering, the system provides a secure, scalable, and user-centric solution. It also identifies future enhancements, such as live API integration and machine learning-based categorization, to improve functionality further.

2. REQUIREMENT SPECIFICATION

The “**Streamlining Personal Finance**” system incorporates an innovative feature for automating transaction tracking by **reading emails** and **extracting transaction details**. The system interacts with the user’s email account, searching for relevant transaction-related emails, such as receipts, purchase confirmations, or bank statements. Once the emails are retrieved, the system uses predefined templates and natural language processing techniques to **identify key information** such as transaction amounts, dates, merchant names, and transaction categories. This data is then **extracted** and automatically **entered into the transaction database**.

By doing so, the system minimizes the need for manual entry, reducing errors and ensuring that financial records are consistently updated. The transaction data is stored in a secure, centralized database, allowing users to view, manage, and visualize their financial information at any time. The **visualization** aspect enables users to see their financial data through graphical representations like **pie charts**, providing insights into spending patterns and budget allocations. This integration not only streamlines the tracking process but also enhances the user experience by providing real-time, automated updates to their financial records.

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 the common issues faced by individuals and freelancers in managing their finances manually. By connecting to the user's email account, the system automatically extracts transaction details from relevant emails, such as receipts and purchase confirmations. This eliminates the need for manual entry, ensuring that financial records are consistently up to date and accurate.

Once the transactions are extracted, they are stored in a secure database, where users can view and manage them. The system also offers visualization tools, like pie charts, to help users understand their spending patterns and budget allocations. This feature provides insightful, real-time updates on financial health, making it easier to track income, expenses, and overall financial performance.

In terms of user interaction, the system is designed with a simple, intuitive interface that allows users to register, log in, and manage their data with ease. It also offers secure data storage, ensuring that all financial information is protected against unauthorized access.

This software aims to offer a seamless, automated solution for financial tracking, providing users with a clear, comprehensive, and visual overview of their finances, thereby enhancing both 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.

Why Not Email extract via GmailAPI possible to implement in this project?

- While it's feasible to use the Gmail API for real email extraction, implementing a **mocked email system** is a more practical choice for this project, as it aligns with the goal of simulating external services.
- This approach avoids the complexities of OAuth and live API integration, focusing instead on demonstrating the functionality of email parsing and transaction data extraction in a controlled environment.

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 Billing 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 provides the following key functionalities to support users in managing their expenses and transactions efficiently:

1. User Access and Authentication:

- Enable users (individuals or freelancers) to register, log in securely, and access their personalized dashboard.

2. Transaction Management:

- Allow manual entry of transactions, especially for expenses not captured through email synchronization.
- Synchronize and extract transaction details automatically from the user's email to ensure comprehensive expense tracking.
- Maintain a unified list of all transactions (manual and auto-synced) for easy review and analysis.

3. Visualization and Reporting:

- Provide data visualizations, such as pie charts and summaries, to give users insights into their financial activity.
- Support filtering and categorization for transactions based on parameters like date, type, and source.

4. Merchant Integration:

- Enable merchants to input purchase and billing details directly into the system, associating them with user accounts.
- Authenticate transactions by verifying the user's credentials against their existing records.

5. Email-Based Transaction Extraction:

- Extract transaction-related information from user emails using email provider APIs (mocked in the current system).
- Filter and parse the extracted data to identify and store relevant transaction details in the database, mapped to the appropriate user account.

6. Administrative Controls:

- Allow administrators to monitor and manage user accounts within the system.
- Provide the ability to 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:**

- **Profile:** Non-technical users managing personal and professional finances.
- **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:**

- **Profile:** Retailers or service providers interacting with customers for purchases or transactions.
- **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:**

- **Profile:** Technical users responsible for system oversight and maintenance.
- **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):**

- **Profile:** Technical professionals building or testing the system in a controlled environment.
- **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.

- **Mocked External Services:** Mock email and transaction services running on local servers for controlled testing.

3. **Third-Party Integration (Mocked):**

- Email systems, banking APIs, and e-commerce platforms are simulated using mocked environments to demonstrate core functionalities without interacting with live APIs.

2.1.5. CONSTRAINTS

1. **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.

2. **Technological Constraints:**

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

3. **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:

- **User Interface (UI):**
 - A web-based, responsive UI designed for users (individuals, freelancers, merchants, and admin) to interact with the system. Features include a dashboard, manual transaction entry form, data visualization, and secure login/logout functionalities.
- **System Interfaces:**
 - **Mock Email Integration:** A simulated system fetches emails containing transaction details using a local server environment, mimicking real-world email interaction.
 - **Data Filtration Interface:** A module to parse emails, extract transaction details using Python's REGEX, and insert them into the transaction database.
 - **Mock E-commerce Integration:** A system for manually entering transaction details and mapping them to users for testing purposes.
- **Hardware Interface:**
 - Requires standard computing devices like desktops, laptops, or mobile devices 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.

2.2.1. USE CASES

1. Base Expense Tracking Application

Actors: User, Admin

Use Cases:

Use Case Name	Use Case Description	Precondition	Postcondition
User Registration	Allows a new user to sign up by providing necessary details like email, username, and password.	User has not registered previously.	User is successfully registered and can log in.
User Login	Enables an existing user to log in using their username/email and password.	User has already registered and has valid credentials.	User is authenticated and redirected to the dashboard.
Add Transaction	Enables a user to add a new financial transaction, including details like amount, category, and date.	User is logged in and has access to the transaction management page.	New transaction is added and visible in the transaction list.
View Transactions	Displays a list of all the user's financial transactions, including date, category, and amount.	User is logged in.	User can view their transaction history.
Export Transactions	Allows the user to export their transaction data to a CSV file for offline analysis.	User is logged in and has at least one recorded transaction.	CSV file is generated with transaction data.
Generate Analytics	Generates visual reports (e.g., pie charts, bar graphs) to analyze spending patterns and transaction categories.	User has transactions recorded in the system.	Visualizations are rendered and displayed to the user.
Reset Password	Allows a user to reset their password if they forget it.	User has a registered email address.	User receives a reset link and can change their password.
Admin Manage Users	Allows an admin to manage user accounts, including	User is an admin with login credentials.	Admin can view and manage user accounts.

	viewing, editing, and deactivating accounts.		
Admin Manage Transactions	Allows an admin to view and manage user transactions, including editing or deleting records.	User is an admin with login credentials.	Admin can modify transaction data for users.

Table 1 Use cases for Centric Financial Assistance portal

2. External Systems:

Use Case Name	Use Case Description	Precondition	Postcondition
Enter Transaction Details	Merchant enters transaction details, including product and customer details.	Merchant is authenticated in the system.	Transaction details are submitted for processing.
Authenticate Customer	Merchant verifies the customer's identity through credentials provided during the transaction.	Customer credentials are valid and entered correctly.	Authentication is successful, and the transaction is validated.
Send Email Notification	After a successful transaction, the system sends an email notification to the customer with transaction details.	Email ID of the customer must be valid.	Email is delivered to the customer.
Extract Emails	The system fetches emails from the user's mocked email account.	User's email ID must be connected to the system.	Relevant emails are retrieved for processing.
Filter Transactional Data	Filters and extracts relevant data (e.g., receipts, invoices) from fetched emails using Python REGEX.	Emails must contain identifiable transactional data.	Transaction data is processed and extracted.
Insert Transactions into Database	Extracted data is stored in the transaction database for recording and visualization.	Filtered data must be valid and mapped to the correct user.	Transaction details are added to the database.
Synchronize Transaction Data	Users sync transactional data from their email or other sources with the database.	Email extraction and filtering are completed.	User's dashboard and transaction records are updated.

Monitor System Usage	Admin oversees system performance, user activity, and error logs.	Admin credentials are authenticated.	System usage insights are available to the admin.
Delete User Accounts	Admin disables or deletes user accounts upon request.	User must request account deletion.	User's data is securely removed from the system.

Table 2 Use Cases for External Systems to Financial Assistance Portal

2.2.1.1. USE CASE DIAGRAM

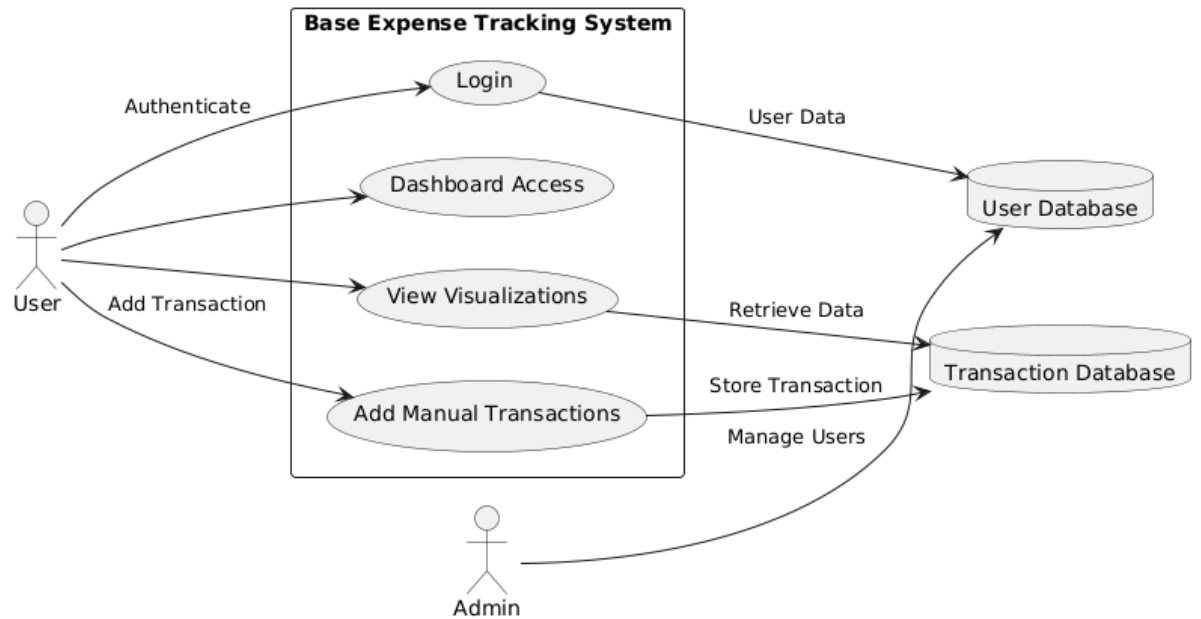


Figure 1 Use Case Diagram for Base Expense Tracking Application

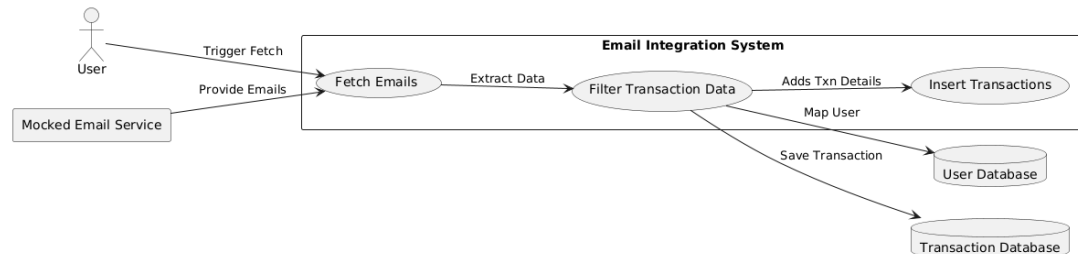


Figure 2 Use case for Email Integration System

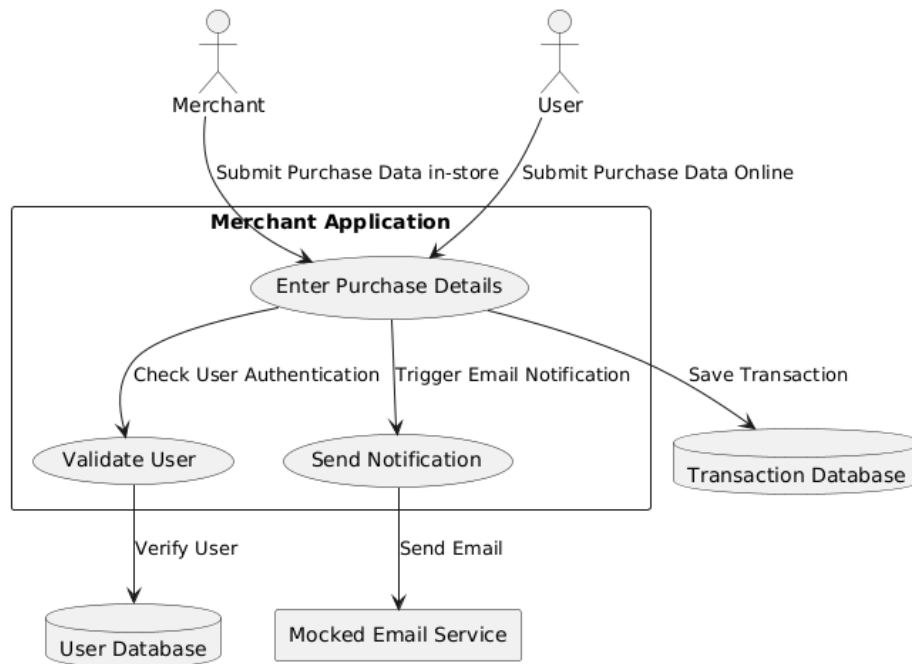


Figure 3 Use Case Diagram for Merchant Application

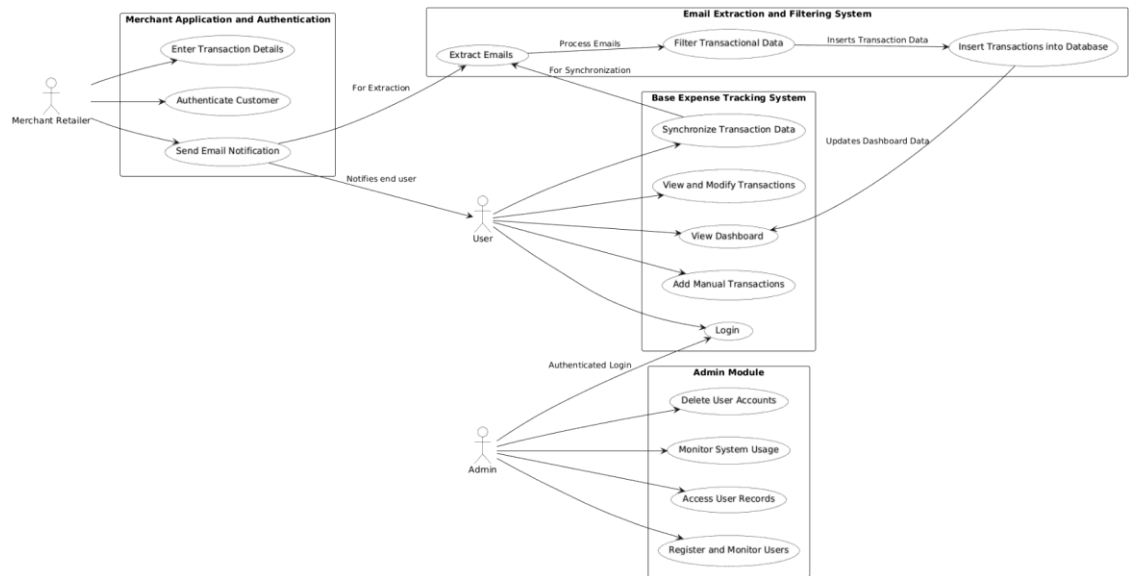


Figure 4 Use Case Diagram for Overall System Interacting with each other

2.2.2. 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:

1) **Entity-Relationship (ER) Diagram:**

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

2) **Data Flow Diagram (DFD):**

- a) The DFD illustrates the flow of data between external entities (users, merchants, and email systems) and the internal components of the system.
- b) 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.

2.2.2.1. ER DIAGRAM

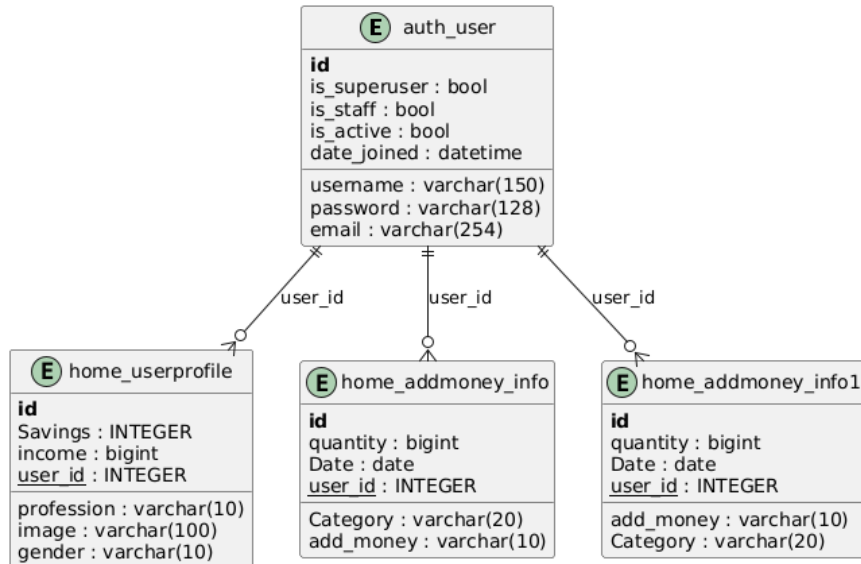


Figure 5 ER Diagram for Commutation to Base Application

2.2.2.2. DATA FLOW DIAGRAM

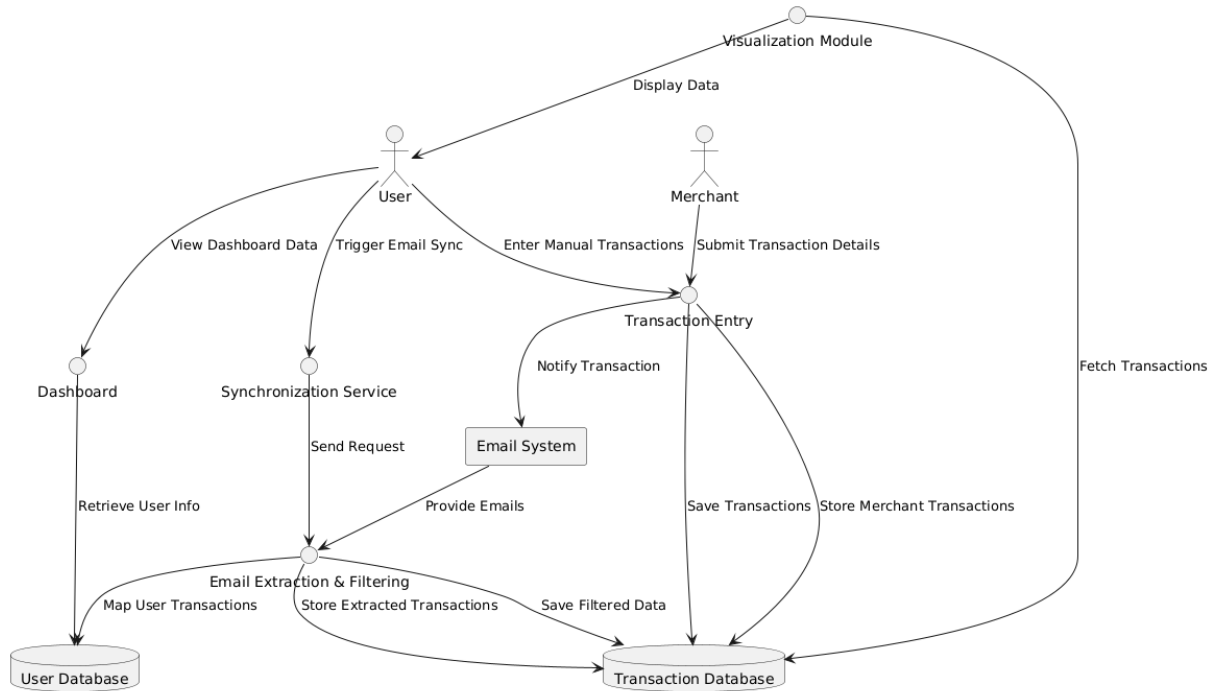


Figure 6 Data flow diagram for entire application

2.2.3. UML DIAGRAMS

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

2.2.3.1. SEQUENCE DIAGRAMS

- **Base Expense Tracking Application:** 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.

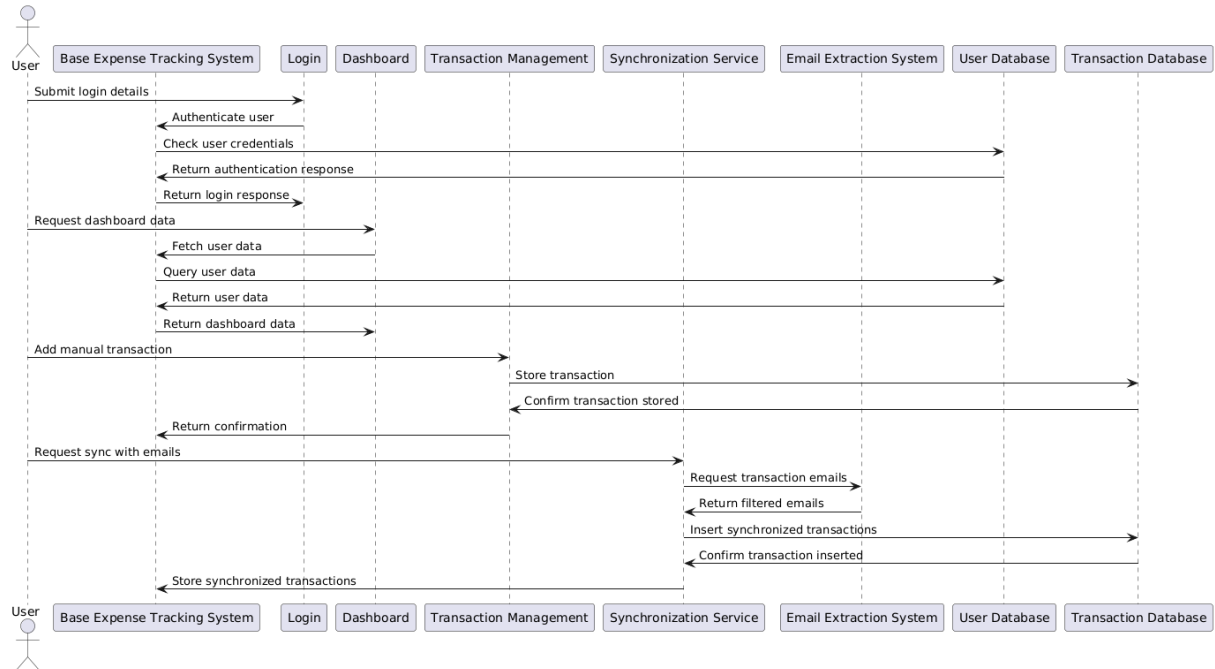


Figure 7 Sequence Diagram for E2E flow of Base Expense Tracker

- 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.

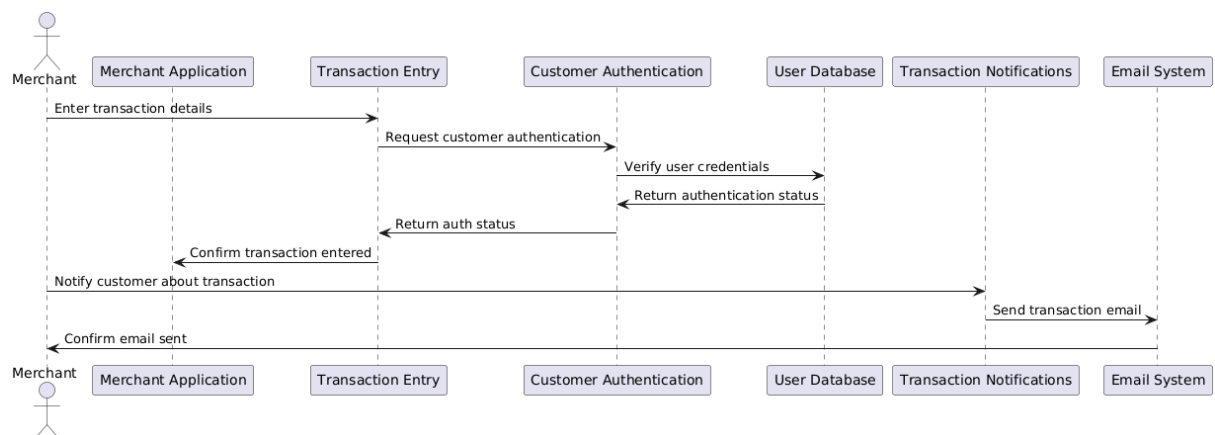


Figure 8 Sequence Diagram for E2E flow for Merchant application

- **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**.

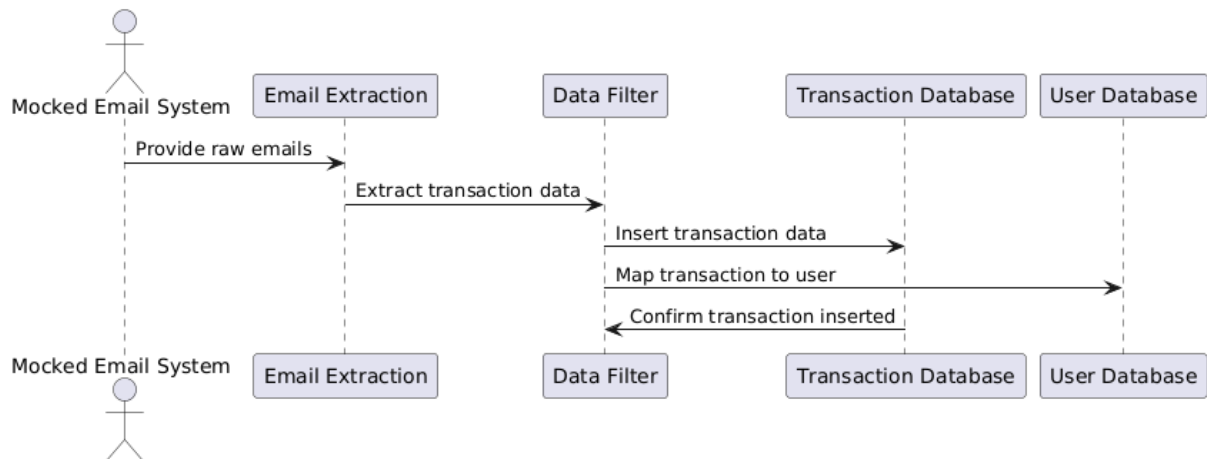


Figure 9 Sequence Diagram for Email Extraction, Filtering and Database Interaction

2.2.3.2. CLASS DIAGRAMS

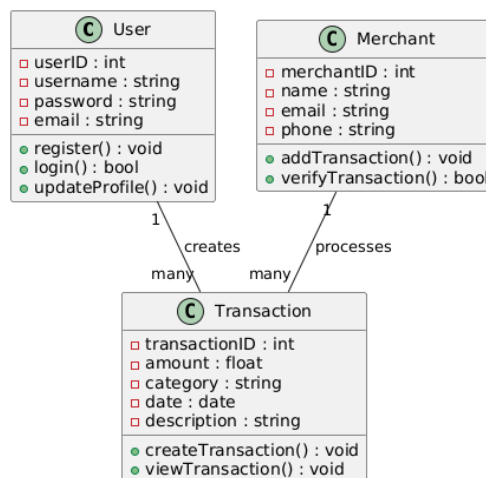


Figure 10 Class Diagram for entire software (high level - tentative)

2.2.4. 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 3 Performance Requirement for the project

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

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 constructed includes major components like the **Base Expense Tracking System**, **Merchant Application**, **Email Extraction and Filtering System**, and how these interact with each other.

Let us categorize this architecture into 3 major segments.

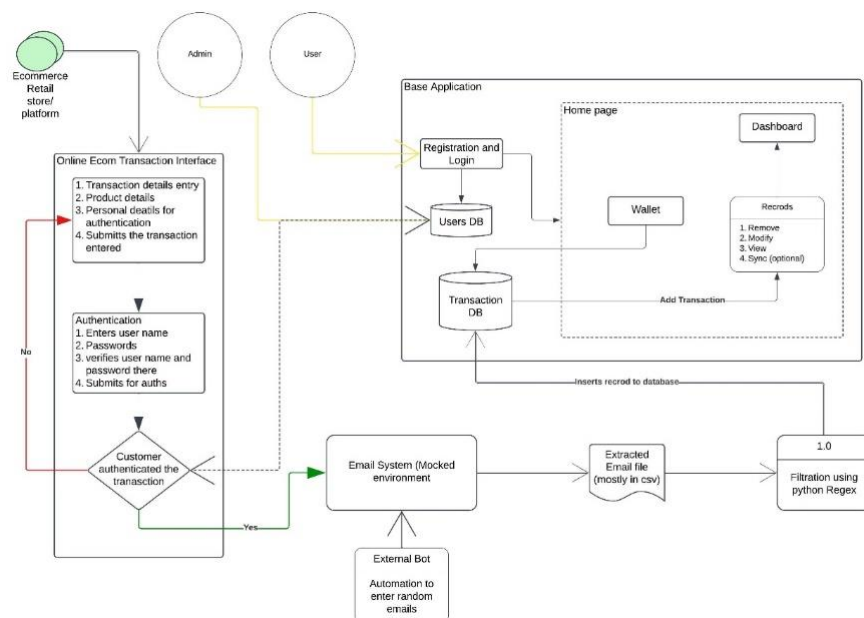


Figure 11 System Architecture of Streamlining Personal Financial Systems

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 also 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 en-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.

Here's a structured table summarizing the roles and steps for your 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.).
	Viewing Transaction History	View past transactions and update details if needed.
	Resolving Disputes	Assist customers in resolving issues related to notifications or transactions.

Table 4 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.
- **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.

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). The scope of testing is detailed below list,

- Validate that all system functionalities, including user management, transaction handling, email extraction, and merchant interactions, function as expected.
- Ensure integration between modules is seamless and data flows correctly.
- Verify that the system is scalable and handles expected loads efficiently.
- Provide confidence in the stability and correctness of the system through thorough testing processes.

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. For instance:

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 5 Test Scenarios for All Modules

5. Entry and Exit Criteria

Entry Criteria

- Test environments, including mock systems and databases, are set up and operational.

Exit Criteria

- All critical and high-priority test cases have been passed successfully.
- System meets performance and scalability requirements.
- The system is ready for deployment based on successful system testing and the completion of non-functional testing.

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.
- **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.
- **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.
- **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.

DATABASE DESIGN

The database consists of several interrelated tables designed 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.
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.
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.
4. **home_addmoney_info1**: Stores additional transaction data, categorized separately for specific use cases. References auth_user through the user_id foreign key.

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 6 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 7 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 8 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.

SCREENSHOTS

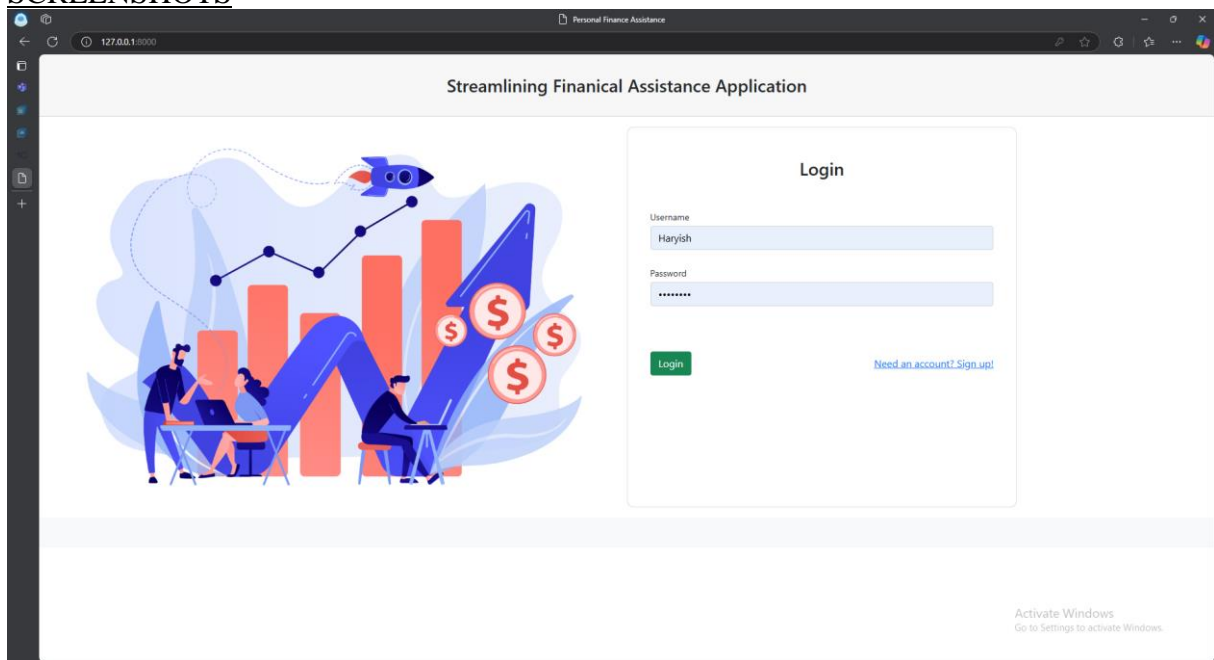


Figure 12 Login to Base Application

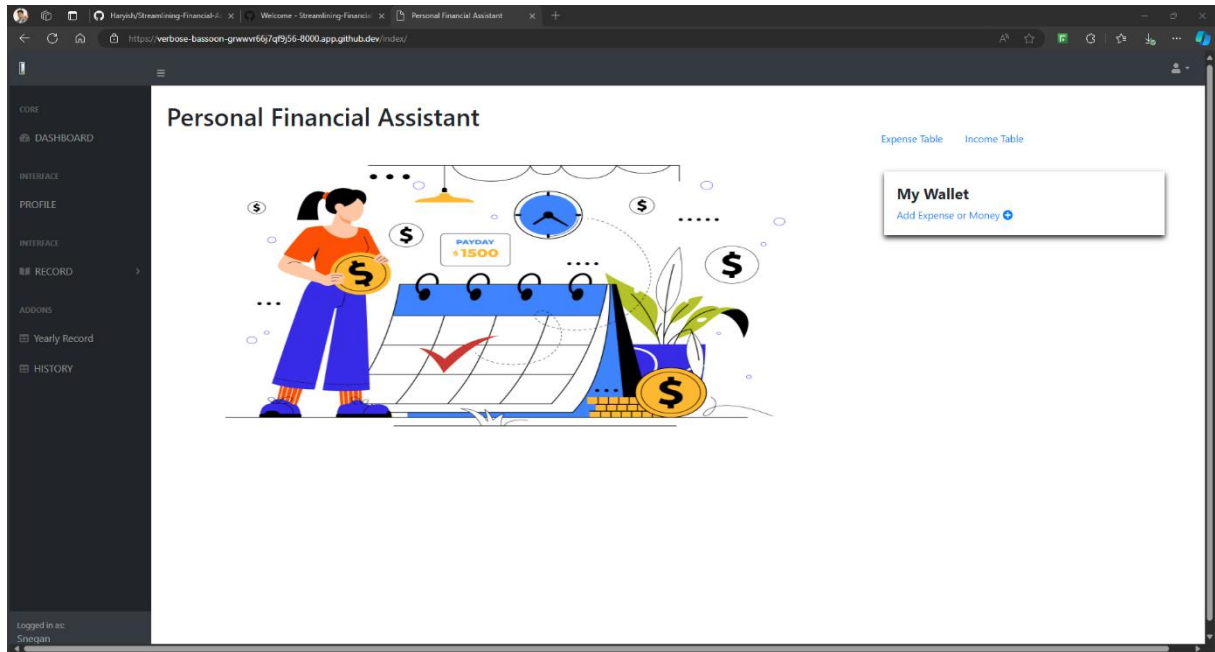


Figure 13 Home Page on Base Application

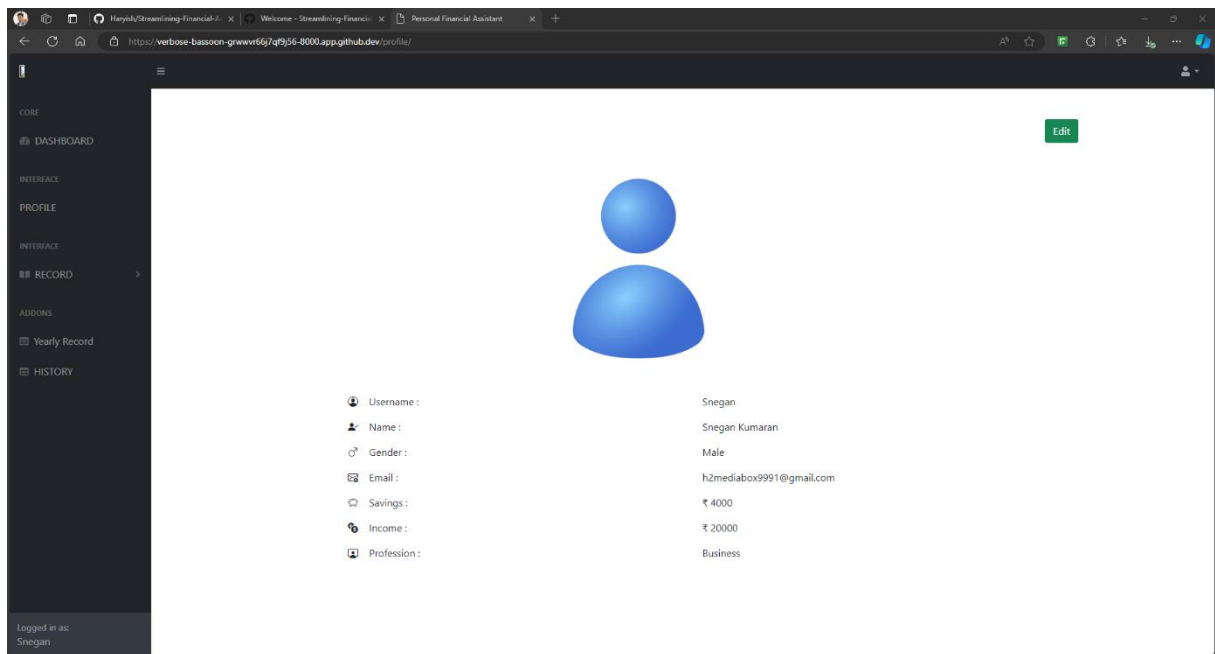
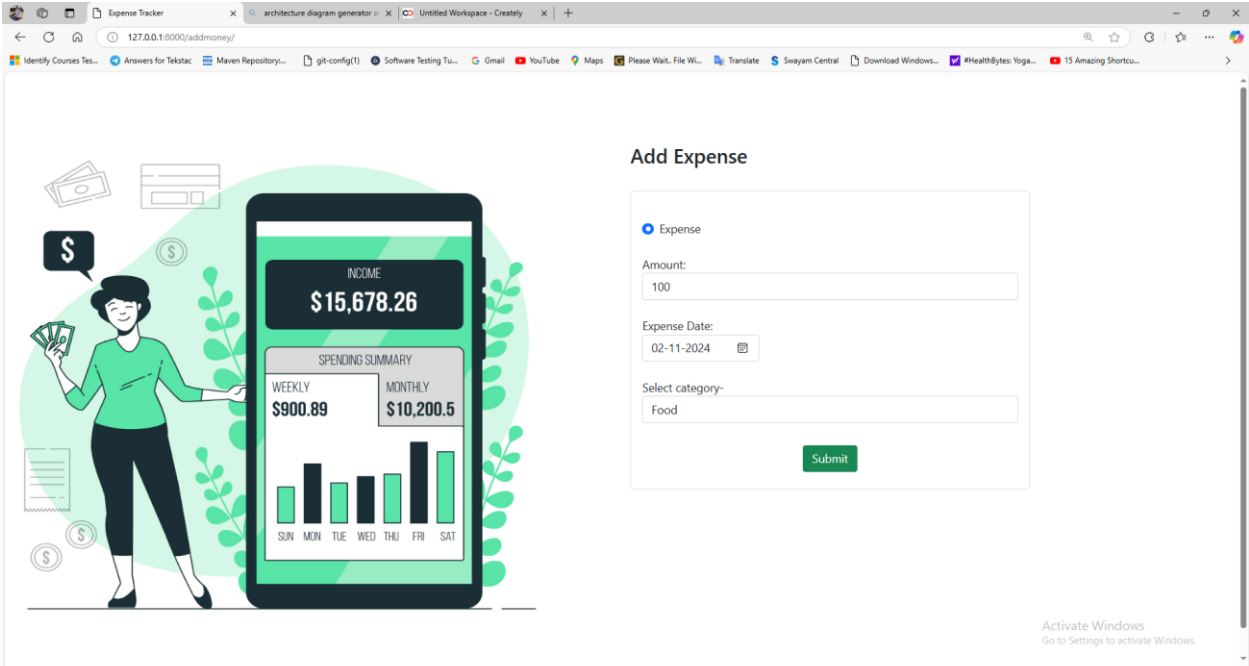


Figure 14 Profile to Base Application



The image shows a web browser window with the URL `127.0.0.1:8000/addmoney/`. On the left, there is an illustration of a woman in a green shirt and black pants holding a smartphone. The phone screen displays an 'INCOME' of \$15,678.26 and a 'SPENDING SUMMARY' with weekly and monthly totals. To the right of the illustration is the 'Add Expense' form.

Add Expense

☒ Expense

Amount:
100

Expense Date:
02-11-2024

Select category:-
Food

Submit

At the bottom right, there is an 'Activate Windows' watermark.

Figure 15 Transaction Manual Entry

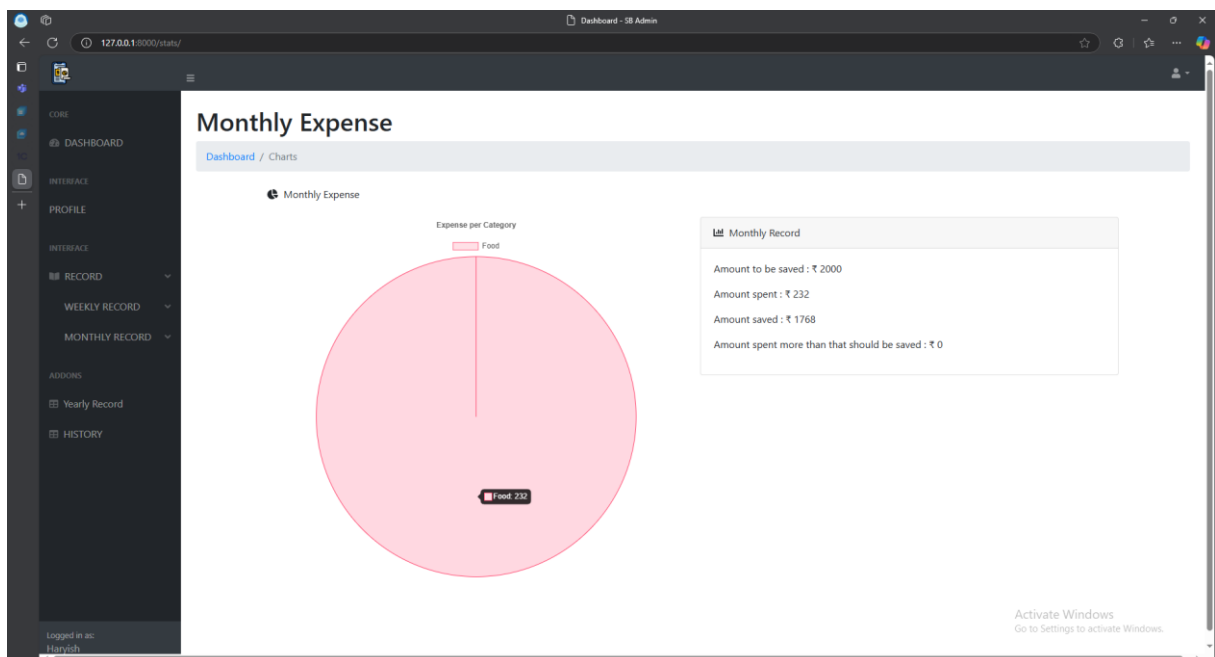


Figure 16 Data Visualization of Monthly Expenses

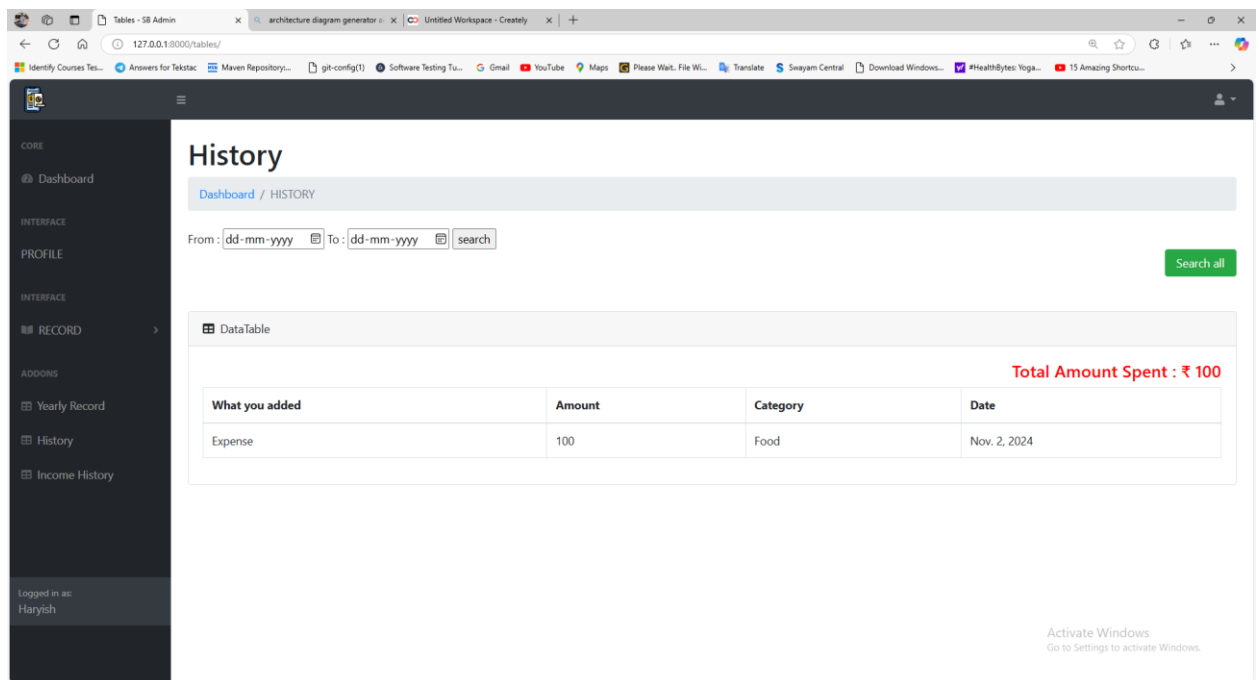


Figure 17 Transaction History - Existing Approach

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.

5. CONCLUSION

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 **Automated Expense Tracking System** 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 **Automated Expense Tracking System** 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.

5.2. FUTURE WORK

While the Automated Expense Tracking System 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 Mohammed Yasmeen Patel**, "Income and Expense Tracker," *Indian Journal of Science and Technology*, vol. 8, no. S2, January 2015.
- [2] **Ma Liqian, Tang Siyu, and J. Black Michael**, "Customized Multi-person Tracker," *Springer International Publishing, Asian Conference on Computer Vision*, Perth, June 2019.
- [3] **Lee, C. F., Chen, H. Y., & Lee, J.**, "Econometric Approach to Financial Analysis, Planning, and Forecasting," *Financial Econometrics, Mathematics, and Statistics*, Springer, New York, NY, 2019.
- [4] **P. Bhatele, D. Mahajan, B. Mahajan, D. Mahajan, N. Mahajan, and P. Mahajan**, "TrackEZ Expense Tracker," *2023 4th International Conference for Emerging Technology (INCET)*, Belgaum, India, 2023.
- [5] **J. L. Yeo, P. S. JosephNg, K. A. Alezabi, H. C. Eaw, and K. Y. Phan**, "Time Scheduling and Finance Management: University Student Survival Kit," *2020 IEEE Student Conference on Research and Development (SCORED)*, Batu Pahat, Malaysia.
- [6] **A. Tamizhselvi, M. Anbu, and K. R. Radhakrishnan**, "Financial and Individual Future Expense Prediction Based on Frequent Patterns Using Micro Services," *2022 Third International Conference on Intelligent Computing Instrumentation and Control Technologies (ICICT)*, Kannur, India, 2022.
- [7] **Bhujang, B. D., Wendole, P. V., Thakare, A. P., Ghodele, P. C., & Khan, S. W.**, "Streamline Personal Finance and Budget Planner," *2023 International Conference for Emerging Technology*.
- [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] **Rahul Shetty**, *Selenium Webdriver with PYTHON from Scratch + Frameworks*, Udemy Course available at [Selenium Webdriver with PYTHON from Scratch + Frameworks | Udemy](#)

APPENDICES

A.1. DETAILS OF PROPOSED ENHANCEMENTS

1. Live API Integrations

Objective: Enable real-time synchronization of transaction data with external platforms.

Benefits:

- Eliminates the need for mocked environments, enhancing the system's practicality.
- Reduces manual effort and potential errors in data entry.

Implementation Challenges:

- Handling diverse API formats and security protocols.
- Ensuring compliance with data protection laws (e.g., GDPR, CCPA).

Proposed Steps:

- Develop modular connectors for each platform (e.g., Gmail, Plaid, Stripe).
- Use OAuth 2.0 for secure API authentication and authorization.
- Implement robust error handling to manage API outages or failures.

2. Automated Categorization

Objective: Replace manual tagging with machine learning-based automation.

Benefits:

- Improves system efficiency by reducing user intervention.
- Enhances the accuracy of transaction categorization.

Implementation Challenges:

- Training models on diverse datasets to ensure adaptability across users.
- Handling ambiguous or incomplete transaction descriptions.

Proposed Steps:

- Use supervised learning models like Support Vector Machines (SVM) or Random Forest for initial implementation.
- Incorporate user feedback to refine model accuracy over time.
- Employ cloud-based solutions for scalable model training and deployment.

3. Advanced Analytics

Objective: Provide actionable insights and predictive capabilities for users' financial data.

Benefits:

- Enhances user engagement through personalized financial advice.
- Identifies trends and potential risks proactively.

Implementation Challenges:

- Processing large volumes of data in real time.
- Balancing computational complexity with system performance.

Proposed Steps:

- Implement ARIMA or Prophet models for time-series forecasting.
- Use clustering algorithms to identify spending patterns.
- Develop dashboards that visually represent predictions and insights.

A.2. SCOPE FOR FUTURE WORK

The integration of the above enhancements will extend the system's scope to include:

- **Fully Automated Expense Management:** Transitioning from a semi-automated to a fully automated system that minimizes user input.
- **Cross-Platform Integration:** Enabling users to manage finances across multiple platforms seamlessly.
- **Personalized User Experience:** Offering custom insights and advice tailored to individual financial behaviours.

These advancements align with modern trends in financial technology, aiming to deliver a comprehensive and intelligent expense tracking solution.