Finance Tracking App

Rebof Katwal

Rebofkatwal7@gmail.com

Table of Contents

Table of Figures

Table of Tables

1. Project Title

The Expense Tracker is a cross-platform desktop application intended to help users easily manage personal finances. Users are also able to track inflows and outflows in cash, besides debts, without many hustles. This application supports custom tags, comprehensive financial summary dashboards, and transaction searching, filtering, and sorting by type, date, and tag. It was designed to be friendly for the user and ensured that debts were managed efficiently and that pending payments came into view. The data will be kept safe but shared in JSON format for easy and structured handling. This application is meant to make financial management a lesser hassle by providing users an intuitive and efficient way of tracking.

1.1 Purpose

This application is designed to enable users in tracking expenses by registering information and providing a secure and interactive environment. It ensures data security by offering SHA256 password hashing and facilitates good financial management.

1.2 Scope

It will also provide the functionality to the user in managing their money, tracking transactions, showing highlights, and managing debts. Some of the important features are user registration, analytics on the dashboard, transaction history, and user-configurable settings. Abstraction for JSON storage usage and a modular architecture for scalability reasons are some of the key technical concepts taken into consideration during its development.

1.3 Objectives

The main objective is to provide an efficient and user-friendly application for expense management that enables easy filtering, and provide insight into one's finance. This project aims at showing how code should be organized using interfaces, service layers, and model structures effectively.

2. Project Features and Functionalities

2.1 Registration and Login Feature

Displaying the Home page, which provides options to either register a new user or log in.

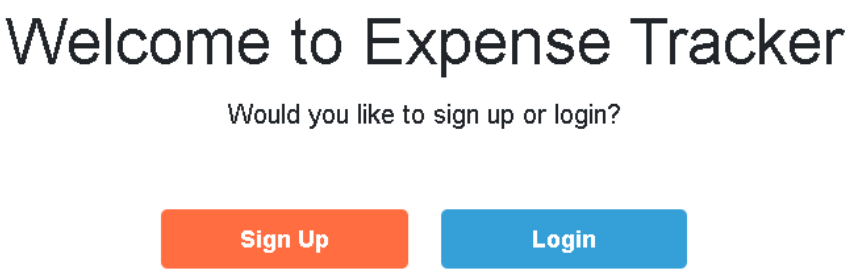


Figure 1: Home Page

Registering a user with the following details.

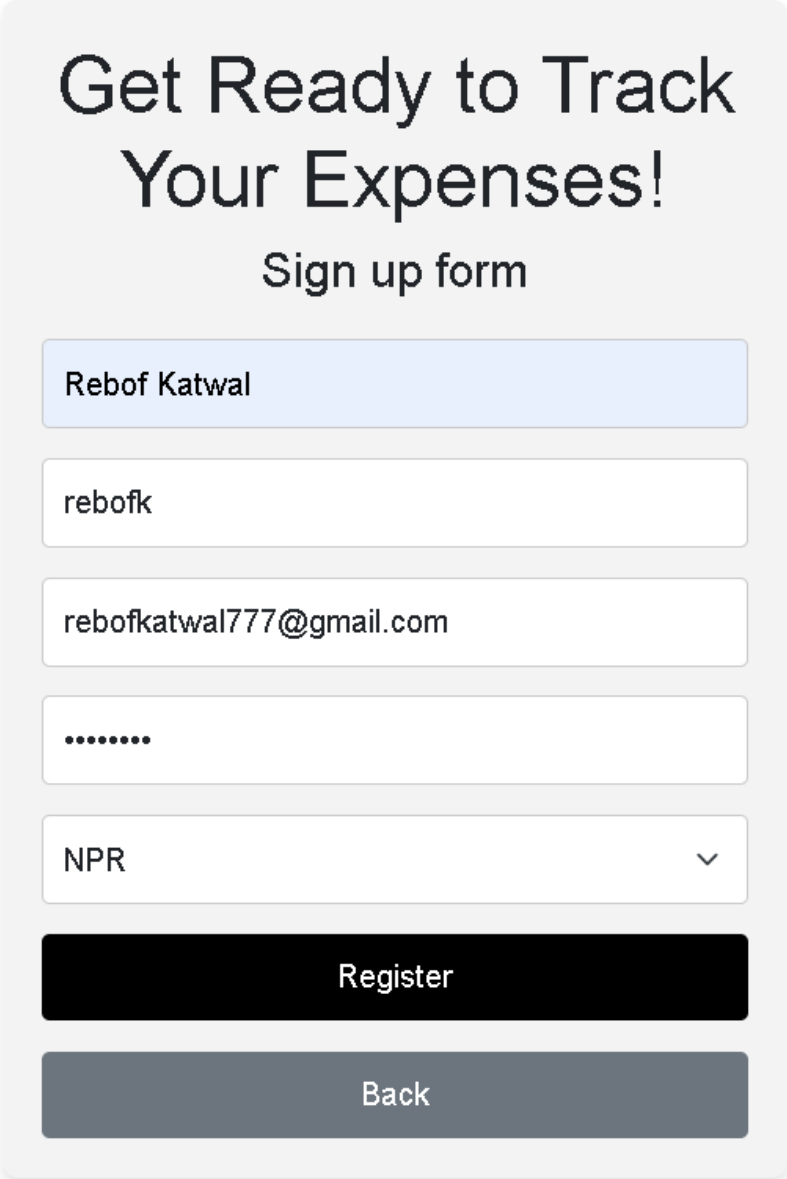


Figure 2: Registration Page

Logging in with same username and password after registration

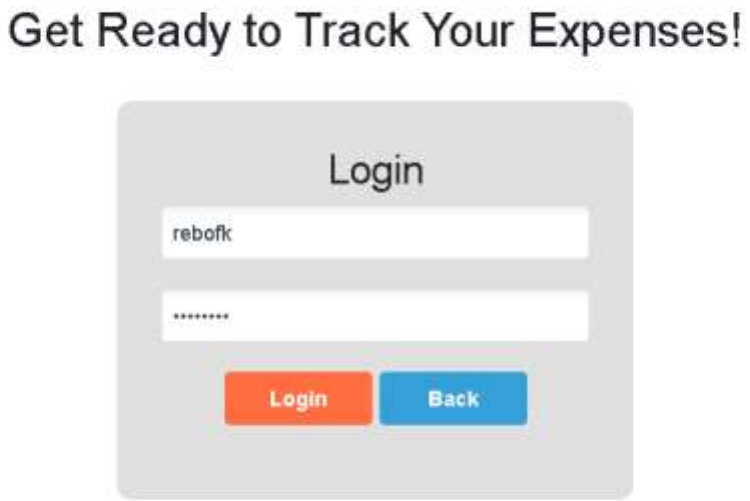


Figure 3: Login Page

2.2 Transaction History Page

After logging we can se the transaction history page where we can filter through the transactions as well.

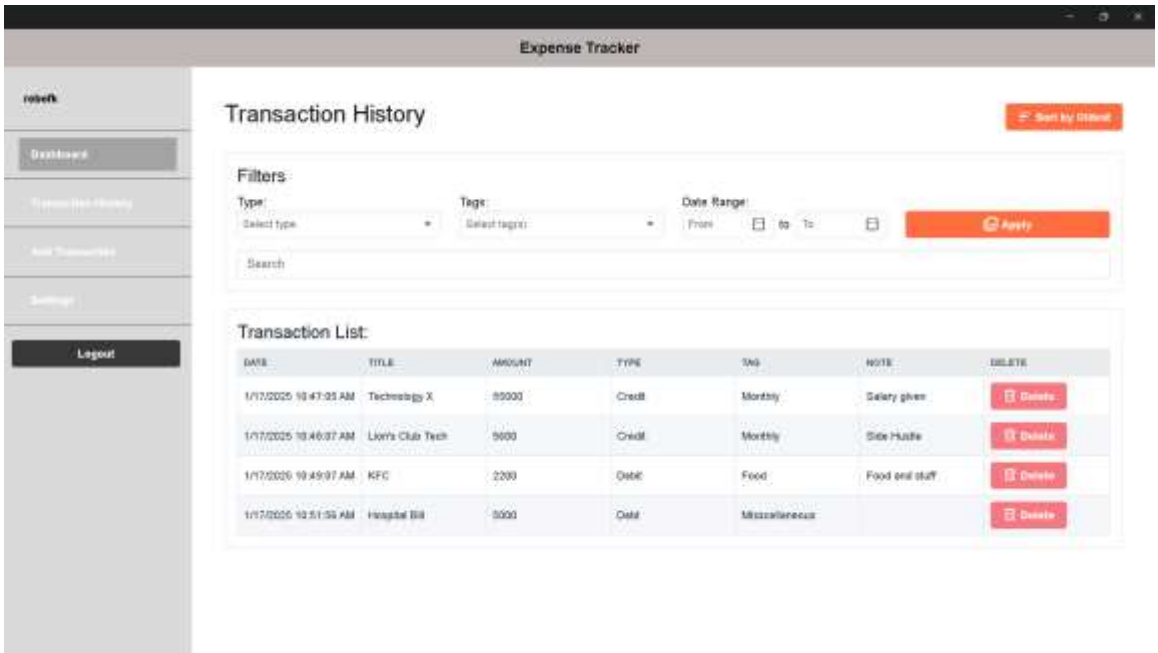


Figure 4: Transaction History Page

2.2.1 Sorting by Date

Using the “Sort by Latest’ button

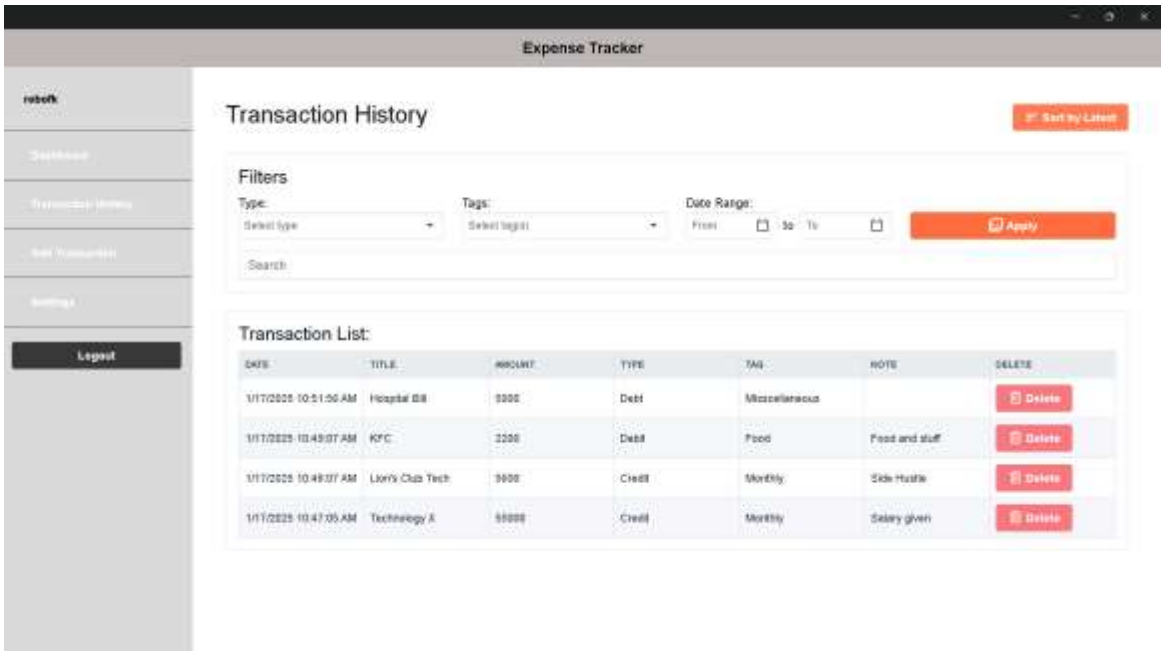


Figure 5: Sorting by date feature

2.2.2 Filtering by Tag

Searching for Monthly Tags.

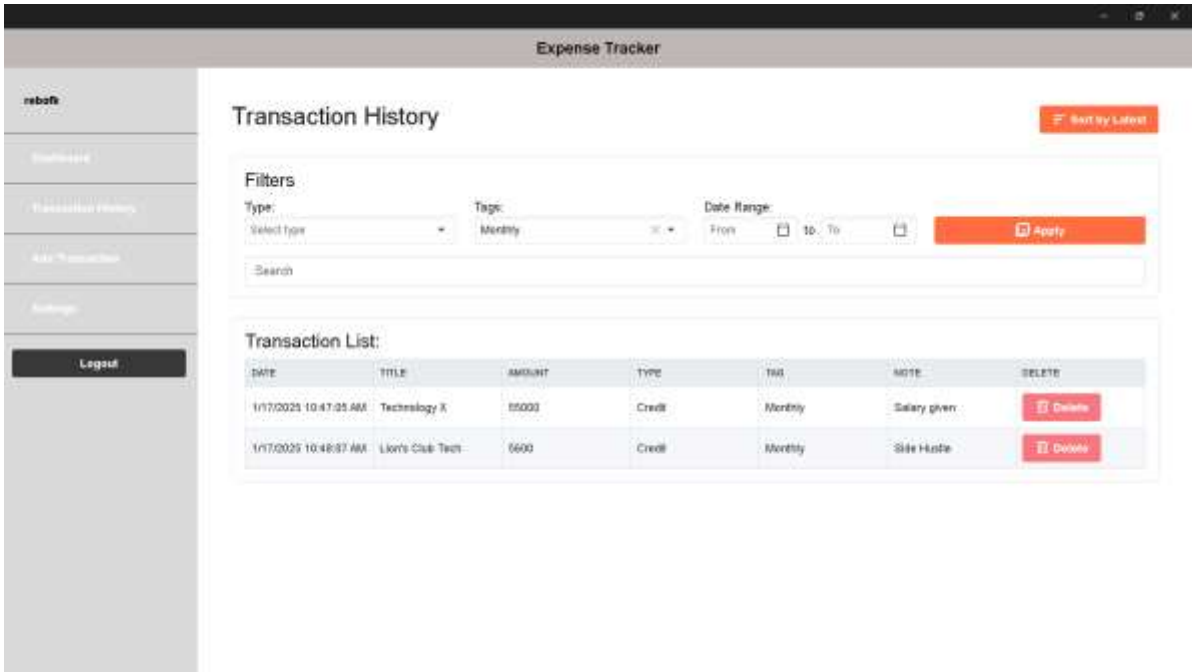


Figure 6: Filtering by tag feature

2.2.3 Filtering by Type

Searching for Debit Type transaction.

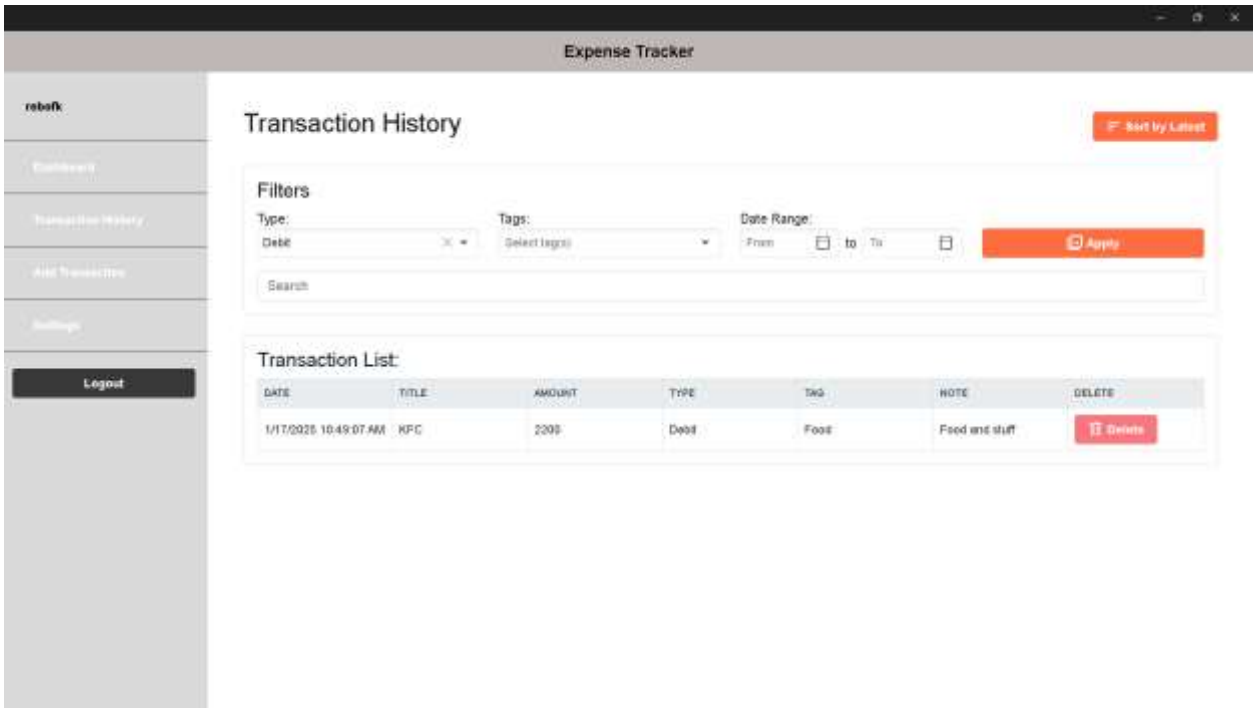


Figure 7: Filtering by type feature

2.2.4 Filtering by Date Range

Applying a date range filter; however, no records are displayed as there is no data within the selected range.

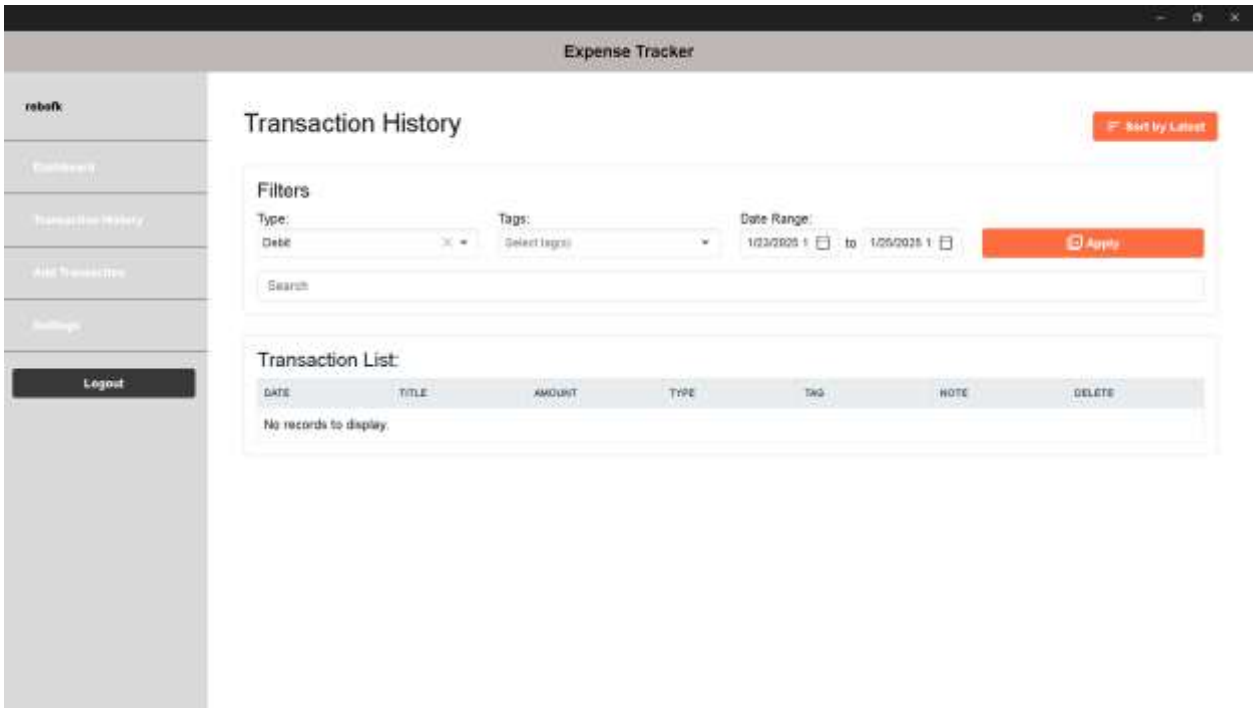


Figure 8: Filtering by date range feature

2.2 Dashboard

The dashboard page displays statistics and highlights the lowest and highest transactions for each type.

The pie chart illustrates the inflow and outflow of cash and debts.

The donut chart shows the transaction count.

Pending debts are also displayed, which can be paid off if there is enough balance available.

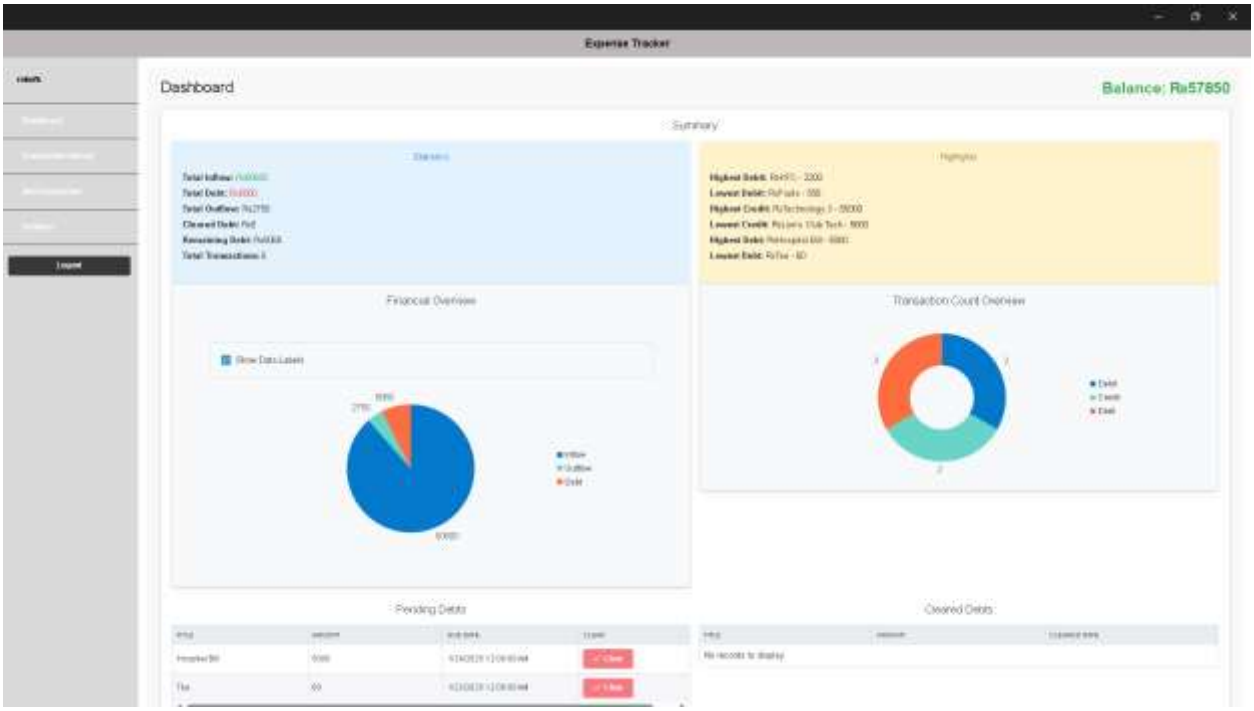


Figure 9: Dashboard Page

2.3.1 Clearing the Debts

After clearing the debts, there is a deduction in the balance as well.

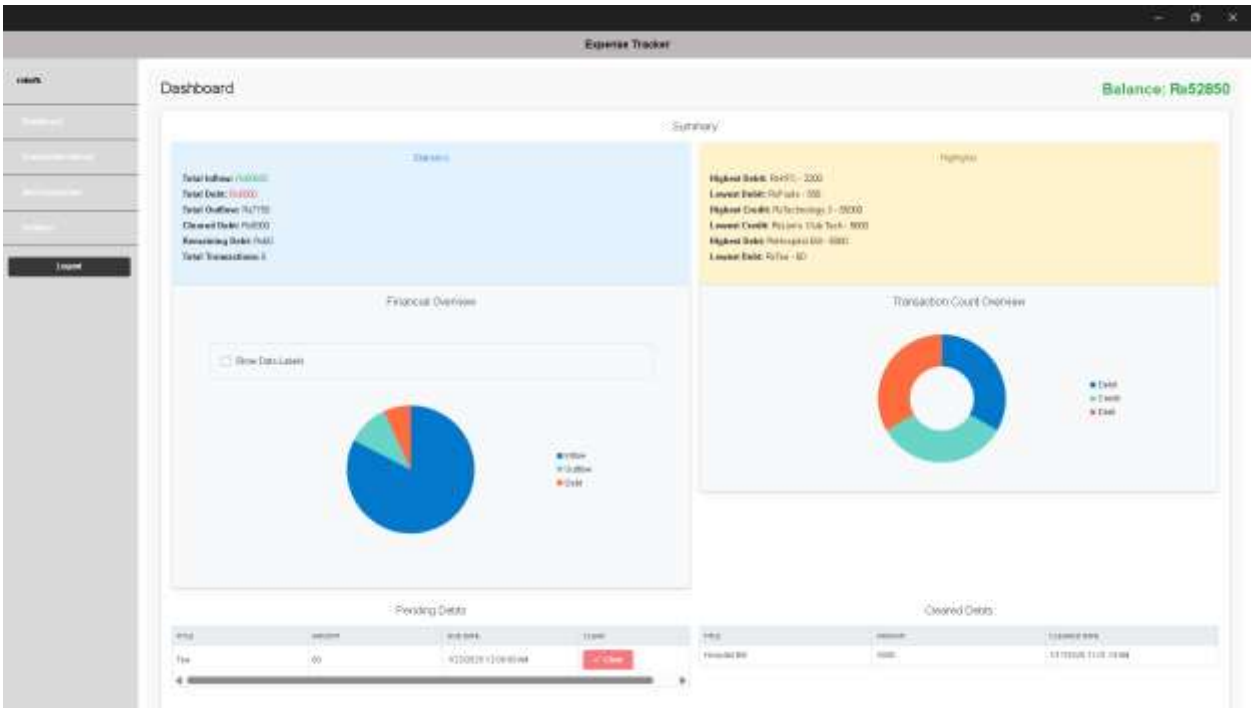


Figure 10: Clearing the debt feature

2.3 Adding Transaction

2.3.1 Adding Credit

I have combined debit, credit, and debt on the same page for user convenience. This way, all three transaction types are accessible without the need for additional navigation pages. Here we are adding a credit transaction.

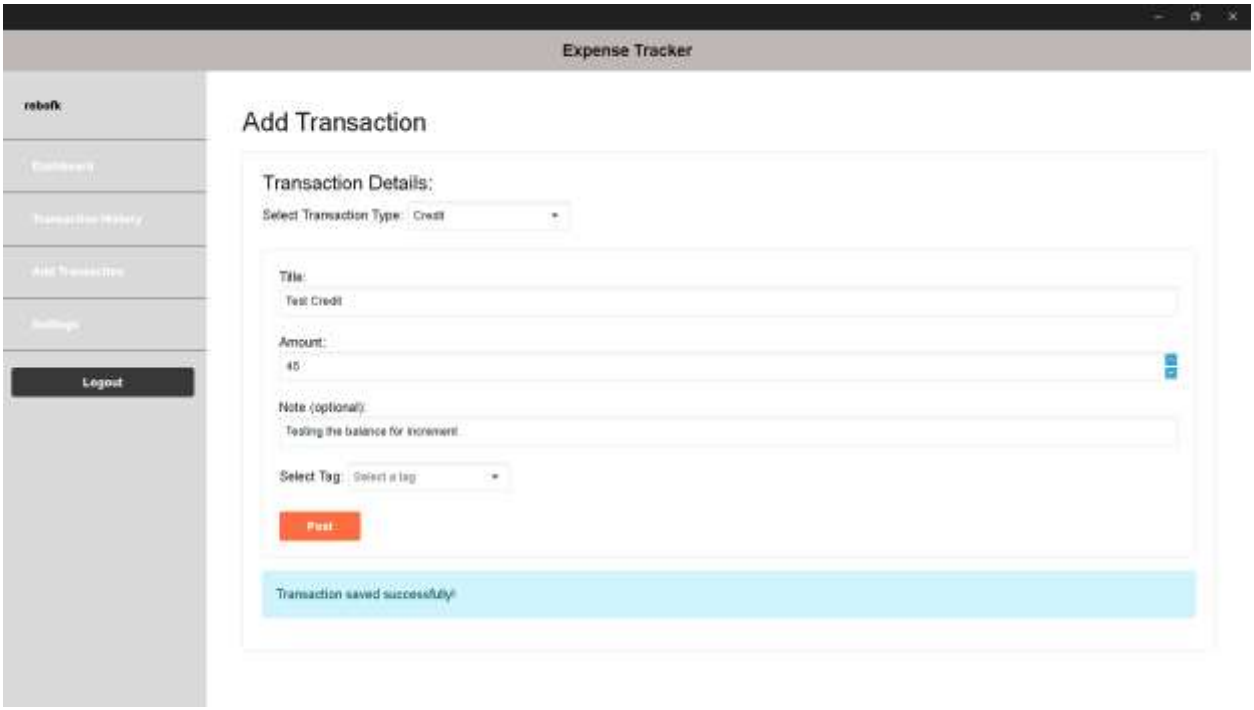


Figure 11: Adding the credit feature

Before the credit transaction, the balance was 57,850 (as shown in the previous dashboard screenshot). After adding 45, the balance has increased to 52,895.

Figure 12: Balance after credit transaction

2.3.2 Adding Debit

Now adding a debit transaction

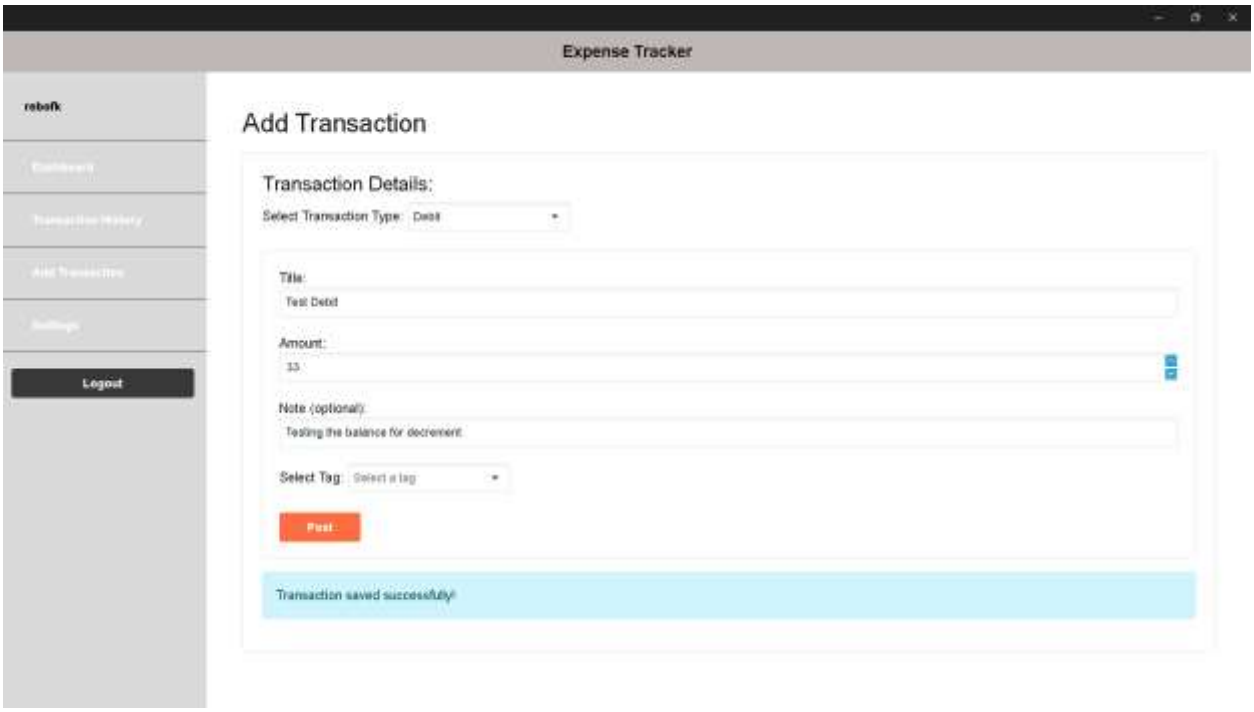


Figure 13: Adding debit feature

The balance was initially 52,895, and after deducting 33 for the debit transaction, the balance is now 52,862.

Figure 14: Balance after debit transaction

2.3.3 Adding Debts

A debt of 1000 was added.

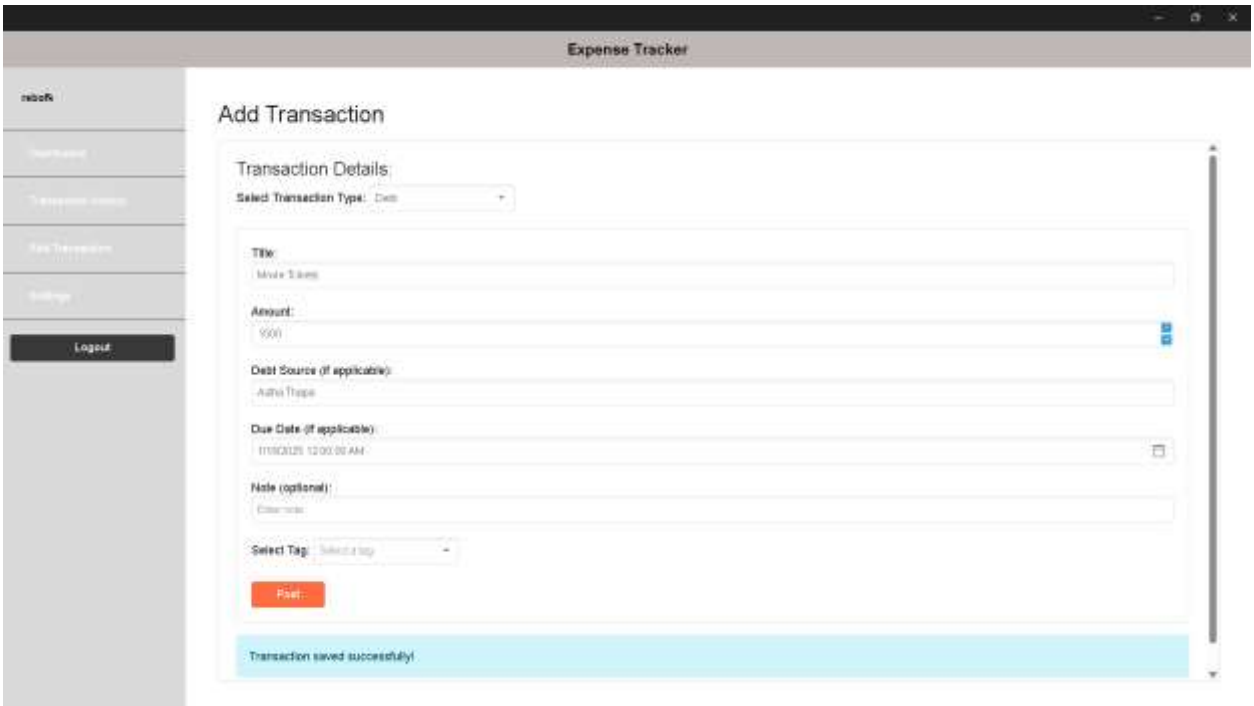
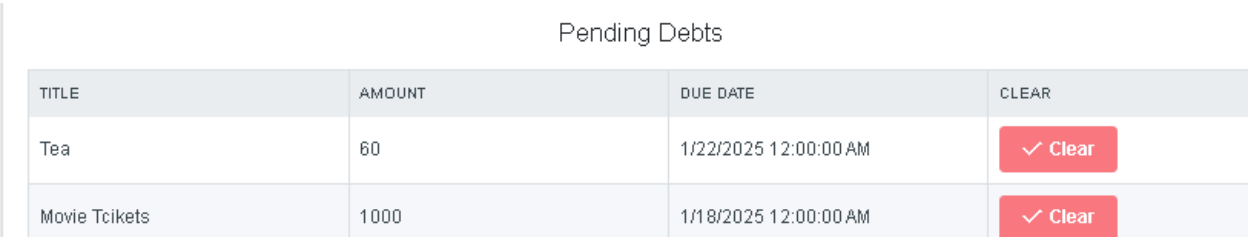


Figure 15: Adding debt feature

Shown in the dashboard as well



Pending Debts

TITLE	AMOUNT	DUE DATE	CLEAR
Tea	60	1/22/2025 12:00:00 AM	Clear
Movie Tcikets	1000	1/18/2025 12:00:00 AM	Clear

Figure 16: Showing debt in dashboard

2.4 Custom Tags

Now we will be adding a custom tag.

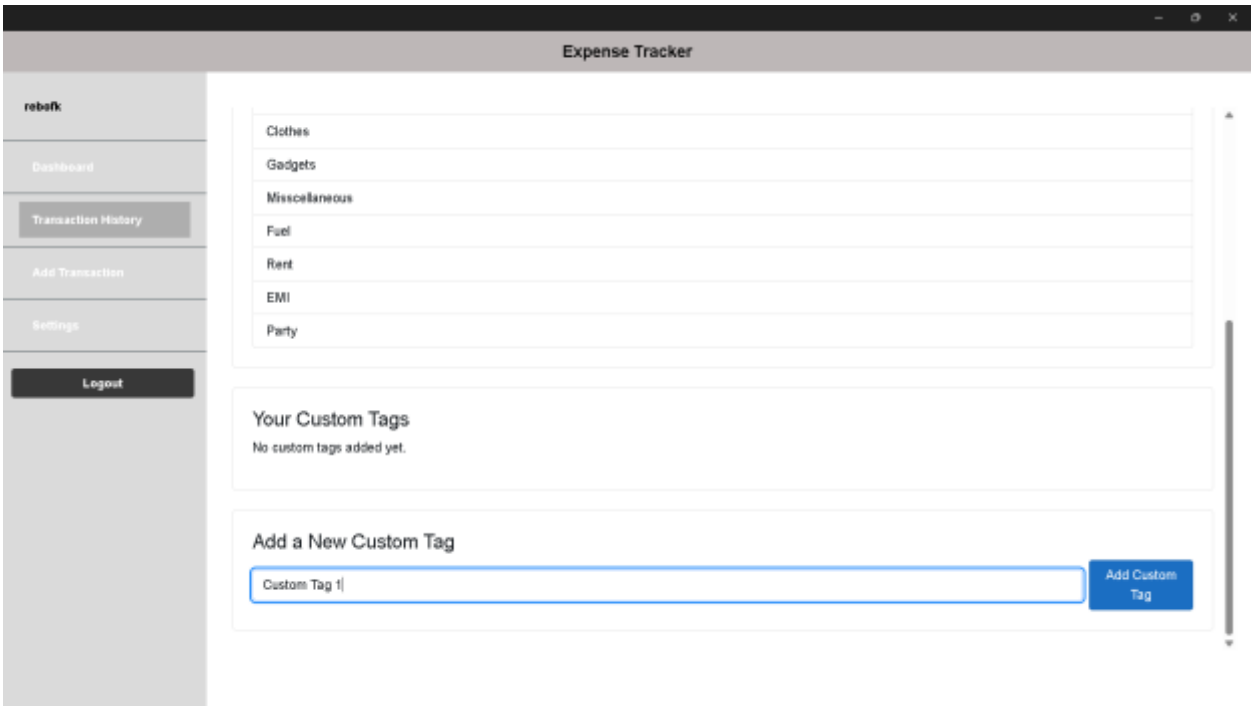


Figure 17: Custom Tags feature

Custom tag was successfully added.

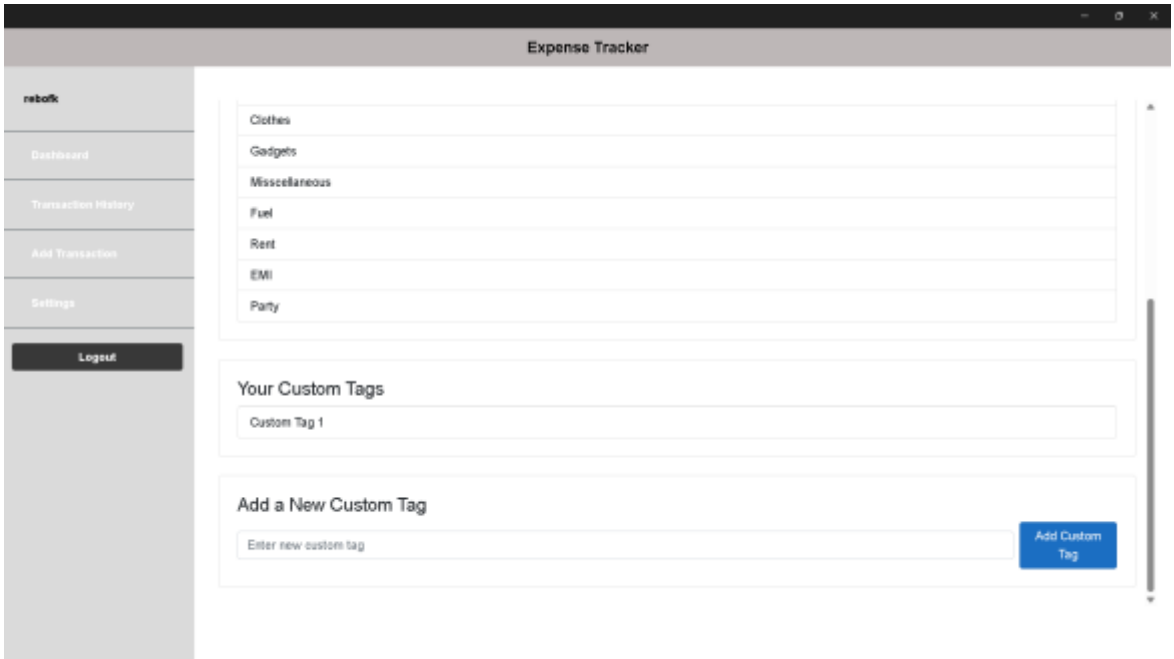


Figure 18: Added the custom tag

We can use the custom tag while adding any type of transaction.

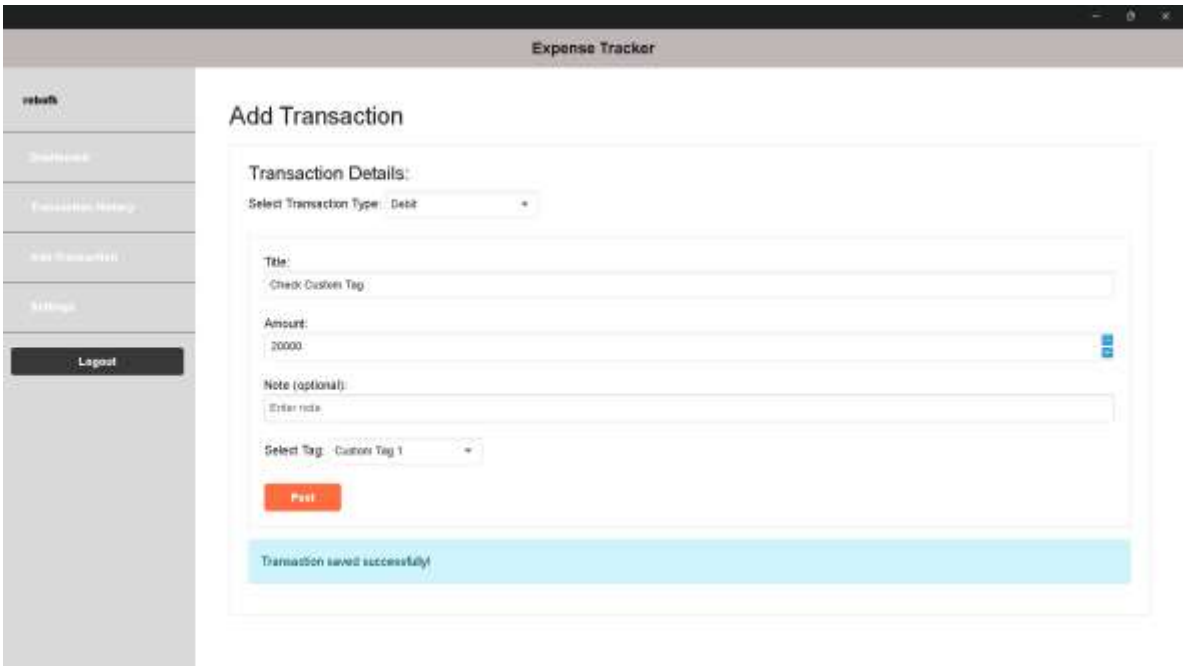


Figure 19: Showing custom tag in a transaction

We can use the custom tag for filtering as well

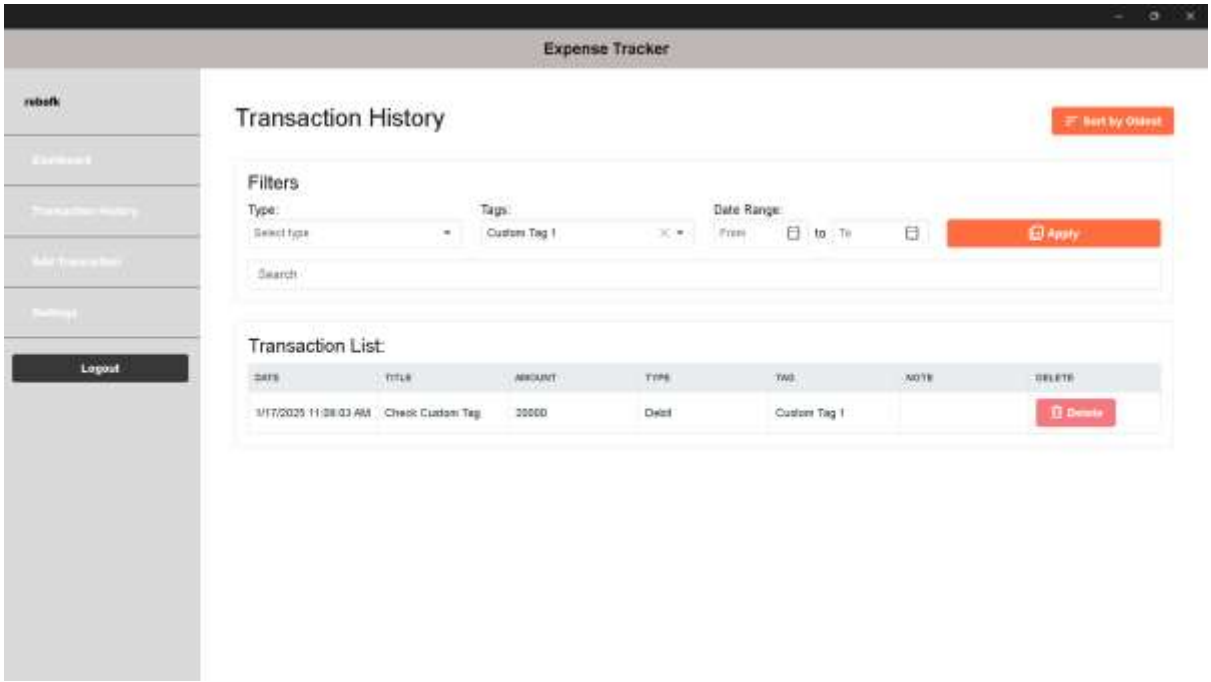


Figure 20: Filtering by custom tag

Using the delete button, I have deleted the previously shown transaction.

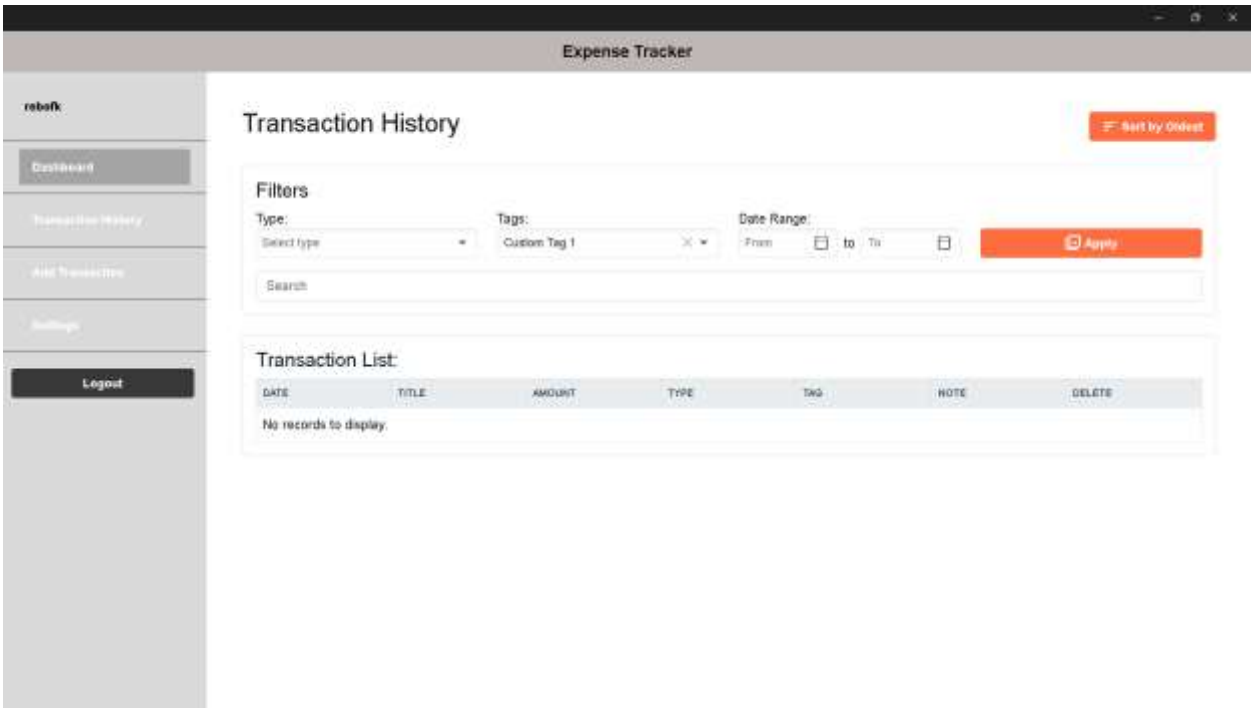


Figure 21: Using the delete button

2.5 Final Dashboard and Transaction History Page after all the Transactions.

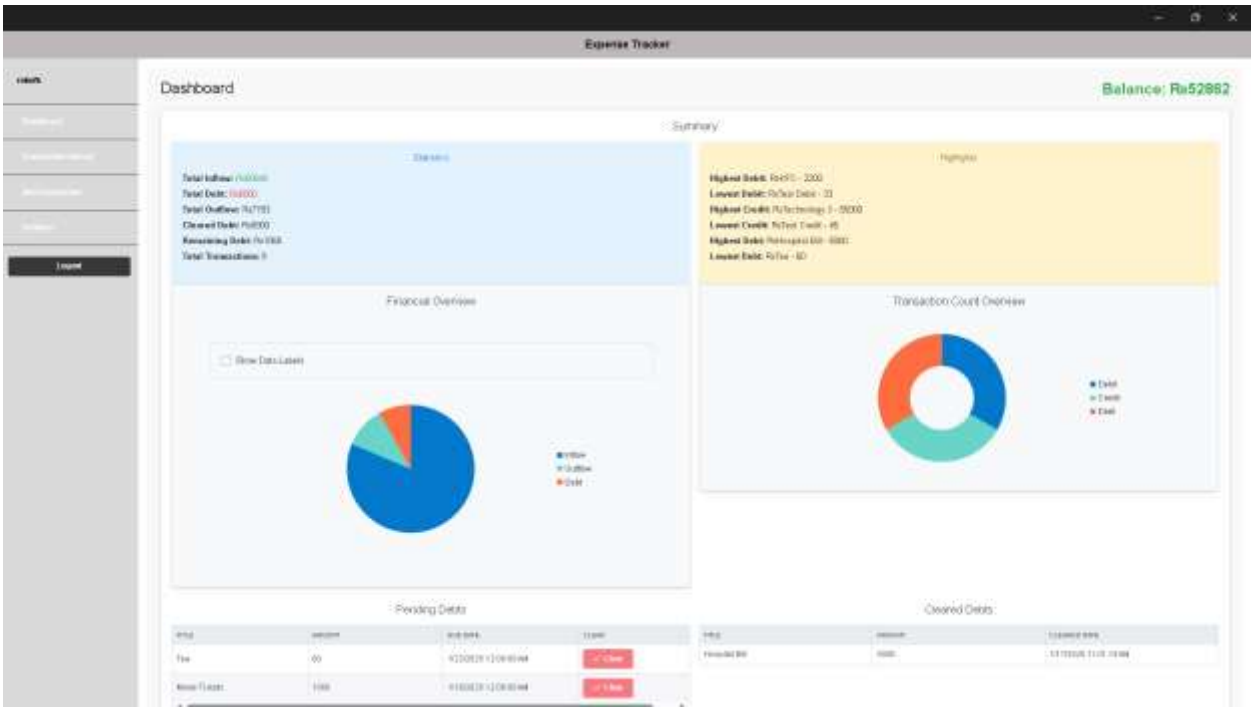


Figure 22: Final Dashboard

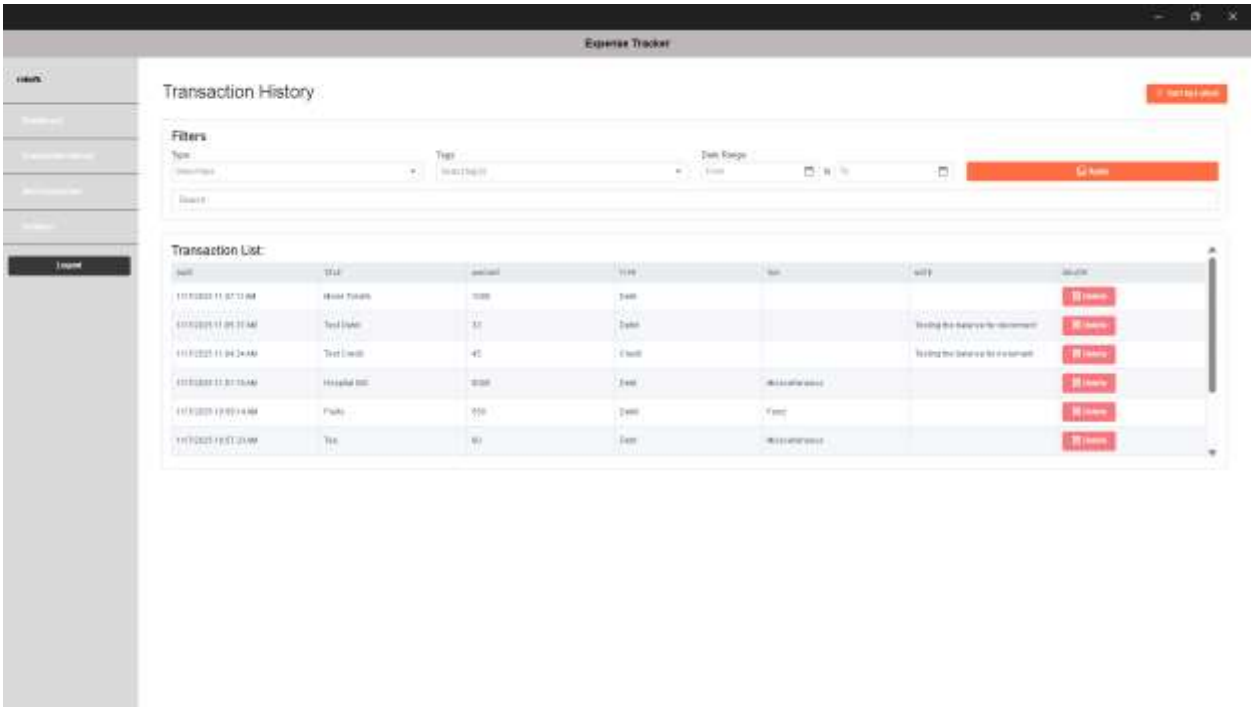


Figure 23: Final History Page

3. Proof of Work

3.1 Entity relationship Diagram

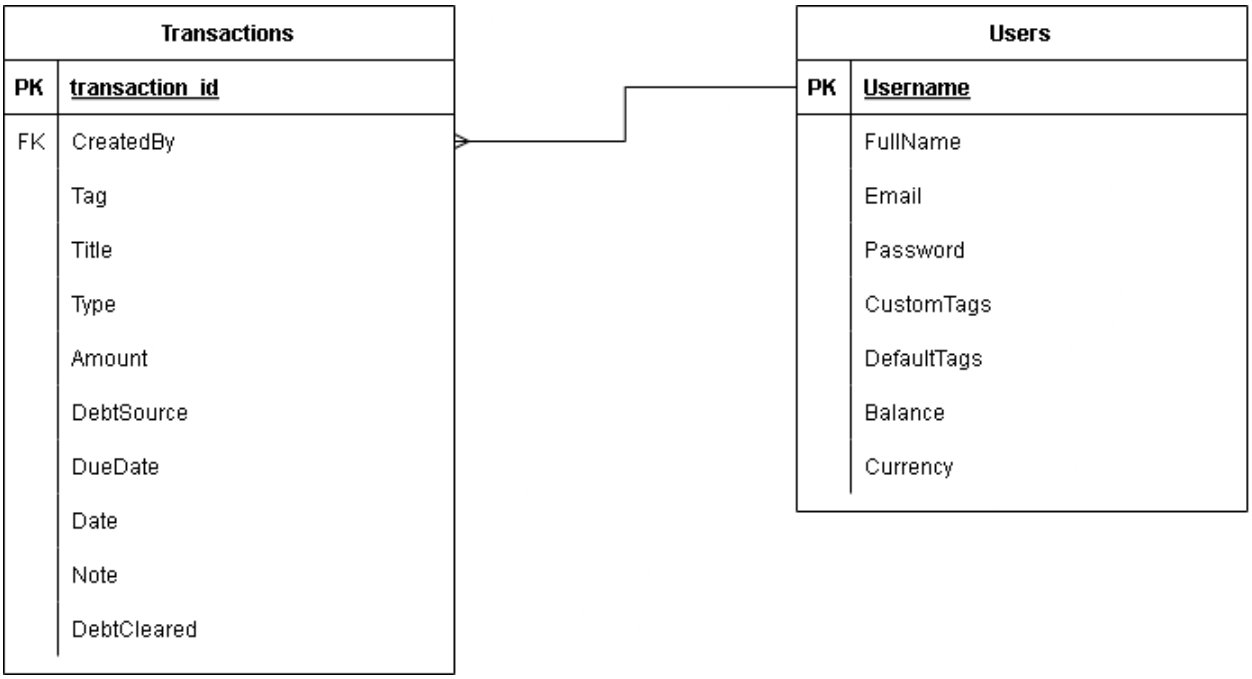


Figure 24: ERD

3.2 Wireframes

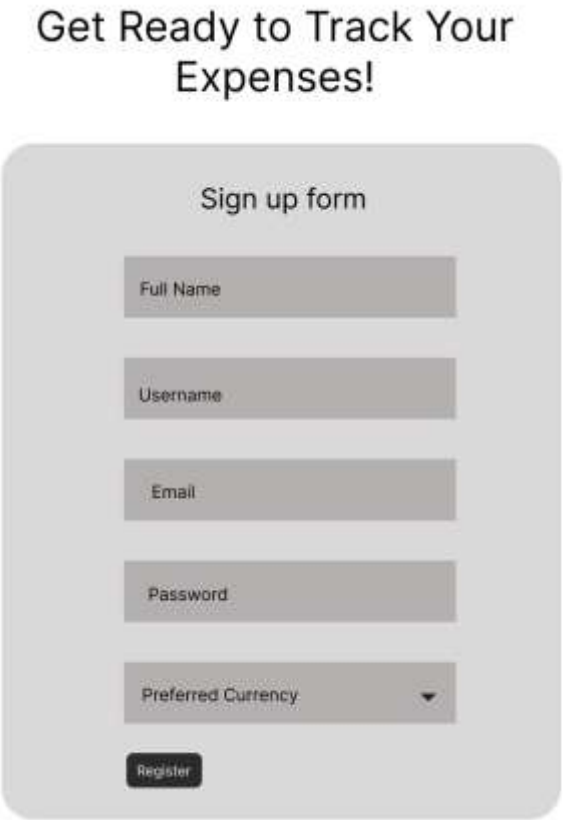


Figure 25: Registration Wireframe

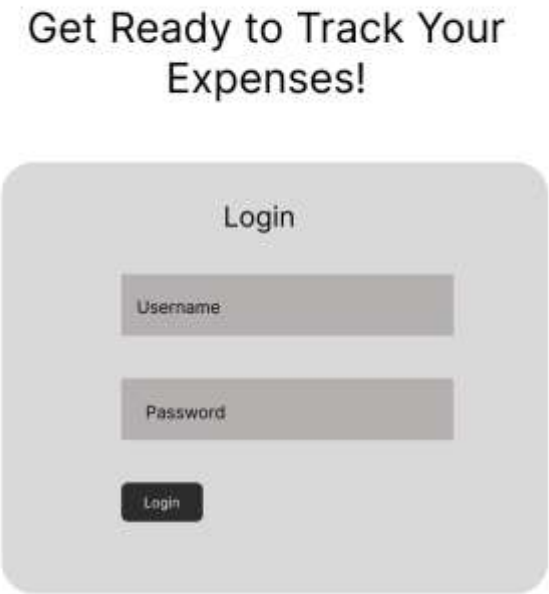


Figure 26: Login Wireframe

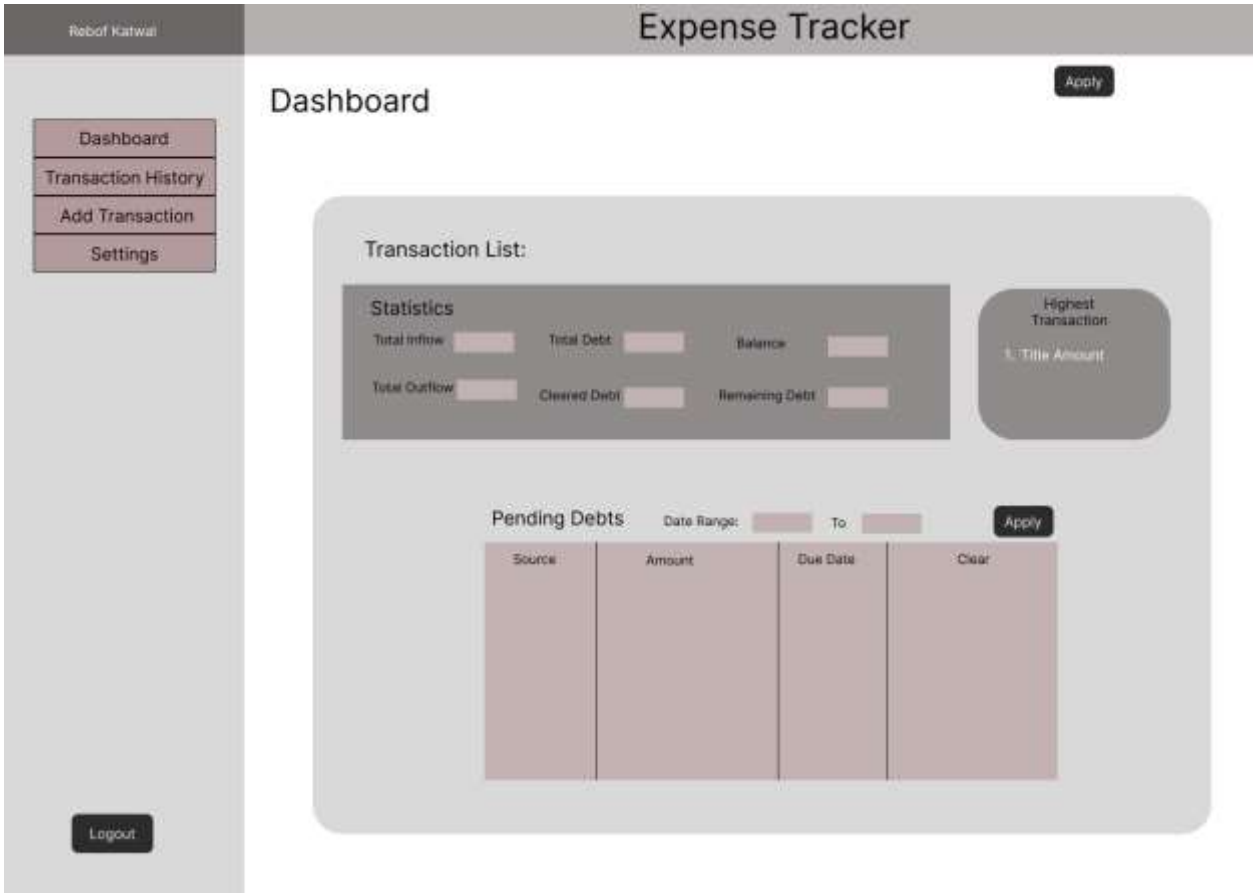


Figure 27: Dashboard Wireframe

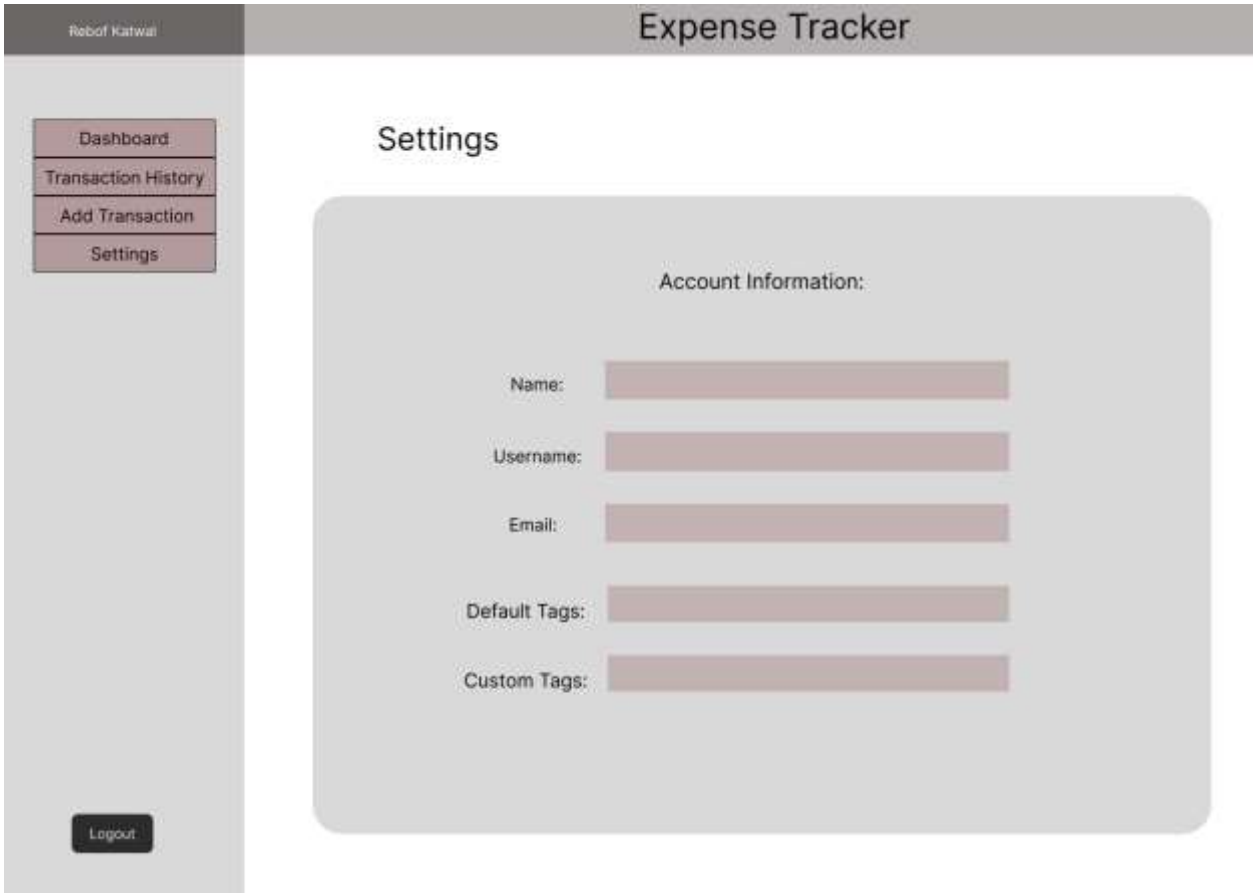


Figure 28: Settings Wireframe

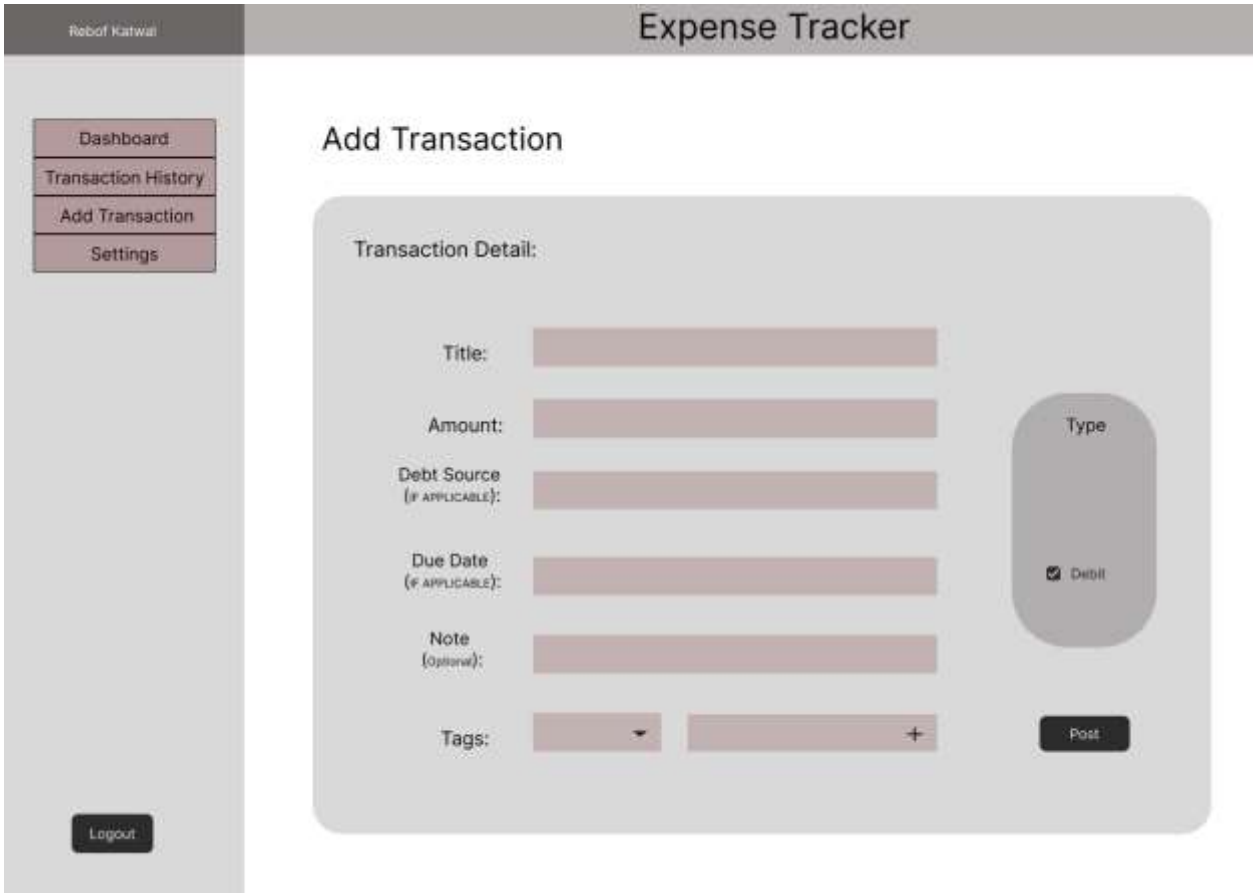


Figure 29: Add Transaction Wireframe

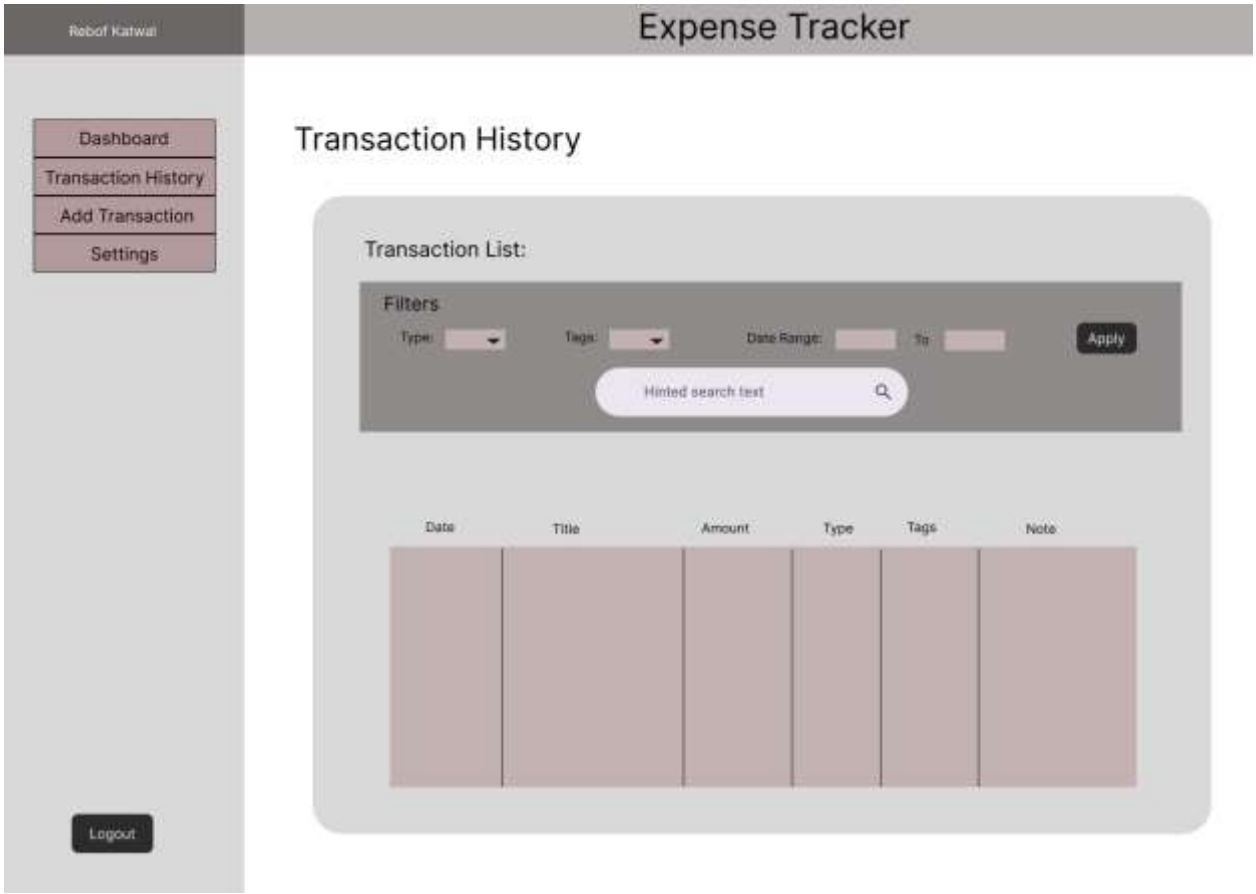


Figure 30: Transaction History Wireframe

3.3 Testing

3.3.1 Test 1

Test no:	1
Objective:	To check the registration and login validations.
Action	• Tried to keep the email empty. • Then kept the password with only one character. • The validation says wo keep @ in the email and password should be 8 characters long. • After, redirected to login. • Wrong credentials were kept. • Could not log in. • With correct credentials successful login.
Conclusion:	The test was successful.

Table 1: Test 1 Registration

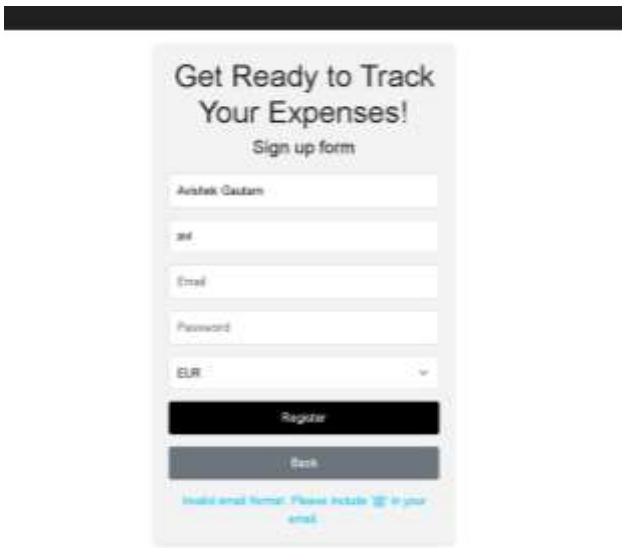


Figure 31: Validation for email

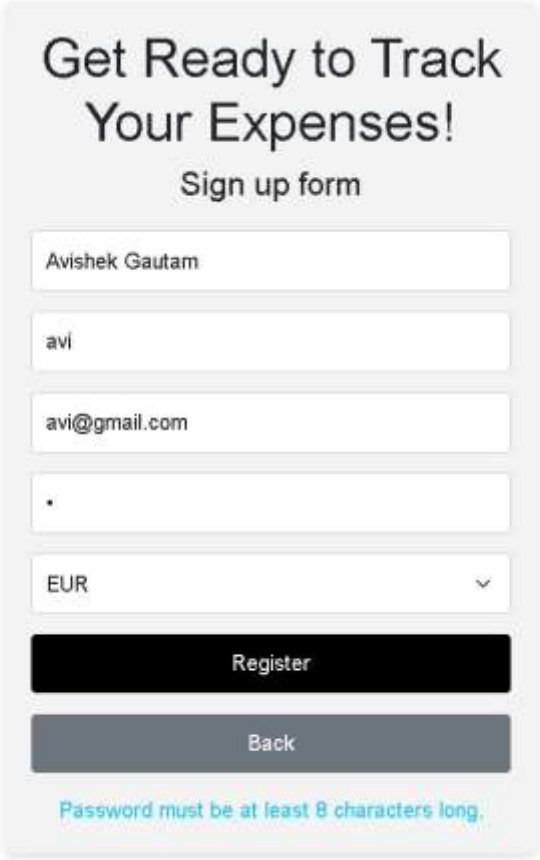


Figure 32: Validation for password creation

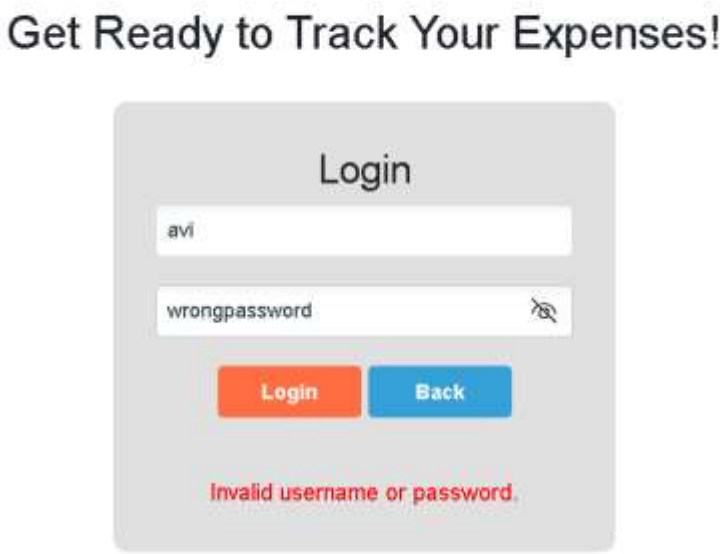


Figure 33: Checking for correct credentials

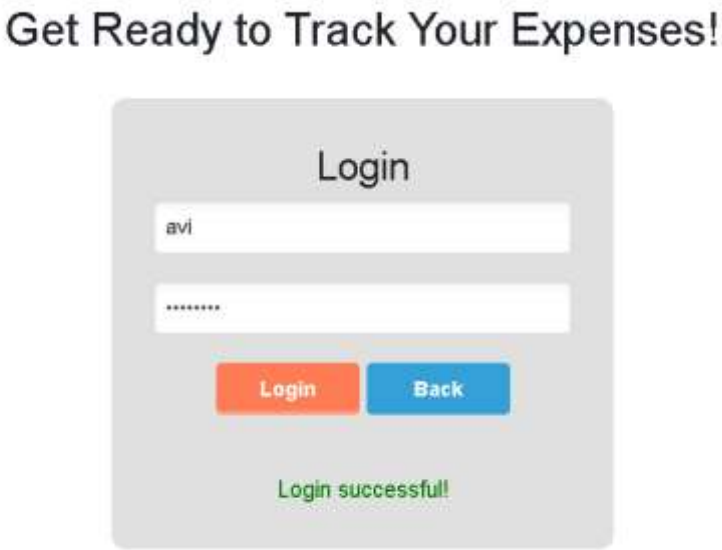


Figure 34: Successful Login

3.3.2 Test 2

Test no:	2
Objective:	To check for validation in credit/debit transaction
Action	The amount was kept at 0. Tried to post the transaction. It fails. Kept the amount at 10. It works.
Conclusion:	The test was successful.

Table 2: Test 2 Credit/Debit Transaction

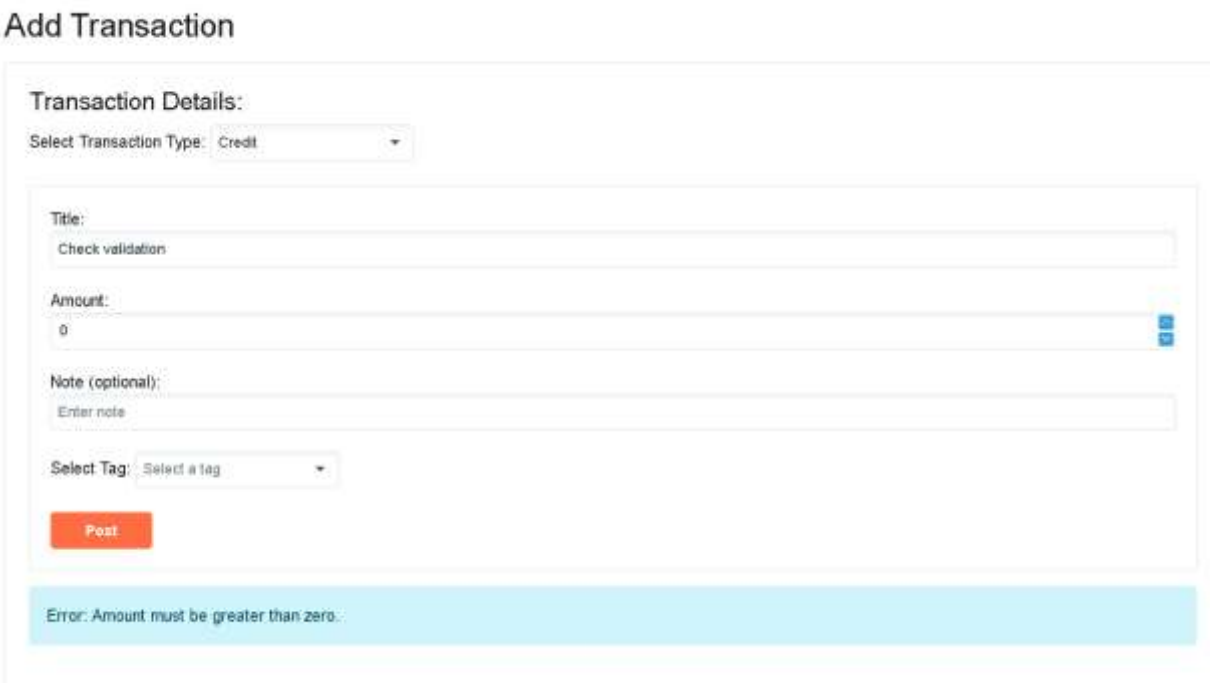


Figure 35: Amount Validation

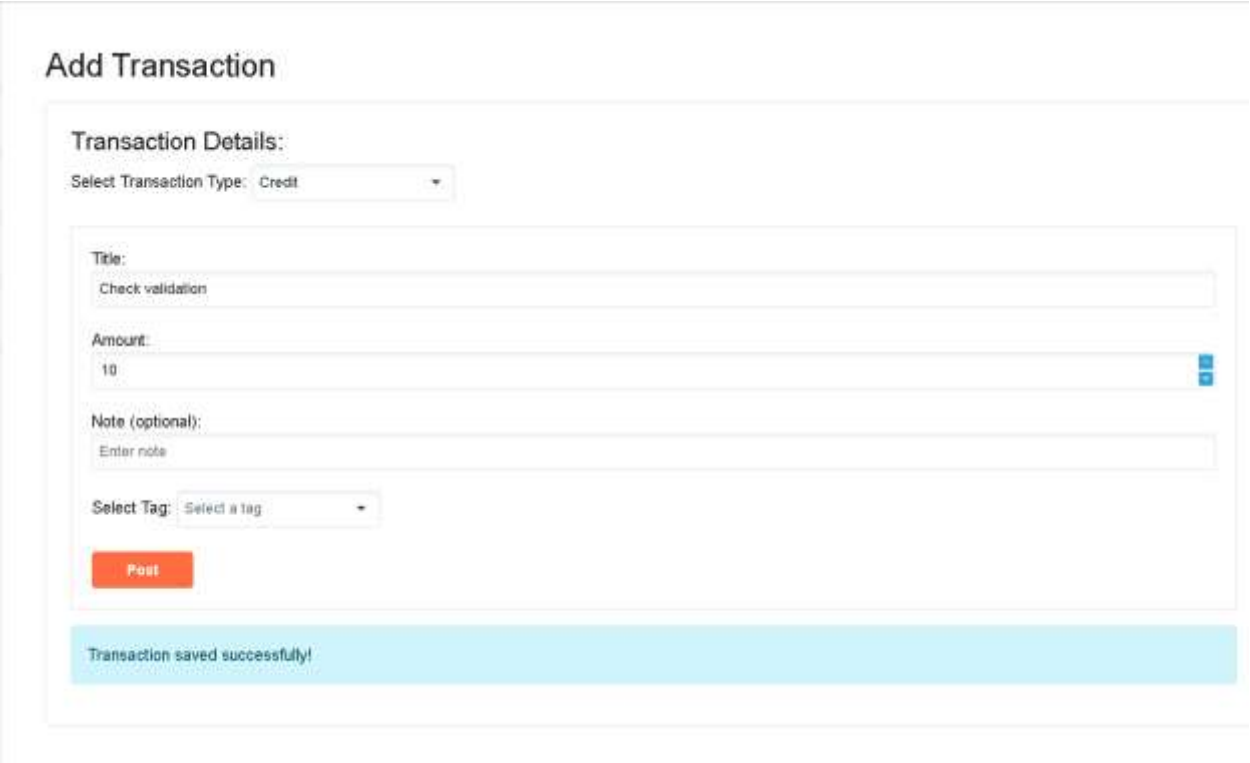


Figure 36: Successful transaction

3.3.3 Test 3

Test no:	3
Objective:	To check for validation in debt transaction
Action	• The debt source and due date were left empty. • Tried to post the transaction • It fails. • Then we enter the debt source and due date. • It works.
Conclusion:	The test was successful.

Table 3: Test 3 Debt Transaction

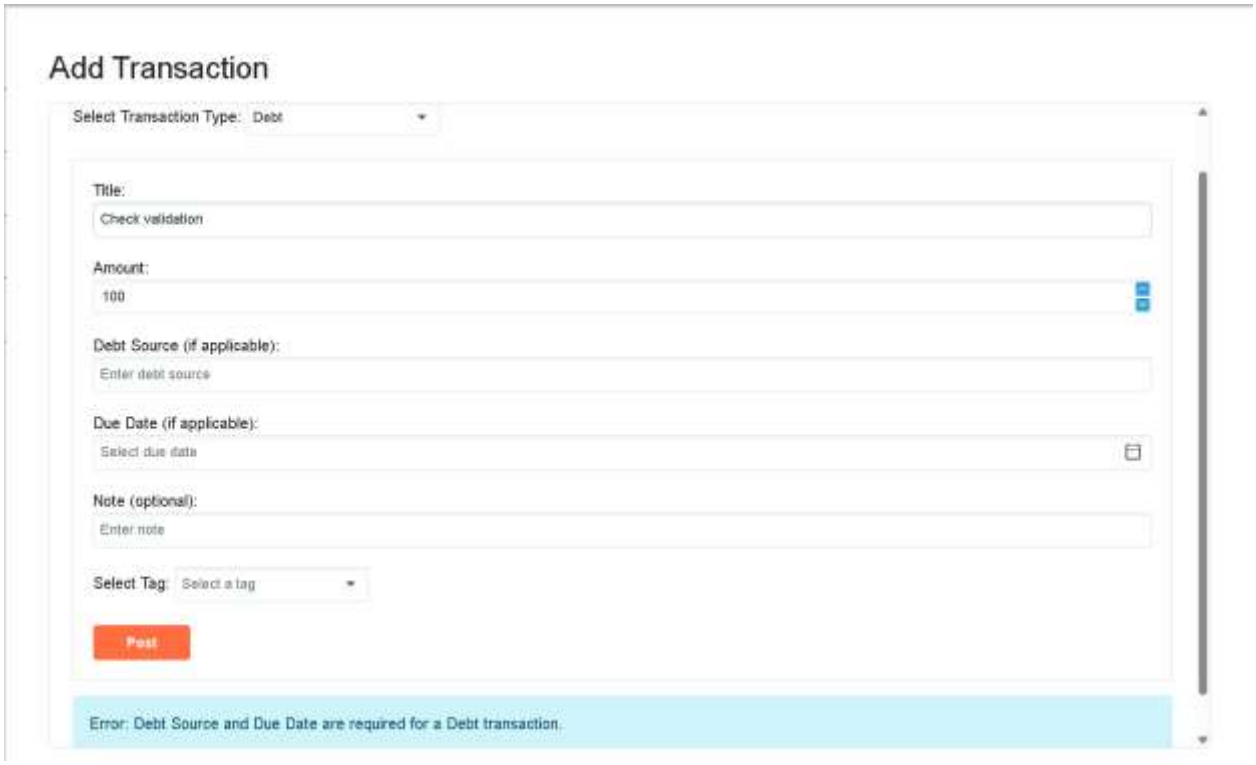


Figure 37: Debt transaction Validation

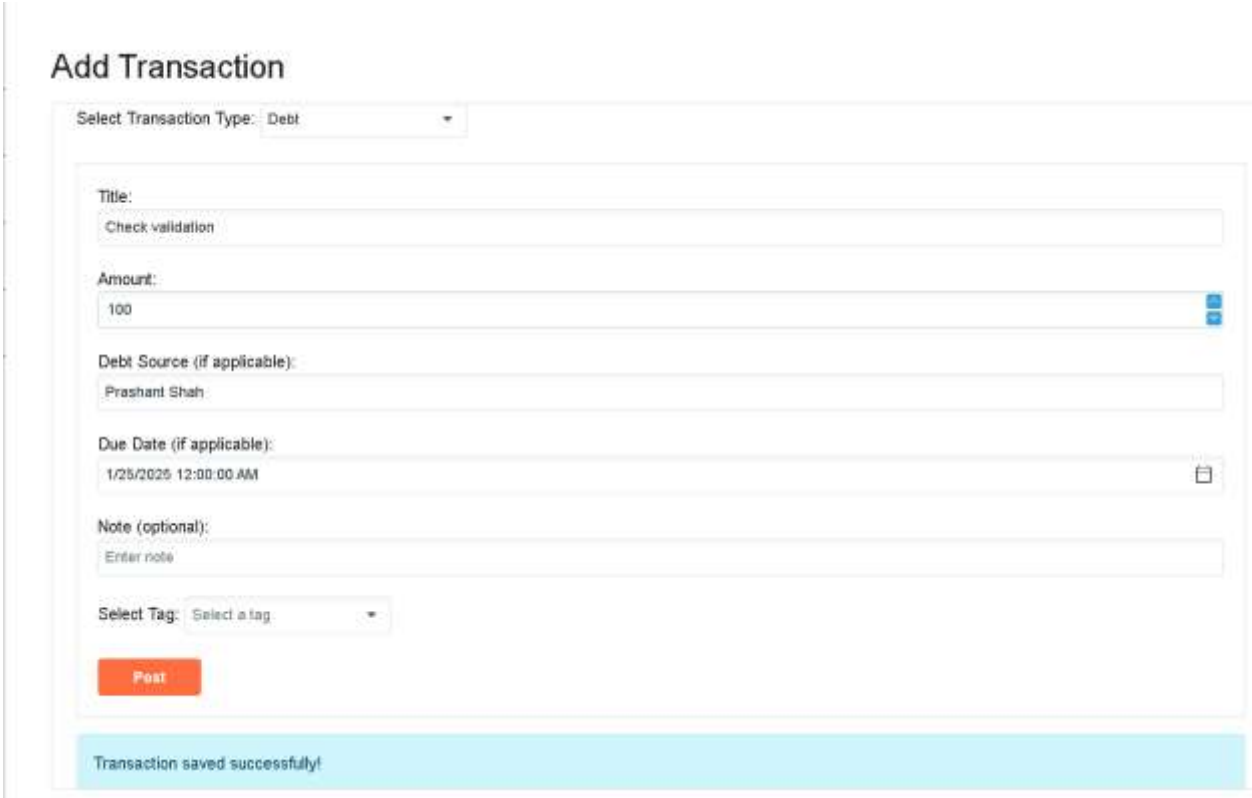


Figure 38: Successful debt transaction

4. Individual Reflection

4.1 Roles and Responsibilities

My core role was to successfully implement and deliver the project in time with all the required functionalities. In addition to balancing other coursework deadlines, I had to effectively manage my time. I had to implement some key technical features, such as password hashing and modular design patterns.

4.2 Personal Insights

The project proved very insightful because I got to learn many concepts in programming that have made my understanding of C#, .NET, and most specifically MAUI so much richer. Working with both Radzen and JSON storage gave insight into abstraction and how it eases data handling.

4.3 Challenges

Given that this was my first time using this language and framework, the task proved to be quite challenging, given the time constraints too. The task was made achievable with daily practice over a month. The own share of issues with JSON parsing and third-party component integration tested my problem-solving capabilities.

4.4 Learnings and Growth

This project completion took my technical skills up a notch. I now fully understand concepts around .NET-especially how interfaces and service-model structures work-and feel confident using Radzen for UI components. It has strengthened my capabilities for complex development environments.

4.5 Impact on Future Work

The project added immense value to my resume. Now, I can develop .NET-based projects independently and create useful applications. It taught me the importance of clean architecture and code reusability, for which my work will be responsible in the future.

4.6 Personal Evaluation

That would be the kind of experience that leads one to massive changes in the skill set of a programmer. All the best features, like Razor syntax, @code{} blocks, dependency injection, and modular design, seemed to give me all the bases for making much larger projects and contributing to the development of teams.

5. Conclusion

The application is designed to be user-friendly, secure, and efficient in managing one's personal expenses. Integration of user registration, a dynamic dashboard with statistics, custom labels on transactions, and an intuitive history page within the application will be extremely valuable for the users. As for the advanced technical practices used in the application, these are Razor syntax, dependency injection, and modular design. Those were my efforts to make it scalable and easy to maintain. All was done in pursuit of the greater goal: to simplify expense tracking without taking so much from the user in terms of experience and data security.

5.1 Implications

This application showed the practical usefulness of the components of Radzen and how third-party libraries can be used to ease the process of developing something. It also proved that using service and model structures helps to keep codes clean and maintainable.

5.2 Recommendations

The future versions of the application will be able to include internet integration, sync with Google Calendar, and save data online through cloud services. Extended use of APIs can be used to extend this application.

5.3 Findings

The components of the Radzen used - charts, tables - reduced development time significantly and are highly functional. Apart from Radzen, I researched about MudBlazor and Winforms to see other alternatives and know where each one is useful.

5.4 Limitations

This application does not contain internet-based features, which limit users from creating expense records offline. Data synchronization in real-time and advanced analytics were not developed in the paper.

5.5 Future Research and Development

Future development may integrate the following: more integrations of applications for extended functionality, real-time data synchronization, and AI/ML-based financial

analysis. Advanced functionality of Radzen and integration with third-party APIs will also be a good addition.

This project wasn't just a serious technological challenge but also a very important learning experience. It laid the foundation for my growth as a programmer, enabling me to work confidently with advanced frameworks and design patterns. The knowledge and skills acquired will definitely influence and improve my contribution to software development projects in the future.