``

CS6004NP

Module Code & Module Title

CS6004NP Application Development

Assessment Type

Milestone Report 2

Semester-

2024/25 Autumn

Student Name: Achyut Tamang

London Met ID: 22068943

College ID: NP04CP4A220011

Assignment Due Date: Thursday, December 12, 2024

Assignment Submission Date: Friday, January 19, 2024

Submitted To:

Mahesh Dhungana

Table of Contents:

Table of Figures:

Github:

I was unable to retrieve your GitHub username to add you as a collaborator on my private repository. If you could kindly provide it, I will grant you access immediately. In the meantime, here is the GitHub repository link: https://github.com/AchyutTamang/HisabMitra_MAUI_PROJECT.git

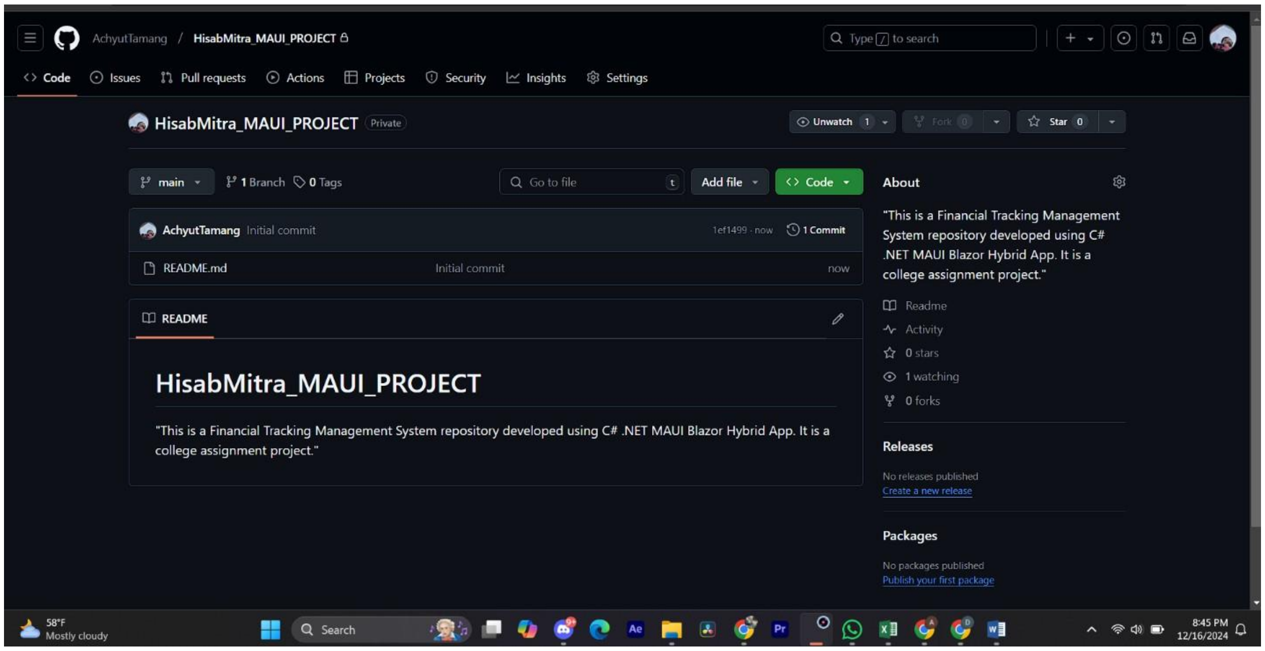
1. Introduction:

The assignment involves developing a financial management system, where the task is to develop a desktop application designed for personal expense tracking. A company has posted the project for on the Pranlancer platform, and due to your strong reputation and positive reviews, I was selected to execute the project within a set deadline. As a C#.NET Developer, I have chosen to use an appropriate .NET Core framework to complete the project. The application will prompt users to enter a username, password, and preferred currency type upon startup. It will track cash inflows (credits, gains, or budgets), cash outflows (debits, spending, or expenses), and debts, with a requirement that sufficient balance be available for outflows. Debt tracking will include the source of the debt, due dates, and ensure that debts can only be cleared with cash inflows, with pending debts clearly highlighted. The system will also allow users to add notes and categorize transactions under labels like credit, debit, or debt, along with customizable tags such as "monthly," "rent," "groceries," "salary," and more. The application will feature search, filter, and sort options by title, transaction type, tags, and date range, along with a dashboard offering detailed statistics on cash inflows, outflows, debts, cleared debts, remaining debts, and the top 5 highest or lowest transactions, along with a pending debt list that can be filtered by date range.

1. Milestone-1:

2.1: Task 1: Initialize private Github repository with an empty MAUI project.

1. Creating an empty private github repository named HisabMitra_MAUI_Project.

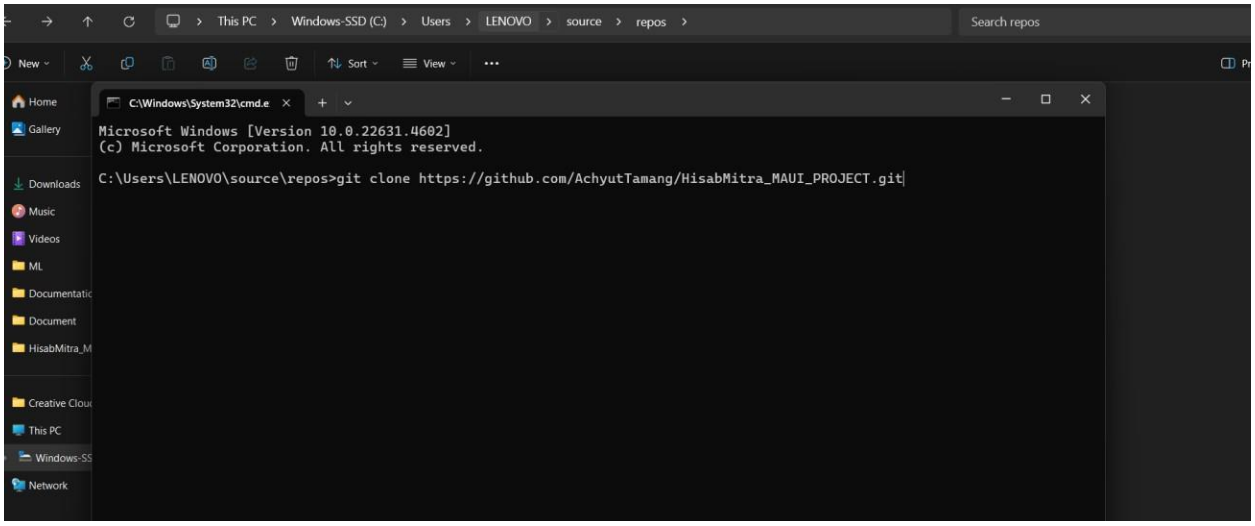


HisabMitra_MAUI_PROJECT

"This is a Financial Tracking Management System repository developed using C# .NET MAUI Blazor Hybrid App. It is a college assignment project."

Figure 1: Creating an empty repo with read me file

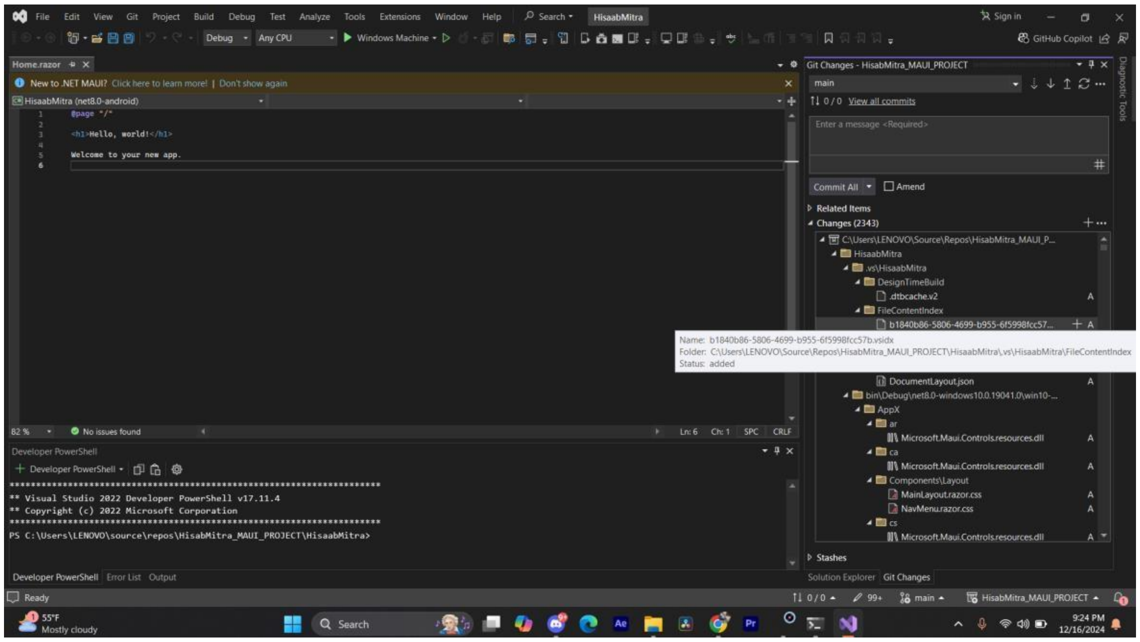
2. Cloning the HisabMitra_MAUI_Project repository inside my C drive.



```
Microsoft Windows [Version 10.0.22631.4602]
(c) Microsoft Corporation. All rights reserved.

C:\Users\LENOVO\source\repos>git clone https://github.com/AchyutTamang/HisabMitra_MAUI_PROJECT.git
```

Figure 2: Cloning repo in destined location

3. Creating an MAUI BLAZOR HYBRID APP named HisaabMitra inside the HisabMitra_MAUI_Project repository.

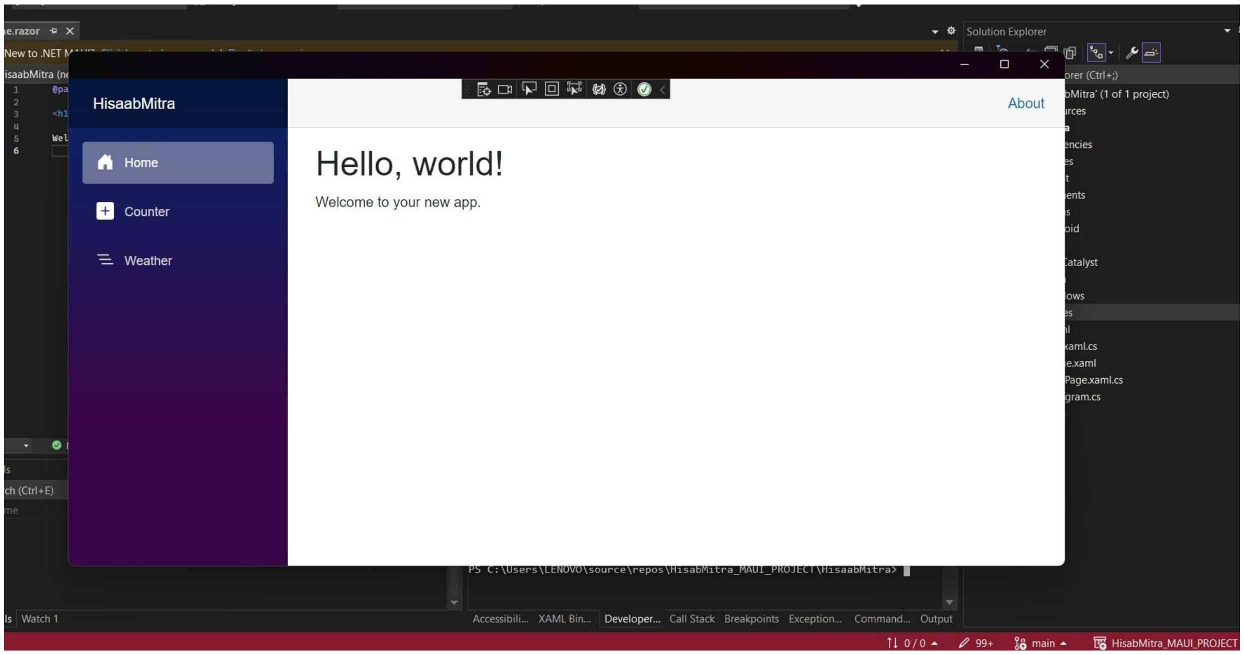


```
@page "/"

<h1>Hello, world!</h1>

Welcome to your new app.
```

```
**********************************************************************
** Visual Studio 2022 Developer PowerShell v17.11.4
** Copyright (c) 2022 Microsoft Corporation
**********************************************************************
PS C:\Users\LENOVO\source\repos\HisabMitra_MAUI_PROJECT\HisaabMitra>
```

Figure 3: Creating an MAUI Blazor Hybrid app named HisaabMitra

4. Successfully Running the HisaabMitra MAUI BLAZOR HYBRID APP



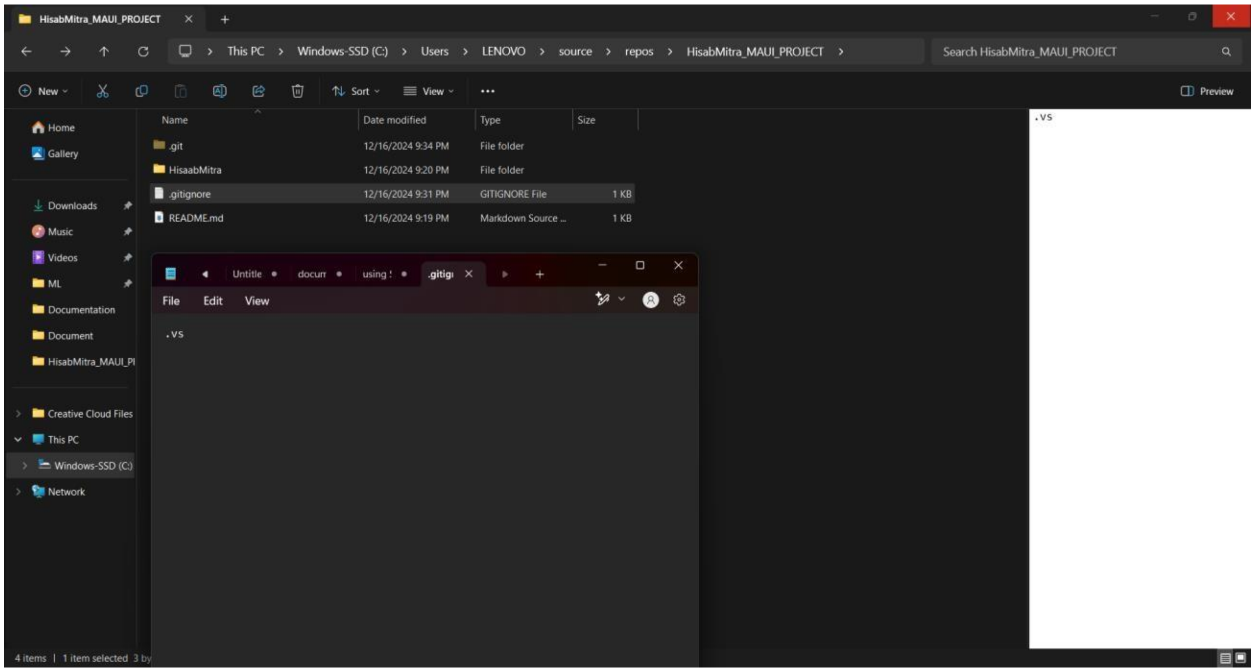
Hello, world!

Welcome to your new app.

```
PS C:\Users\LENOVO\source\repos\HisaabMitra_MAUI_PROJECT\HisaabMitra>
```

Figure 4: successfully running the app

5. Creating an gitignore file to ignore the unwanted folder named vs.

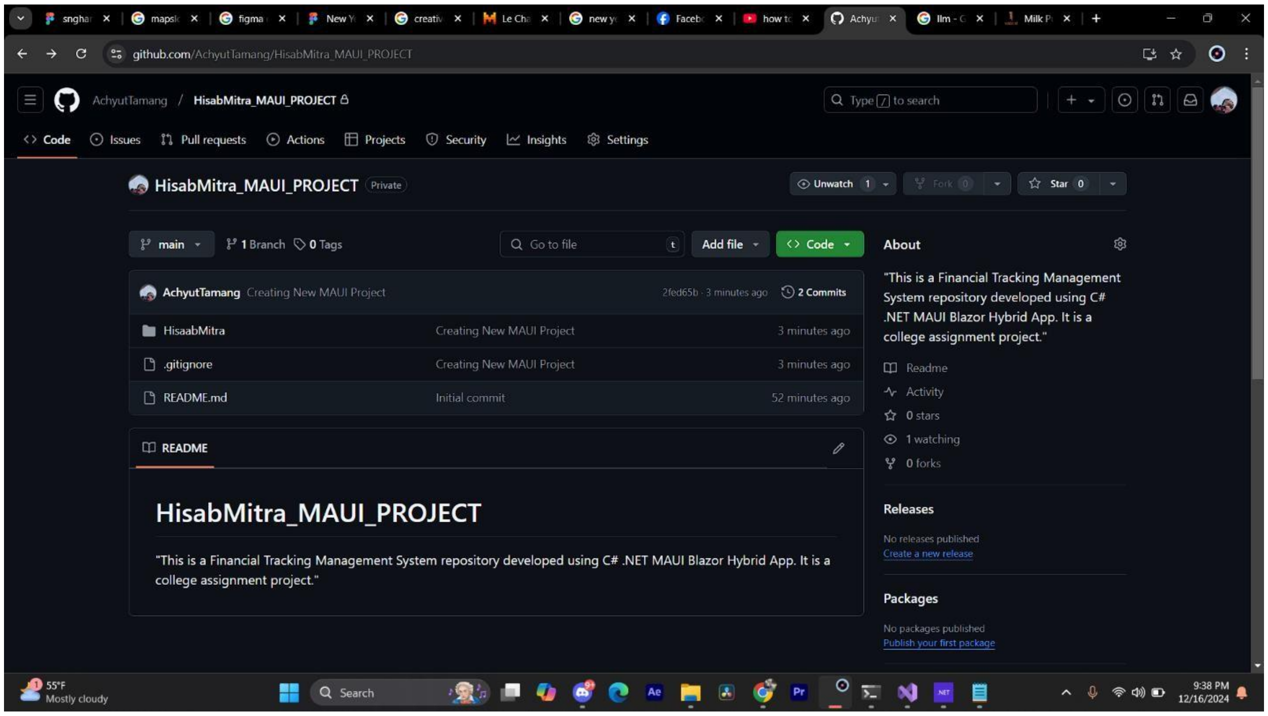


Name	Date modified	Type	Size
.git	12/16/2024 9:34 PM	File folder	
HisaabMitra	12/16/2024 9:20 PM	File folder	
.gitignore	12/16/2024 9:31 PM	GITIGNORE File	1 KB
README.md	12/16/2024 9:19 PM	Markdown Source ...	1 KB

```
.vs
```

Figure 5: add an .gitignore file to ignore unwanted files

6. Pushing the HisaabMitra Project in Github


github.com/AchyutTamang/HisabMitra_MAUI_PROJECT

AchyutTamang / HisabMitra_MAUI_PROJECT

Code · Issues · Pull requests · Actions · Projects · Security · Insights · Settings

HisabMitra_MAUI_PROJECT Private

main · 1 Branch · 0 Tags

AchyutTamang Creating New MAUI Project 2fed65b · 3 minutes ago 2 Commits

HisaabMitra	Creating New MAUI Project	3 minutes ago
.gitignore	Creating New MAUI Project	3 minutes ago
README.md	Initial commit	52 minutes ago

About

"This is a Financial Tracking Management System repository developed using C# .NET MAUI Blazor Hybrid App. It is a college assignment project."

Readme · Activity · 0 stars · 1 watching · 0 forks

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

README

HisabMitra_MAUI_PROJECT

"This is a Financial Tracking Management System repository developed using C# .NET MAUI Blazor Hybrid App. It is a college assignment project."

Figure 6: Pushinf the MAUI app in Github

2.2: Project Report:

I have named the project HisaabMitra, which means 'Account Friend.' It represents a helpful digital tool for managing finances with the slogan, 'Paisa Ko Digital Sathi' (Your Digital Friend for Money), highlighting the ideas of a trusted digital partner that makes managing money simpler.

a) UI Design (Wireframe):

For this project, HisaabMitra, I initially created wireframes using Balsamiq to outline and structure all the functionalities of the application. These wireframes serve as a foundational blueprint, ensuring that the core features and user flows are well-planned. Once the wireframing stage is complete, I plan to further refine and develop a more detailed and visually appealing user interface (UI) design using Figma. This step will provide a clear, polished, and interactive representation of the app's final look and feel, bridging the gap between the initial concept and the final implementation.

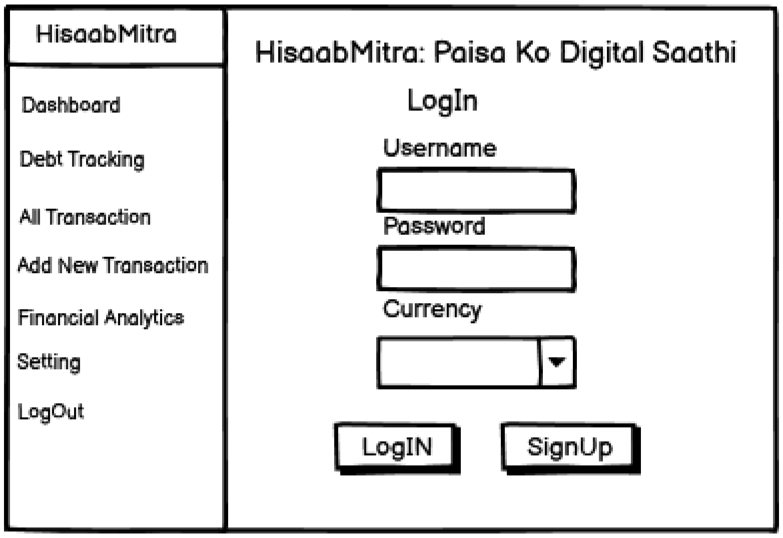


Figure 7: Wire frame of Login

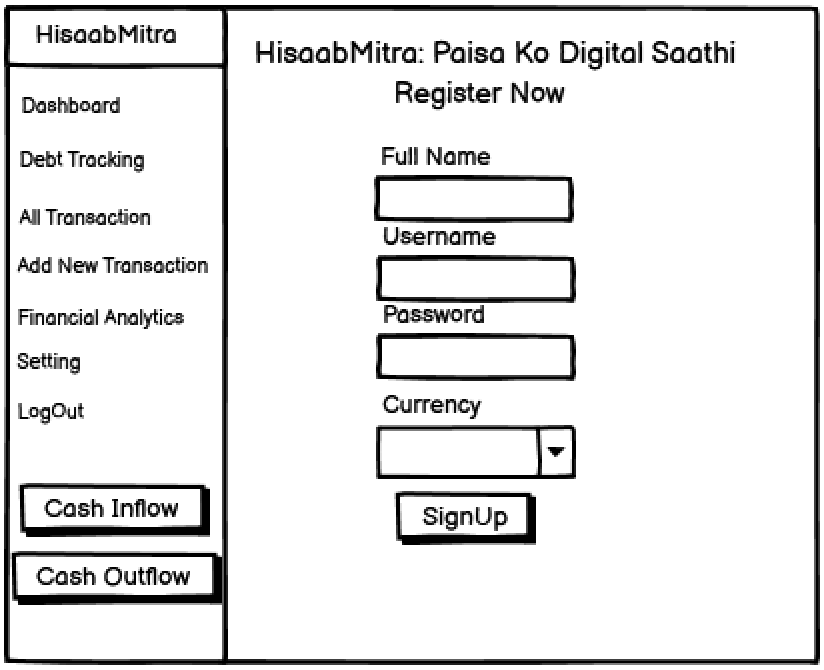


Figure 8: Wireframe of Register

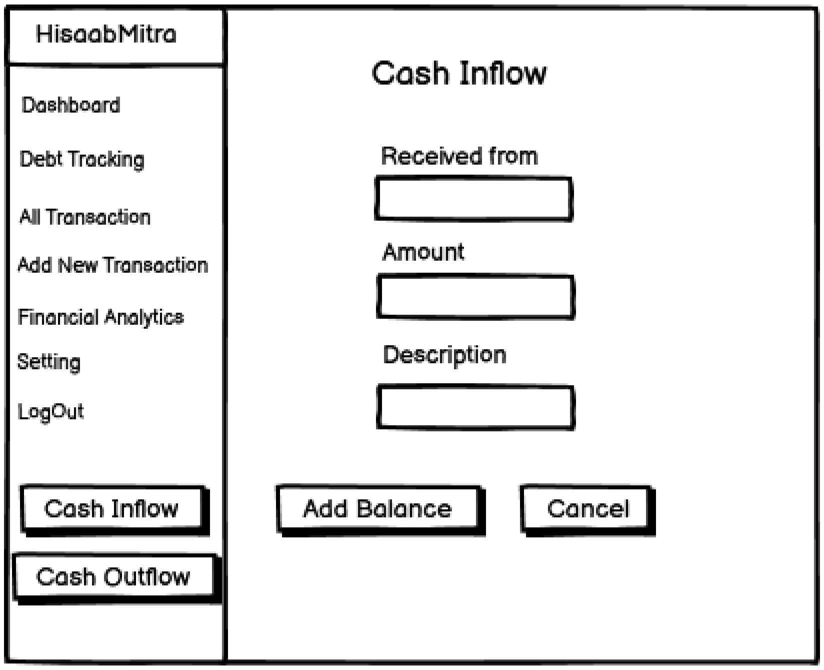


Figure 9: Wireframe of Cash flow

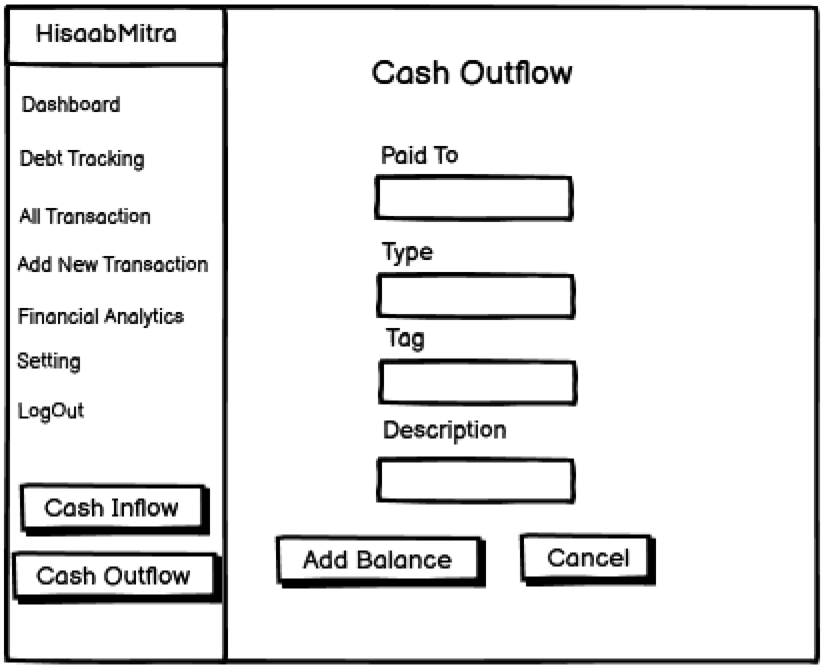


Figure 10: Wireframe of Cash Outflow

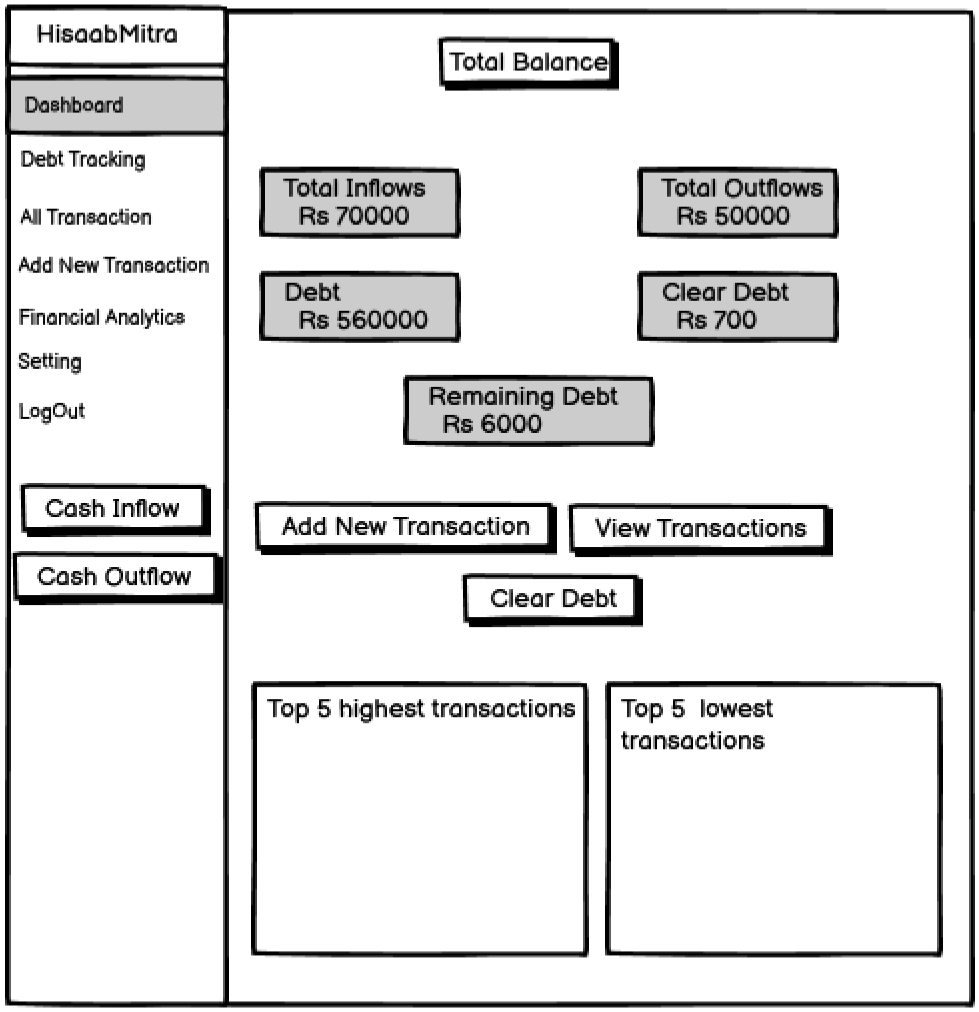
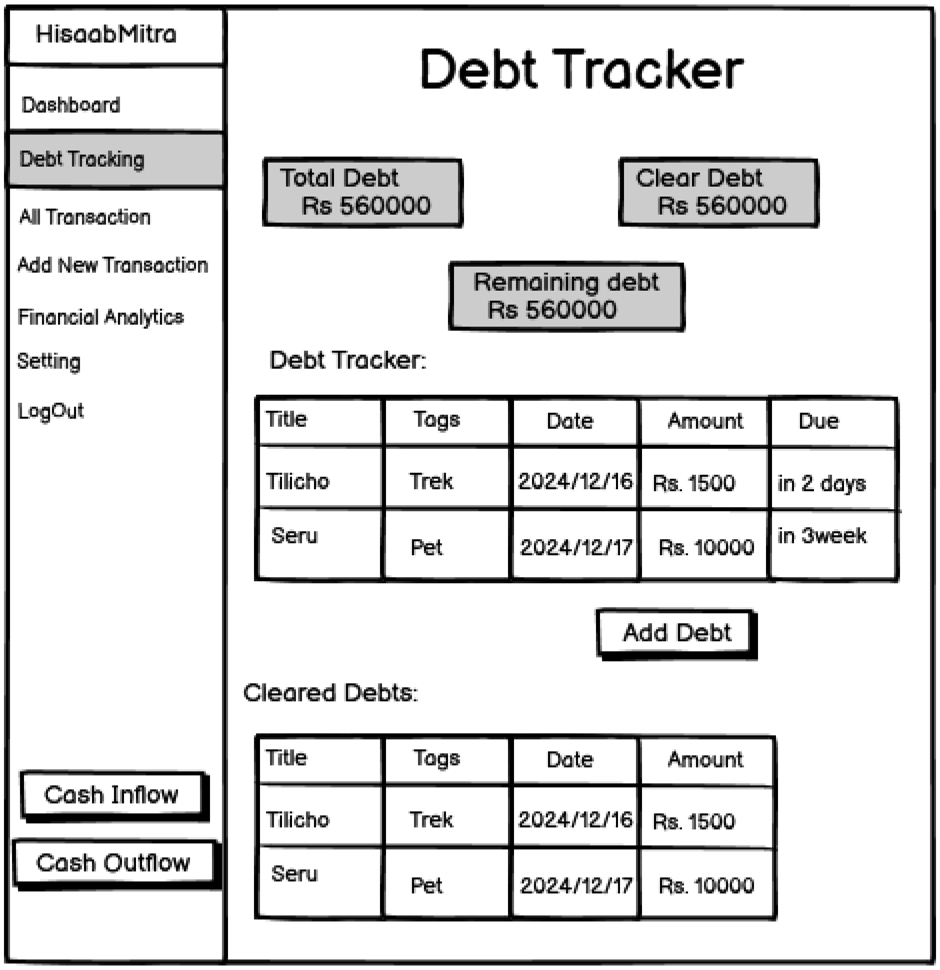


Figure 11: Wireframe of Dashboard



Title	Tags	Date	Amount	Due
Tilicho	Trek	2024/12/16	Rs. 1500	in 2 days
Seru	Pet	2024/12/17	Rs. 10000	in 3week

Title	Tags	Date	Amount
Tilicho	Trek	2024/12/16	Rs. 1500
Seru	Pet	2024/12/17	Rs. 10000

Figure 12: Wireframe of Debttracker

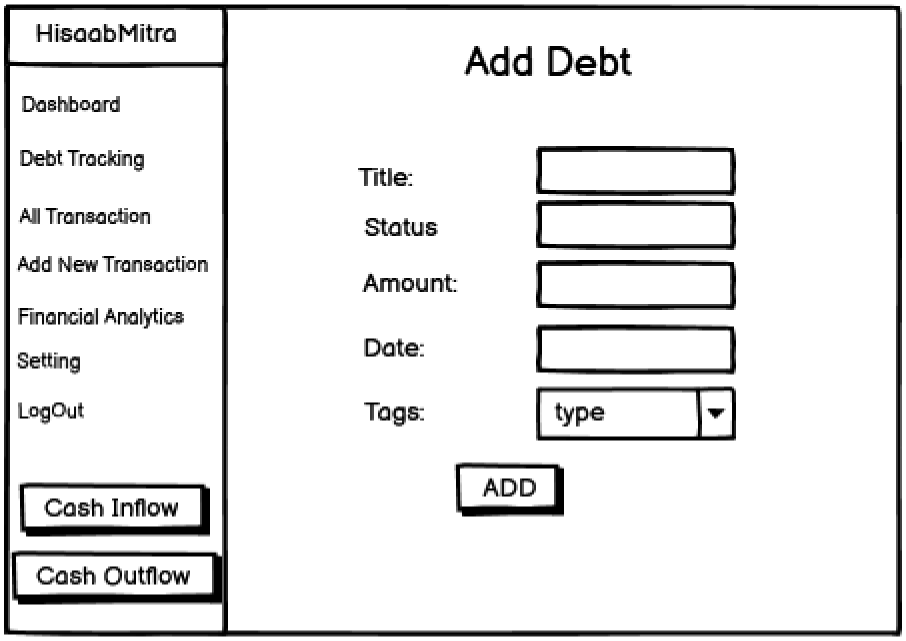
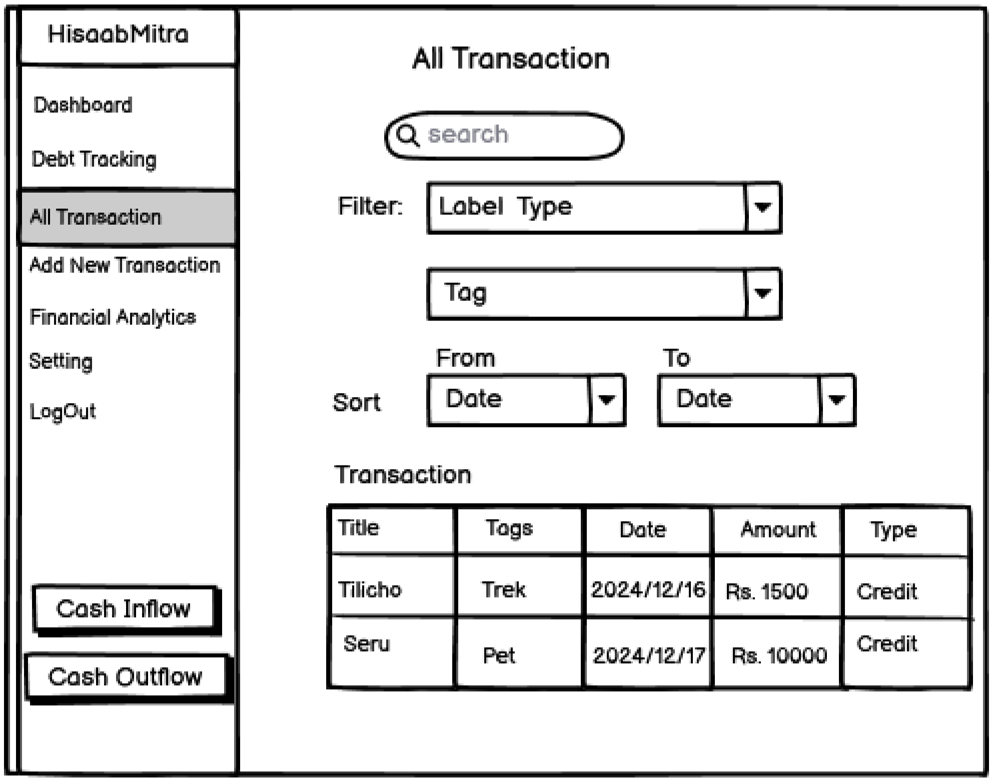


Figure 13: Wireframe of Add Debt



Title	Tags	Date	Amount	Type
Tilicho	Trek	2024/12/16	Rs. 1500	Credit
Seru	Pet	2024/12/17	Rs. 10000	Credit

Figure 14: Wireframe of all transaction

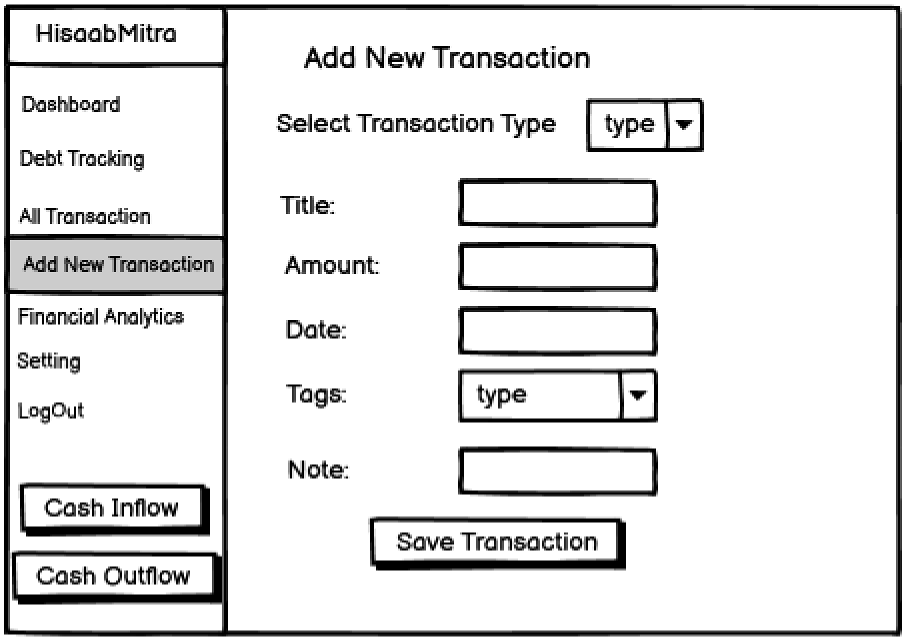
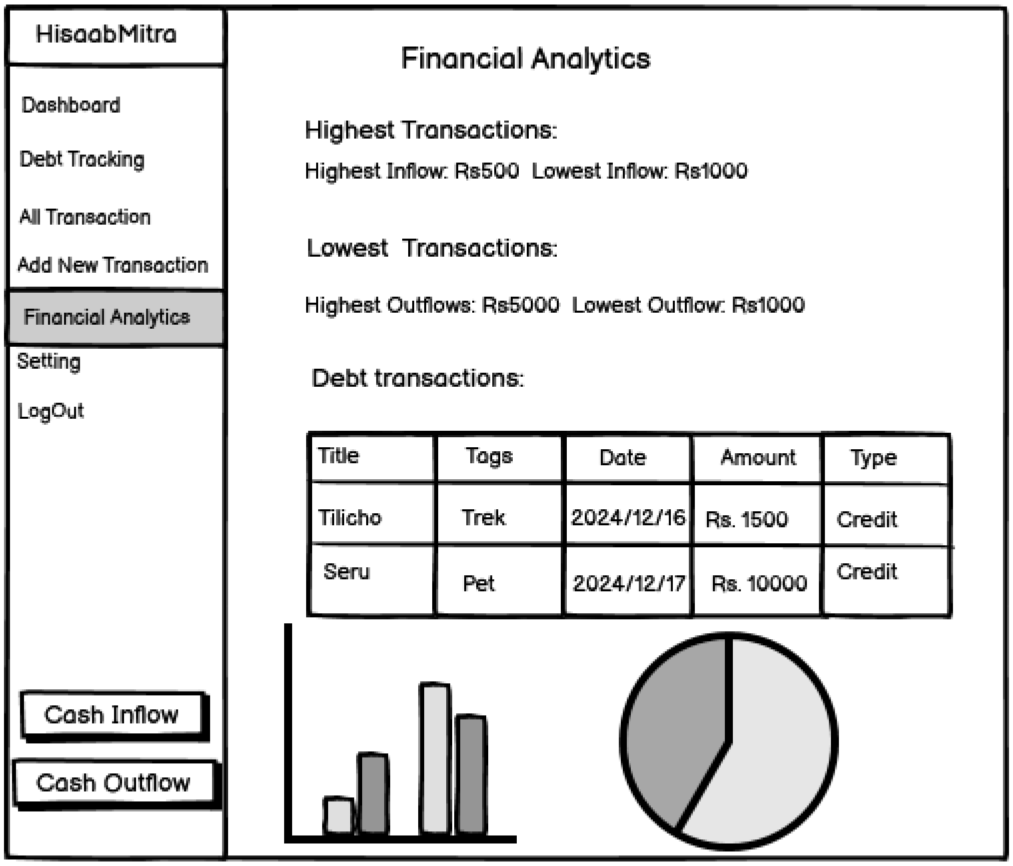


Figure 15: Wireframe of Add new transactio



Title	Tags	Date	Amount	Type
Tilicho	Trek	2024/12/16	Rs. 1500	Credit
Seru	Pet	2024/12/17	Rs. 10000	Credit

Figure 16: Wireframe of financial analytics

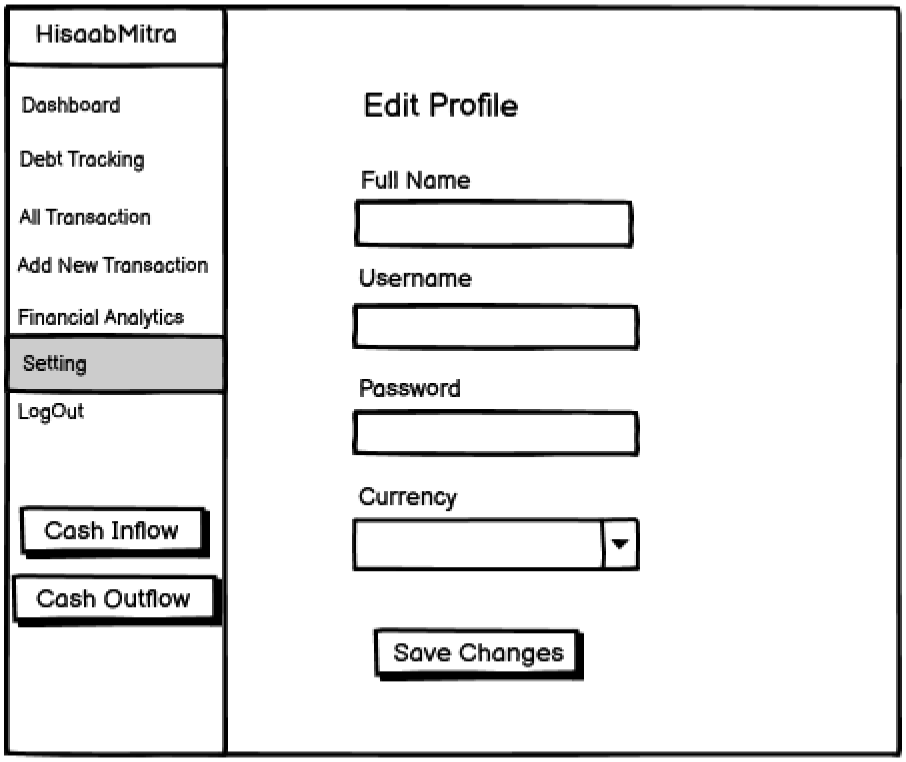


Figure 17: Wireframe of setting

a) Prototype:

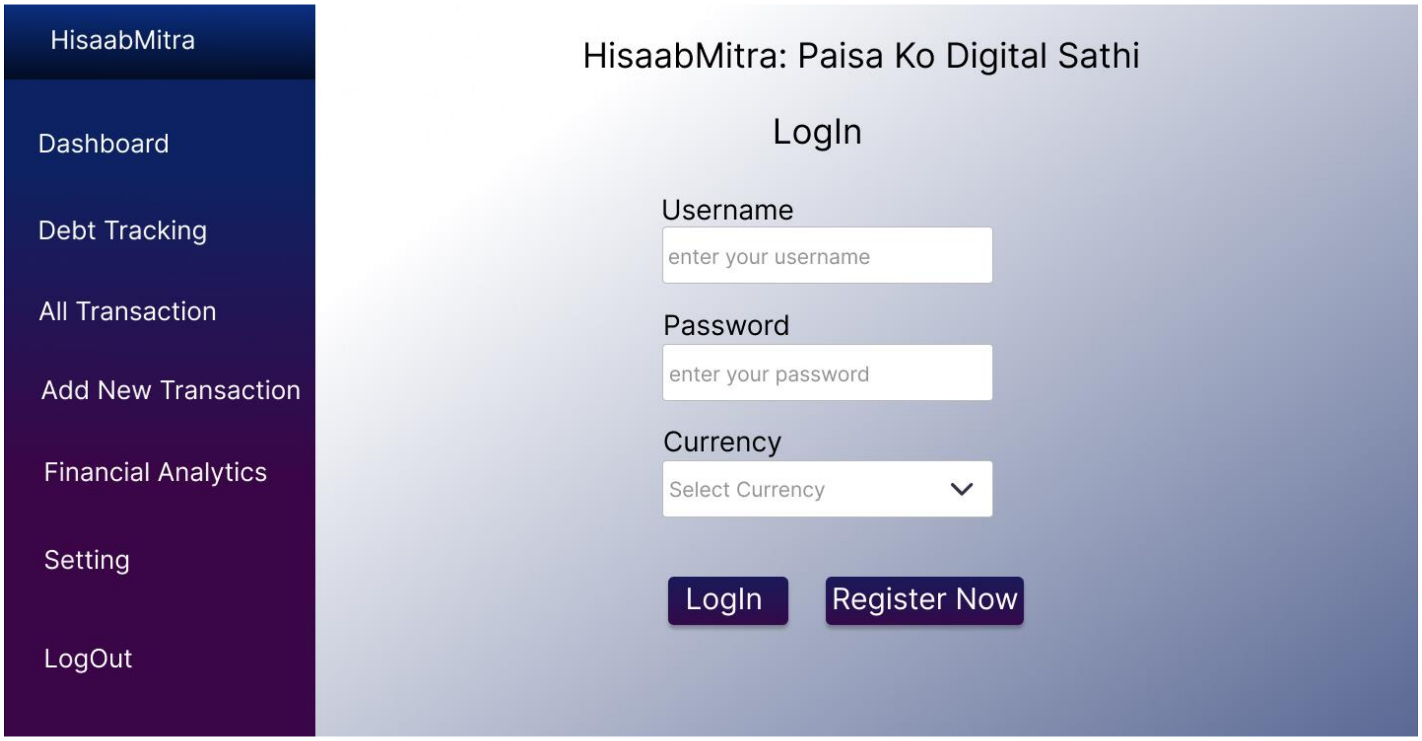


Figure 18: Prototype of Login

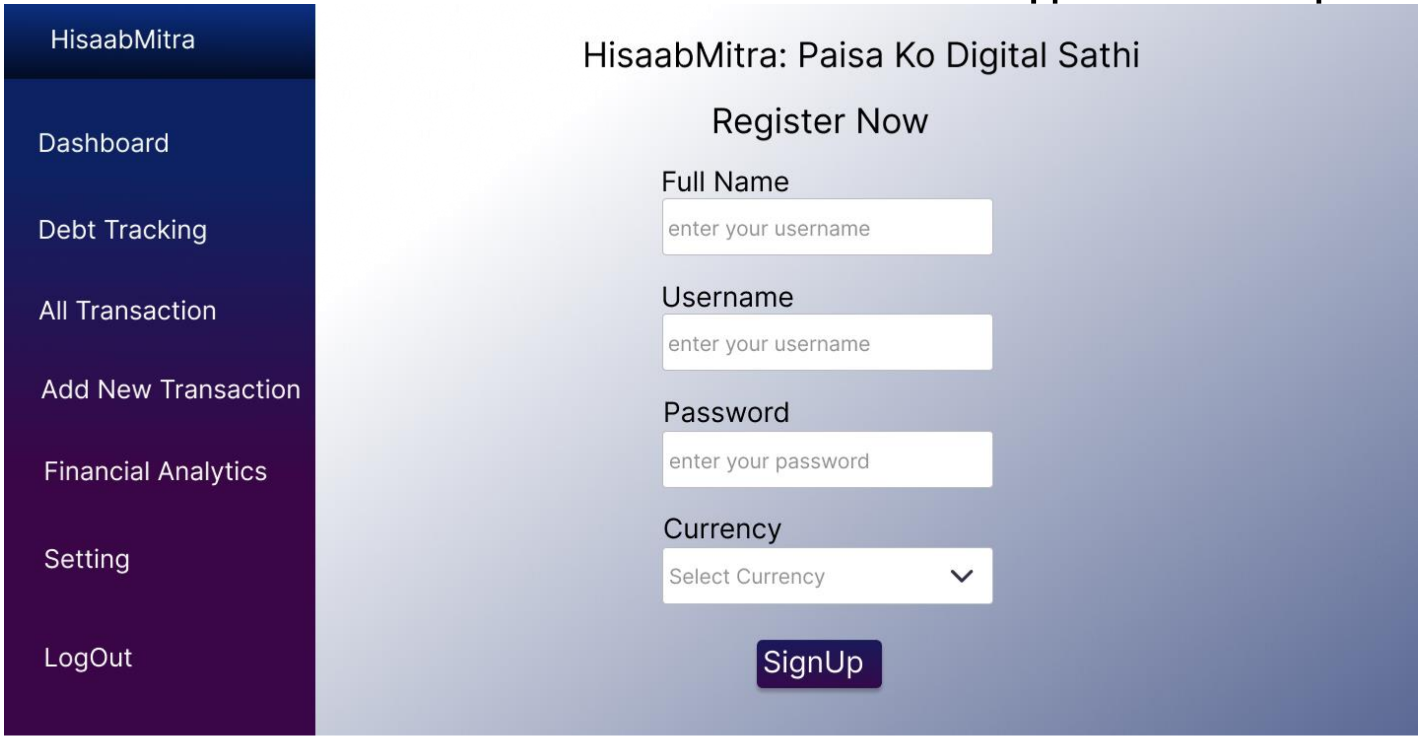


Figure 19: Prototype of register

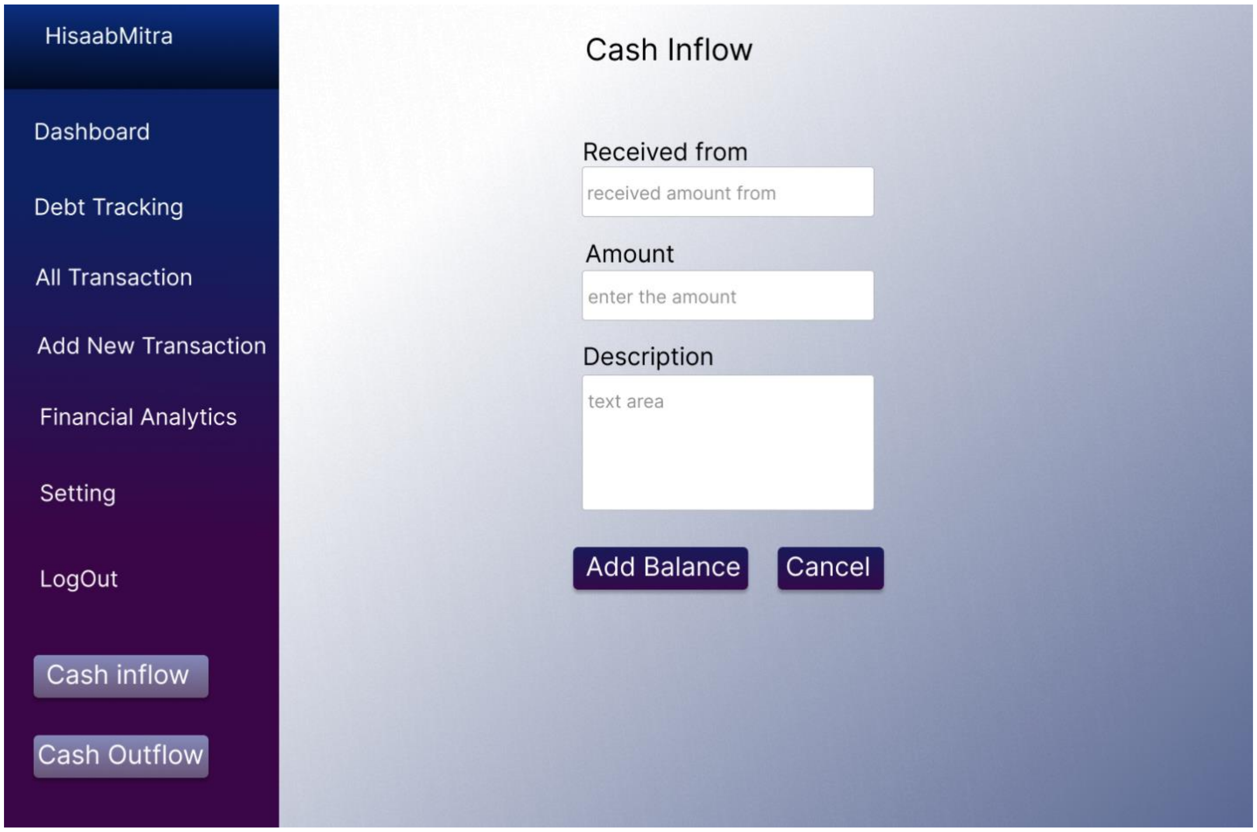


Figure 20: Prototype of Cash Inflow

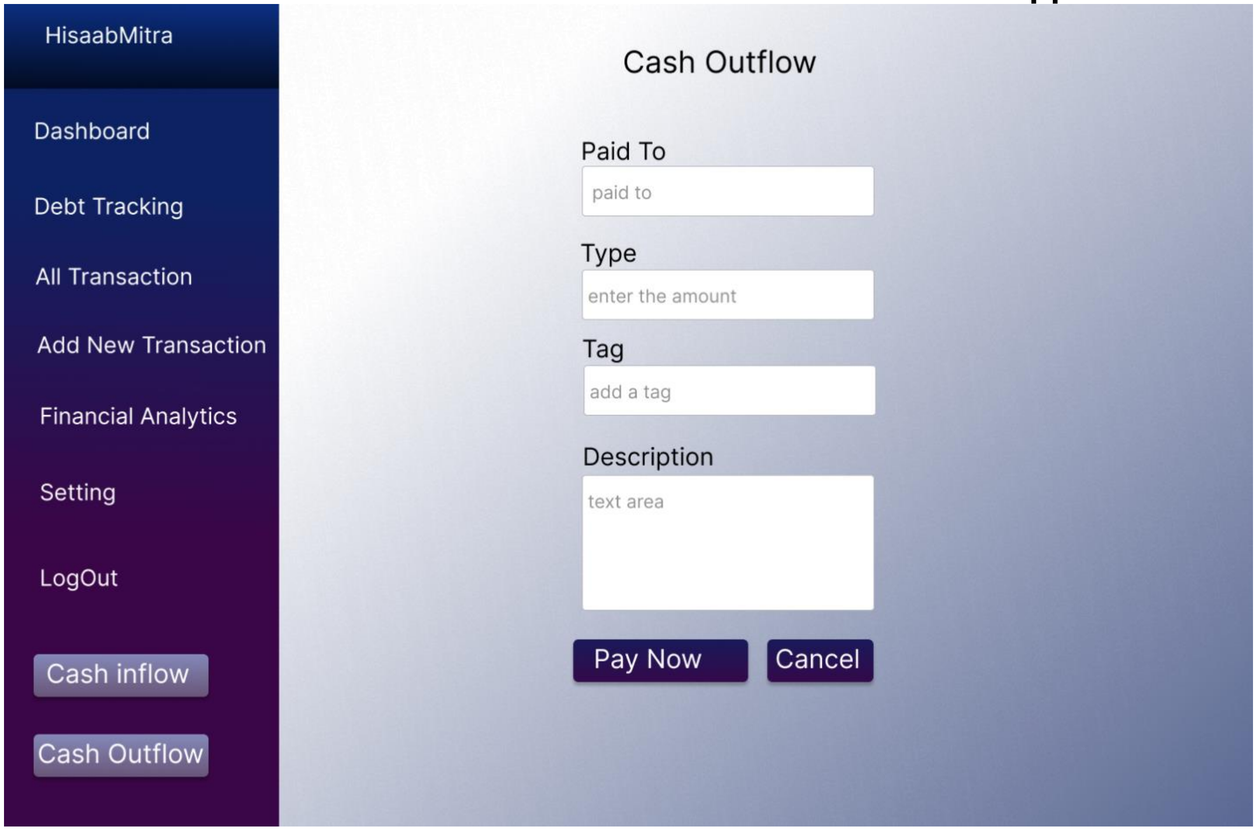
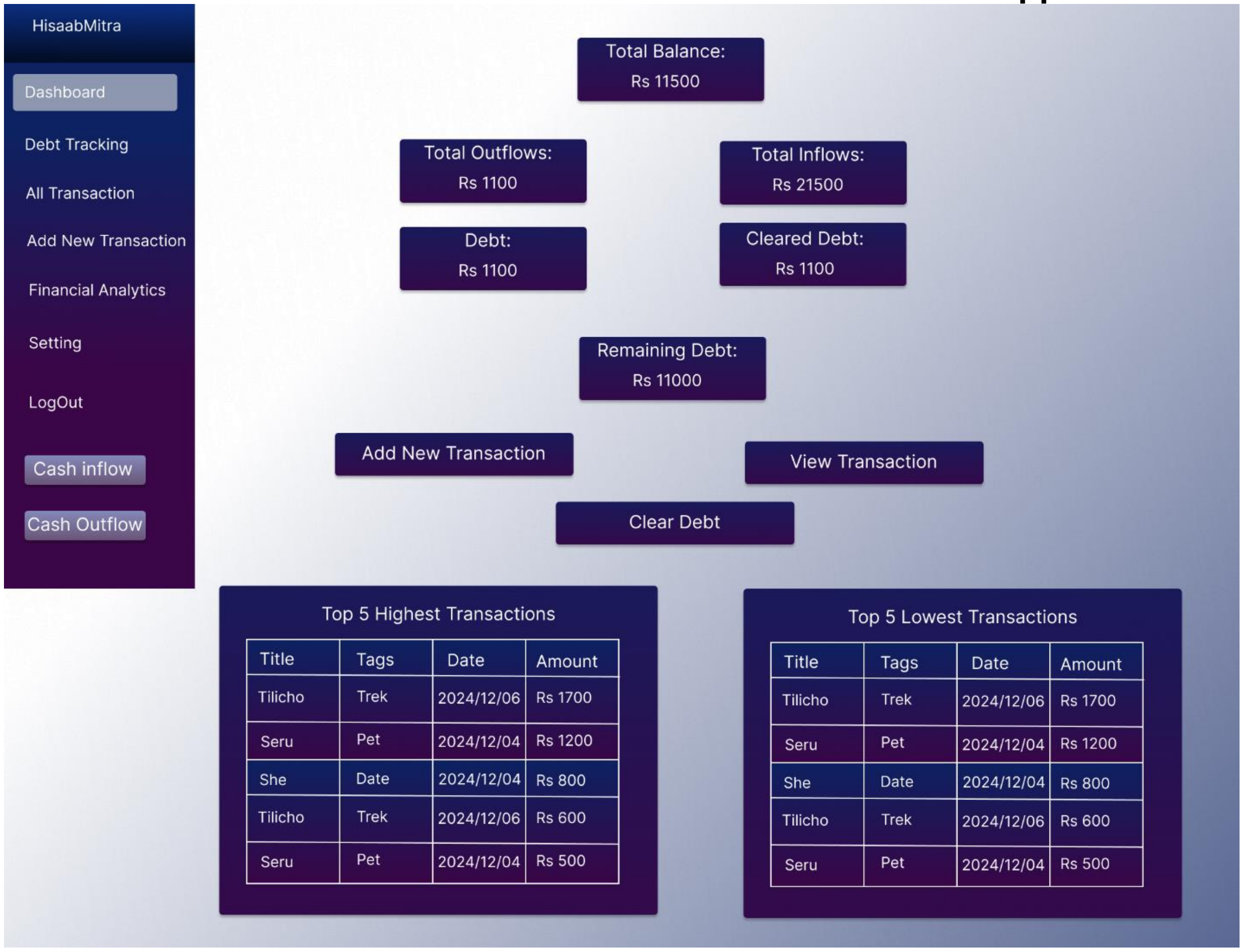


Figure 21: Prototype of cashoutflow



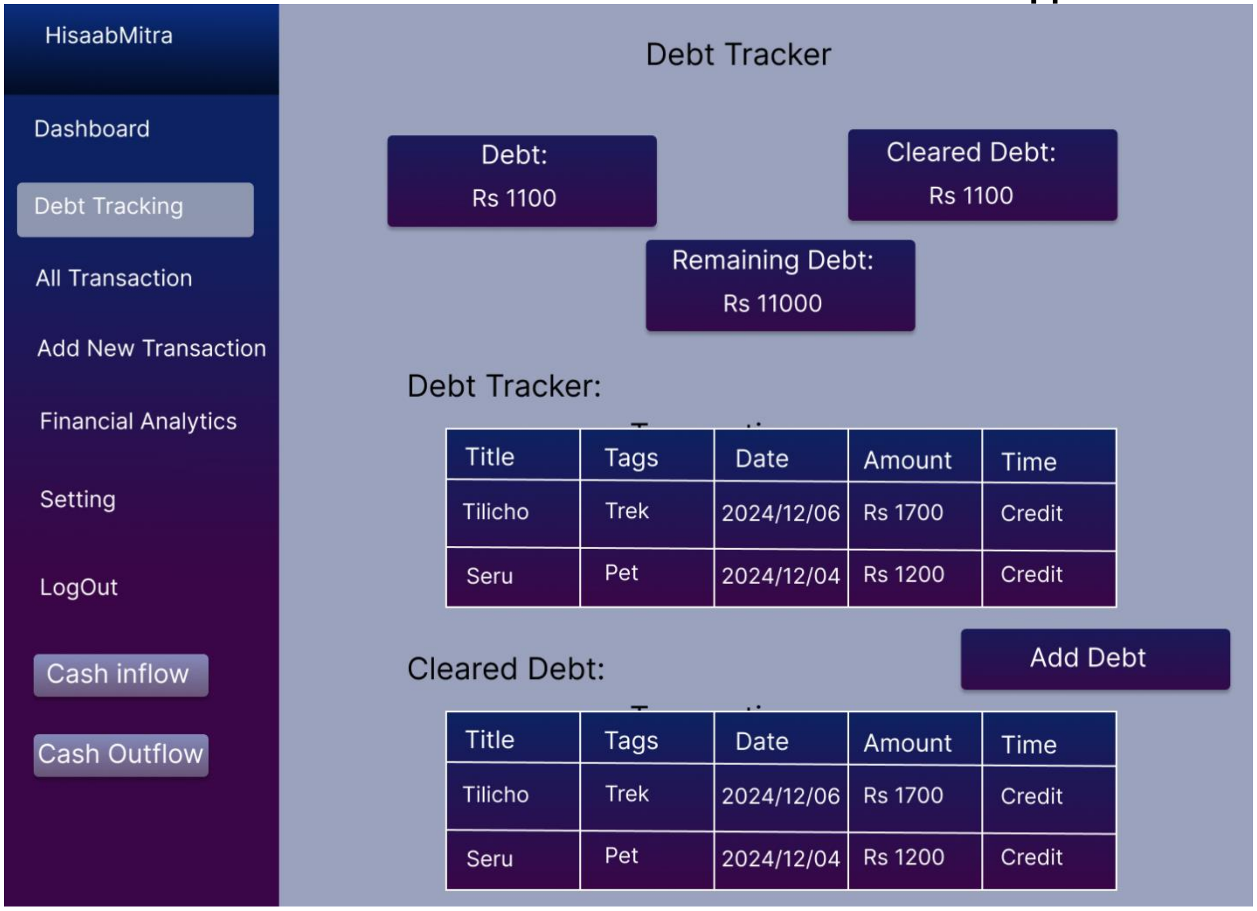
Top 5 Highest Transactions

Title	Tags	Date	Amount
Tilicho	Trek	2024/12/06	Rs 1700
Seru	Pet	2024/12/04	Rs 1200
She	Date	2024/12/04	Rs 800
Tilicho	Trek	2024/12/06	Rs 600
Seru	Pet	2024/12/04	Rs 500

Top 5 Lowest Transactions

Title	Tags	Date	Amount
Tilicho	Trek	2024/12/06	Rs 1700
Seru	Pet	2024/12/04	Rs 1200
She	Date	2024/12/04	Rs 800
Tilicho	Trek	2024/12/06	Rs 600
Seru	Pet	2024/12/04	Rs 500

Figure 22: Prototype of Dashboard



Debt Tracker

Debt: Rs 1100

Cleared Debt: Rs 1100

Remaining Debt: Rs 11000

Debt Tracker:

Title	Tags	Date	Amount	Time
Tilicho	Trek	2024/12/06	Rs 1700	Credit
Seru	Pet	2024/12/04	Rs 1200	Credit

Cleared Debt:

Title	Tags	Date	Amount	Time
Tilicho	Trek	2024/12/06	Rs 1700	Credit
Seru	Pet	2024/12/04	Rs 1200	Credit

Figure 23: Debt Tracker

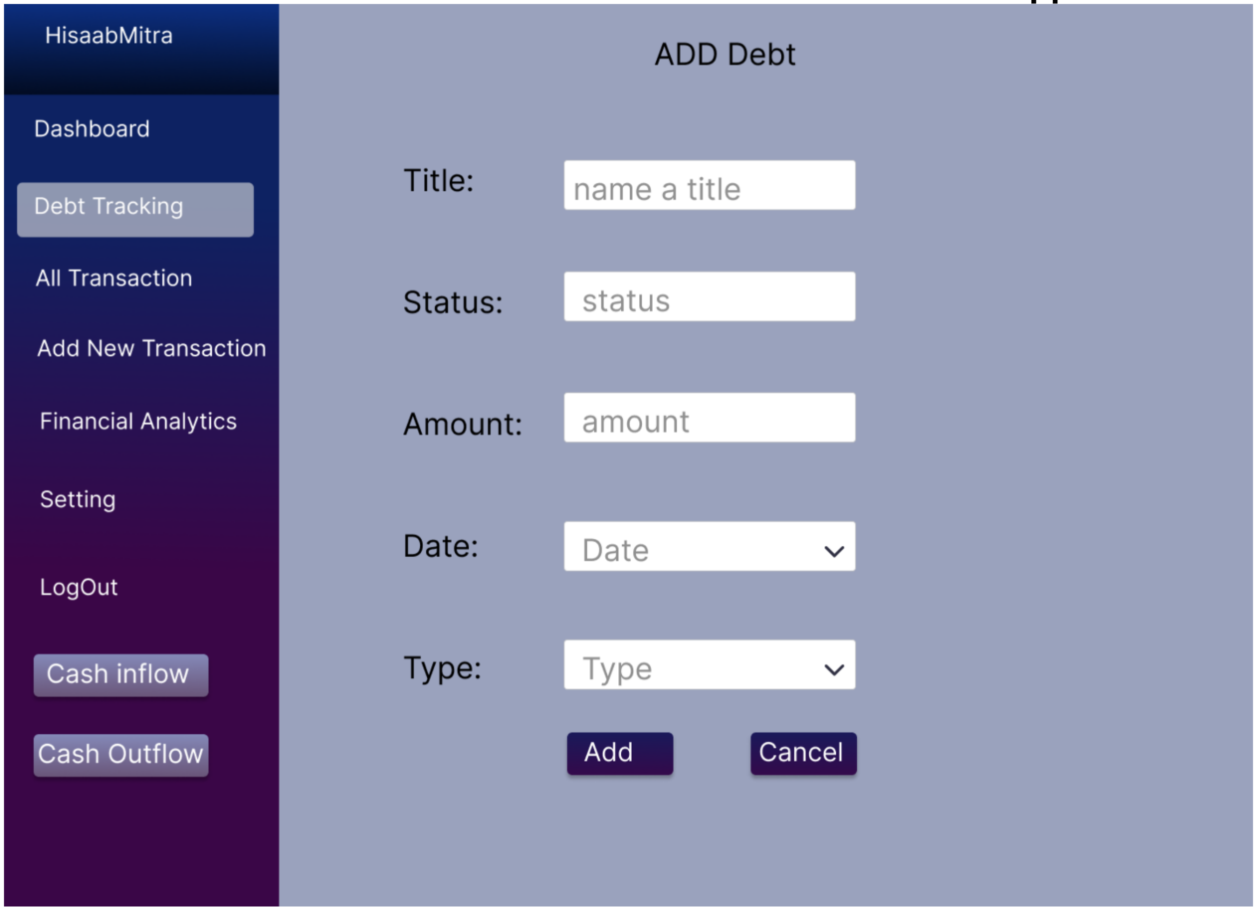
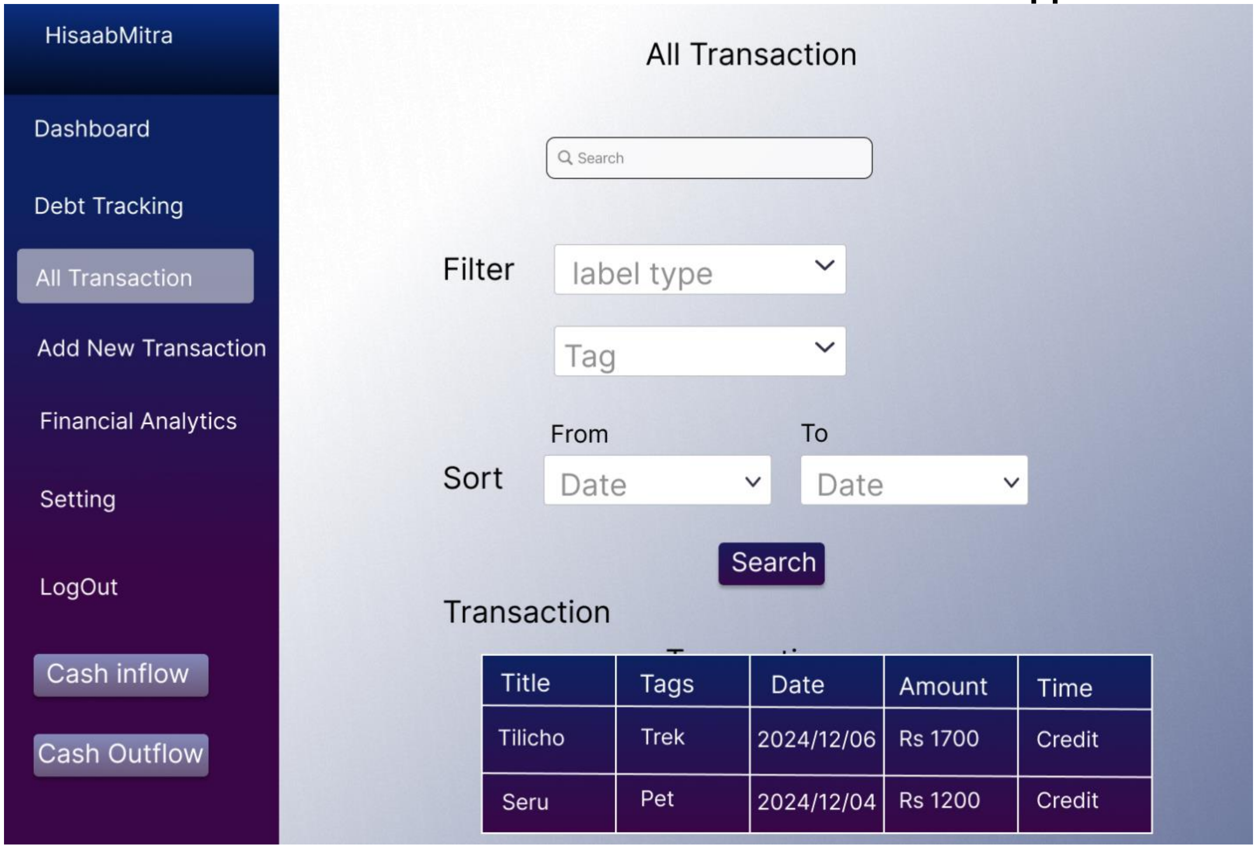
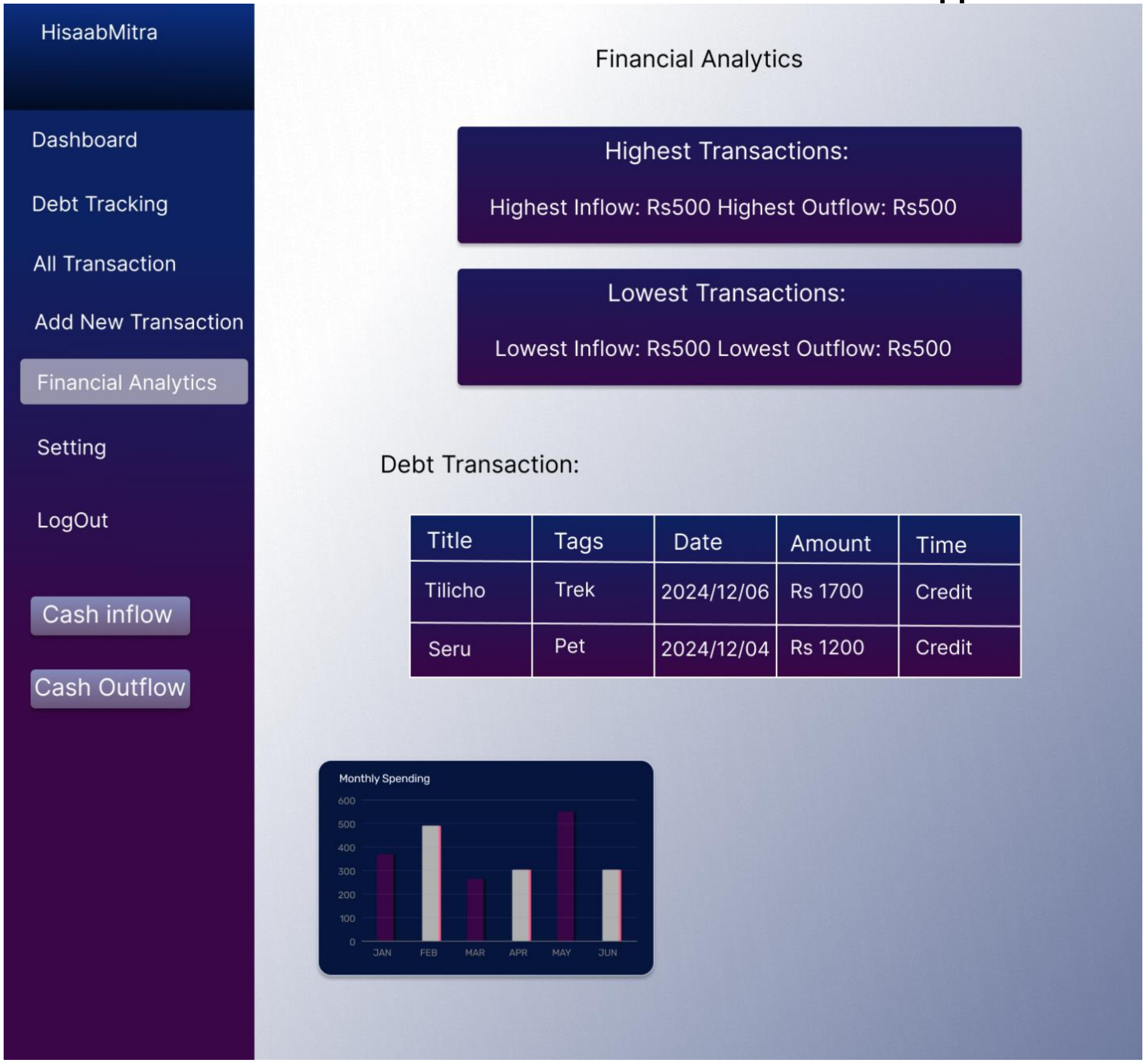


Figure 24: Prototype of Add debt



Title	Tags	Date	Amount	Time
Tilicho	Trek	2024/12/06	Rs 1700	Credit
Seru	Pet	2024/12/04	Rs 1200	Credit

Figure 25: Prototype of All Transaction



Financial Analytics

Highest Transactions:

Highest Inflow: Rs500 Highest Outflow: Rs500

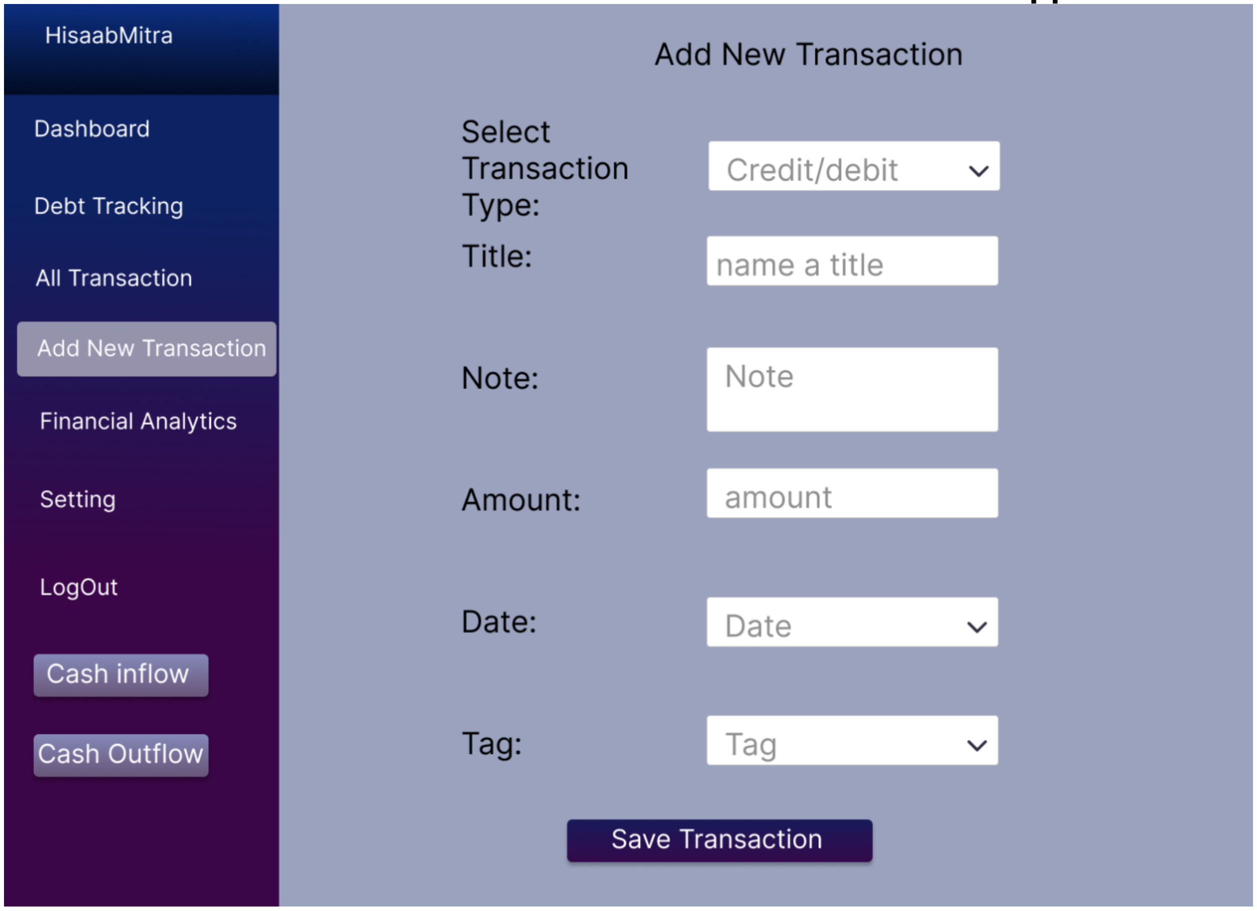
Lowest Transactions:

Lowest Inflow: Rs500 Lowest Outflow: Rs500

Debt Transaction:

Title	Tags	Date	Amount	Time
Tilicho	Trek	2024/12/06	Rs 1700	Credit
Seru	Pet	2024/12/04	Rs 1200	Credit

Figure 26: Prototype of Financial analytics



Add New Transaction

Select Transaction Type:	Credit/debit
Title:	name a title
Note:	Note
Amount:	amount
Date:	Date
Tag:	Tag

Figure 27: Add new transaction

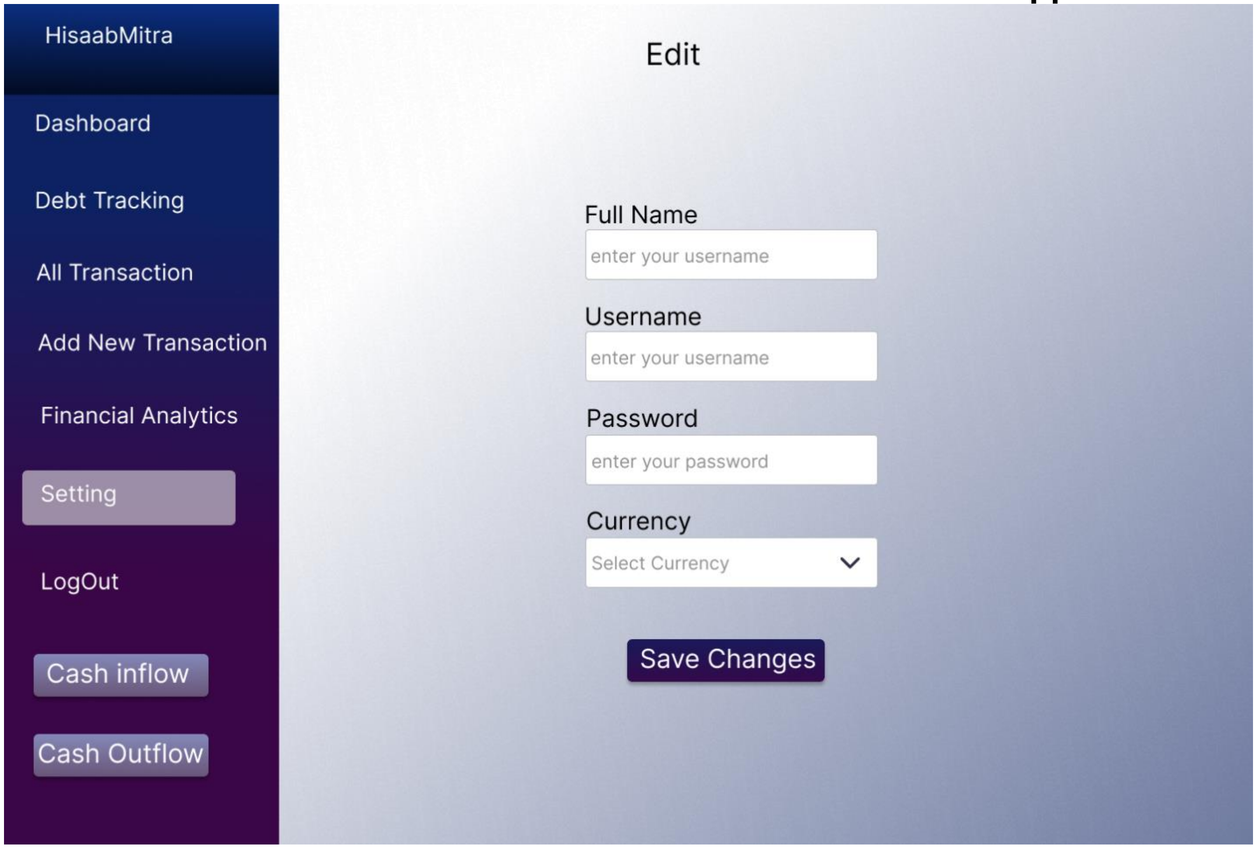


Figure 28: Prototype of setting

b) Data/Entity Modeling:

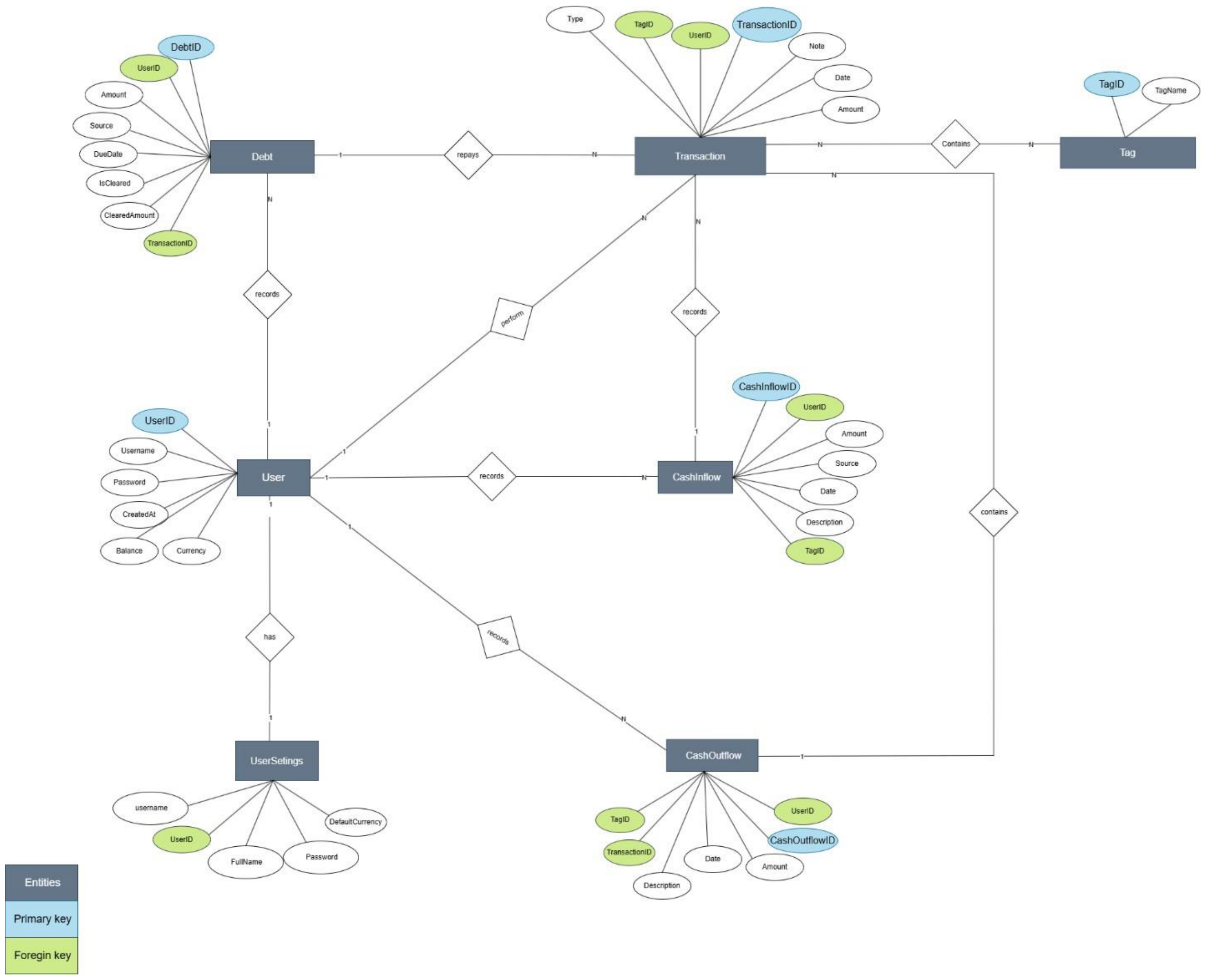


Figure 29: ERD

c) Technology Stack:

1. **Framework:**

Blazor: Blazor is a powerful web framework that allows developers to create interactive web applications using C# and .NET. With its server-side rendering and elimination of the need for JavaScript, it has become a popular choice for web development.

.**NET MAUI:** .NET MAUI is a new framework for building mobile apps on iOS, Android, and Windows using .NET and C#. It allows developers to create one codebase that works across multiple platforms, speeding up development and making maintenance easier. With improvements like better performance, enhanced tools, and support for hot reload, it ensures a consistent user interface across devices using responsive design and platform-specific styling.

.NET MAUI simplifies cross-platform app development, offering a flexible and powerful framework for creating fast, responsive apps, and when paired with Blazor, it enables efficient creation of both web and mobile apps.

.NET MAUI Blazor hybrid apps: .NET MAUI Blazor hybrid apps combine the power of Blazor with the flexibility of .NET MAUI, allowing developers to build both web and mobile apps using a single codebase. This eliminates the need to write separate code for different platforms and enables the creation of responsive, customized user interfaces that work the same across all devices. With features like server-side rendering, improved performance, better tooling, and support for hot reload, these apps simplify the development process while delivering efficient, cross-platform solutions. They represent the future of app development, offering a powerful framework for creating fast, responsive apps.

I have chosen MAUI Blazor Hybrid App for developing this project because it is specifically designed to create modern desktop applications efficiently. Since this is a single project exclusively for a desktop application, MAUI Blazor Hybrid is an ideal choice due to its ability to blend native desktop features with web technologies like HTML and CSS. Here’s how it aligns with the project:

1. Focus on Desktop: MAUI supports native desktop development for platforms like Windows and macOS, ensuring the application is optimized for desktop usage while maintaining a professional and modern look.

2. Single Codebase: Using a single codebase in C# for the entire application reduces development time and complexity. This is particularly advantageous for a one-time, focused project.

3. Blazor Integration: The hybrid approach allows the use of Blazor for building the UI, combining the power of web technologies with native capabilities. This makes it easy to design a user-friendly and interactive desktop application interface.

In summary, MAUI Blazor Hybrid App is an excellent choice for this single, desktop-only project because it combines efficient development practices, modern UI capabilities, and native performance, ensuring the project is delivered with high quality and within a streamlined workflow

2. **External Libraries:**

For this project, I will use the following C# libraries to make development easier and the app more functional and user-friendly:

1. **Newtonsoft.Json:**

   This library is designed to handle JSON data, which is frequently used to exchange information between an app and a server. It simplifies the process by converting JSON data into .NET objects, such as classes, and vice versa. This makes it easier to work with API responses or store app data in a structured JSON format. By using this library, developers can avoid writing complex code for parsing and formatting, streamlining the process of handling data.

   How it will be used:

   - To handle API responses, like fetching or sending data to/from the server.
   - To store app settings or user information locally in JSON files.

2. **Syncfusion:**

   This library offers advanced UI components that make it easier to create attractive and interactive app interfaces. It provides pre-built tools like charts, grids, and calendars, which help display and manage large amounts of data. Additionally, it allows for customization, so the app's design can be tailored to fit the desired look and fee

How it will be used:

- To add data grids and charts for presenting information in a clean and organized way.
- To create dashboards or reports for a better user experience.

3. **CommunityToolkit.Maui:**

This toolkit adds extra features to make the app more flexible and interactive. It includes tools like swipe gestures, pop-ups, and snack bars to improve user interaction. It also makes it easier to create custom layouts and simplifies tasks like adding animations and other design elements.

How it will be used:

- To enable swipe gestures for actions like deleting items or navigating pages.
- To design a unique and responsive user interface.
- To enhance the overall app experience with interactive features.

4. **Persistence Mechanism:**

For this project, I will be using JSON to store and send data because of it's lightweight and easy to read. The data can be saved in a file and opened later when needed. Using JSON in C# is simple with tools like System.Text.Json or Newtonsoft.Json, which help with reading and writing JSON data. This method is great for storing data in a clear and easy-to-understand way

2. Architecture pattern:

While researching architectural patterns for MAUI Blazor hybrid apps, I came across two patterns: MVU (Model-View-Update) and MVVM (Model-View-ViewModel). For this project, HisaabMitra, I will be following the MVU pattern.

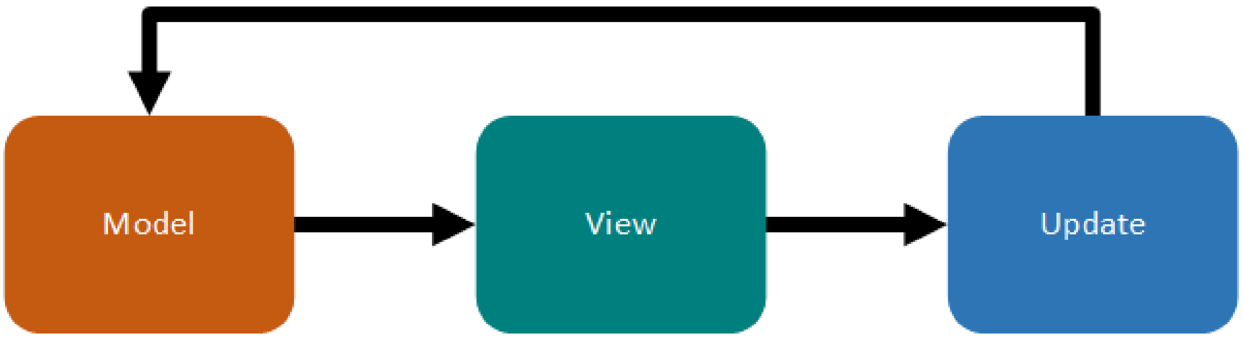


Figure 30: MVU pattern

The Model-View-Update (MVU) pattern, also known as the Elm Architecture, is a design pattern used primarily for building interactive applications. It emphasizes a unidirectional data flow and immutability, making it particularly suitable for functional programming paradigms. MVU has gained traction in C# development, especially with the advent of frameworks like .NET MAUI.

Key Components of MVU

Model: Represents the application's state. It is immutable, meaning any changes to the model result in the creation of a new model instance rather than modifying the existing one.

View: A function that takes the model as input and returns a representation of the UI. The view is pure and always reflects the current state of the model, allowing for straightforward testing and debugging

Update: A function that defines how the model changes in response to user actions or events. It takes a message (an event) and the current model, returning a new model along with any commands that might need to be executed.

Implementing MVU pattern in Login Setup

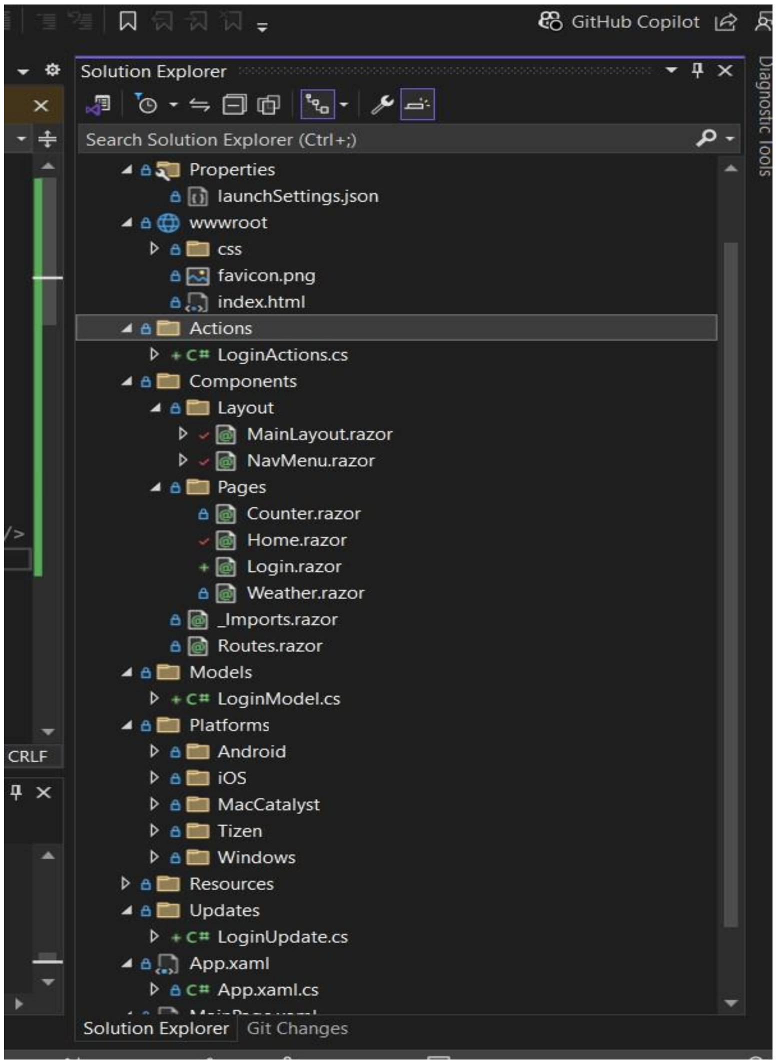


Figure 31: Implementing MVU pattern

For the project, I have created four folders to implement the MVU (Model-Update-View) pattern, which helps maintain a clean and structured architecture. These folders include Models, Pages (UI), Updates, and Actions. The Model folder contains the application's state, represented by classes that store essential data such as user information and app settings. The Pages (UI) folder holds Blazor pages or components responsible for rendering the UI based on the current state and triggering actions that modify the state. The Updates folder contains the logic for state changes, where functions take the current state and an action, returning the updated state. Finally, the Actions folder defines the actions that initiate state changes, such as user interactions or events. This organization ensures a clear separation of concerns: the Models holds the state, the Update function processes actions and updates the state, and the Pages render the view while dispatching actions. This approach enhances maintainability, testability, and scalability within the application.

3. Milestone-2:

Output:

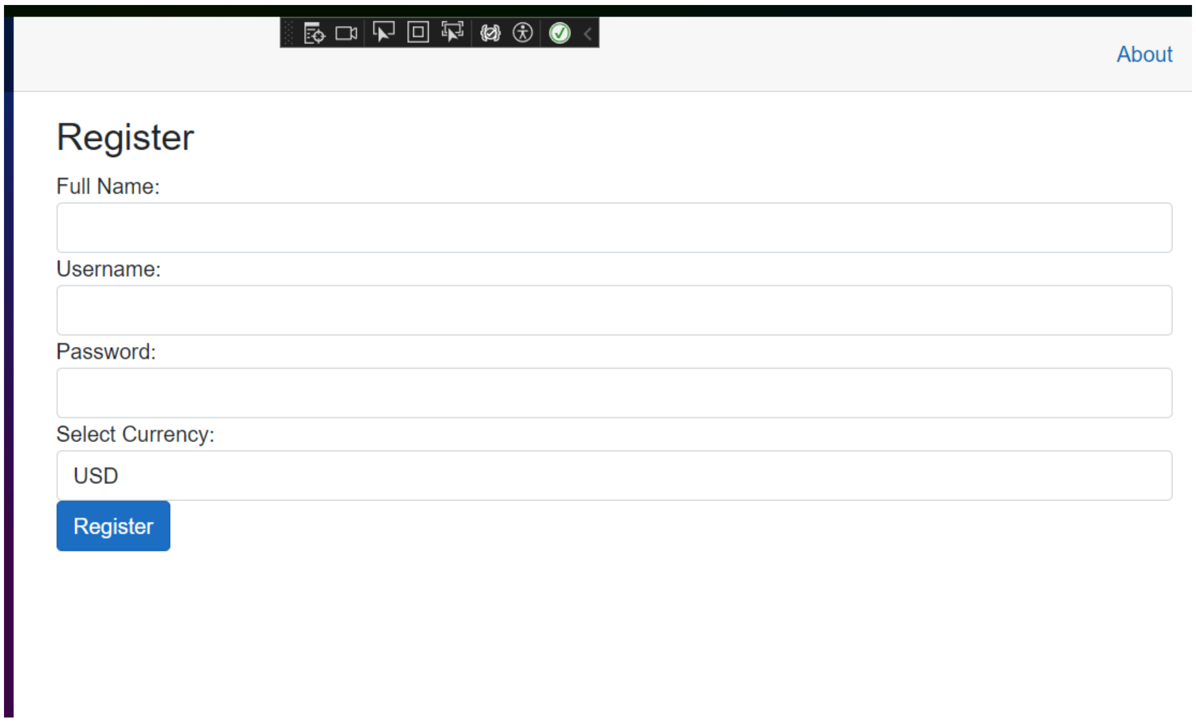
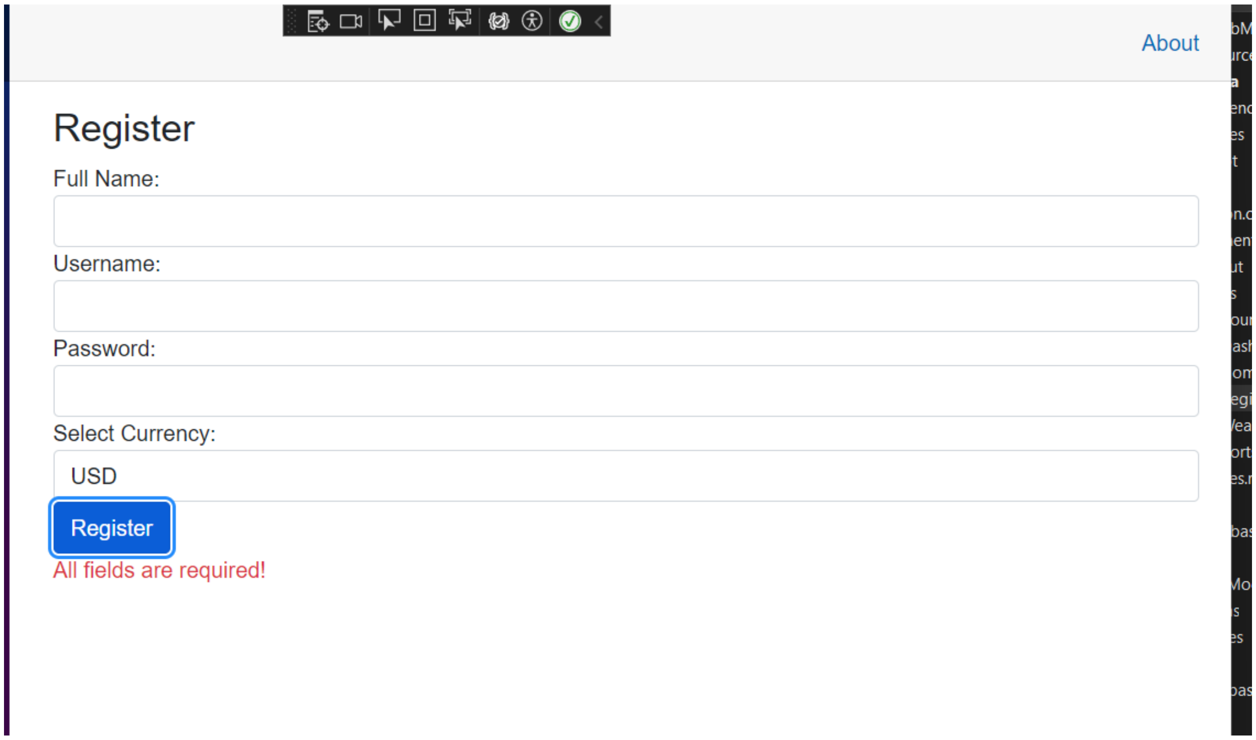


Figure 32: Register page



Register

Full Name:

Username:

Password:

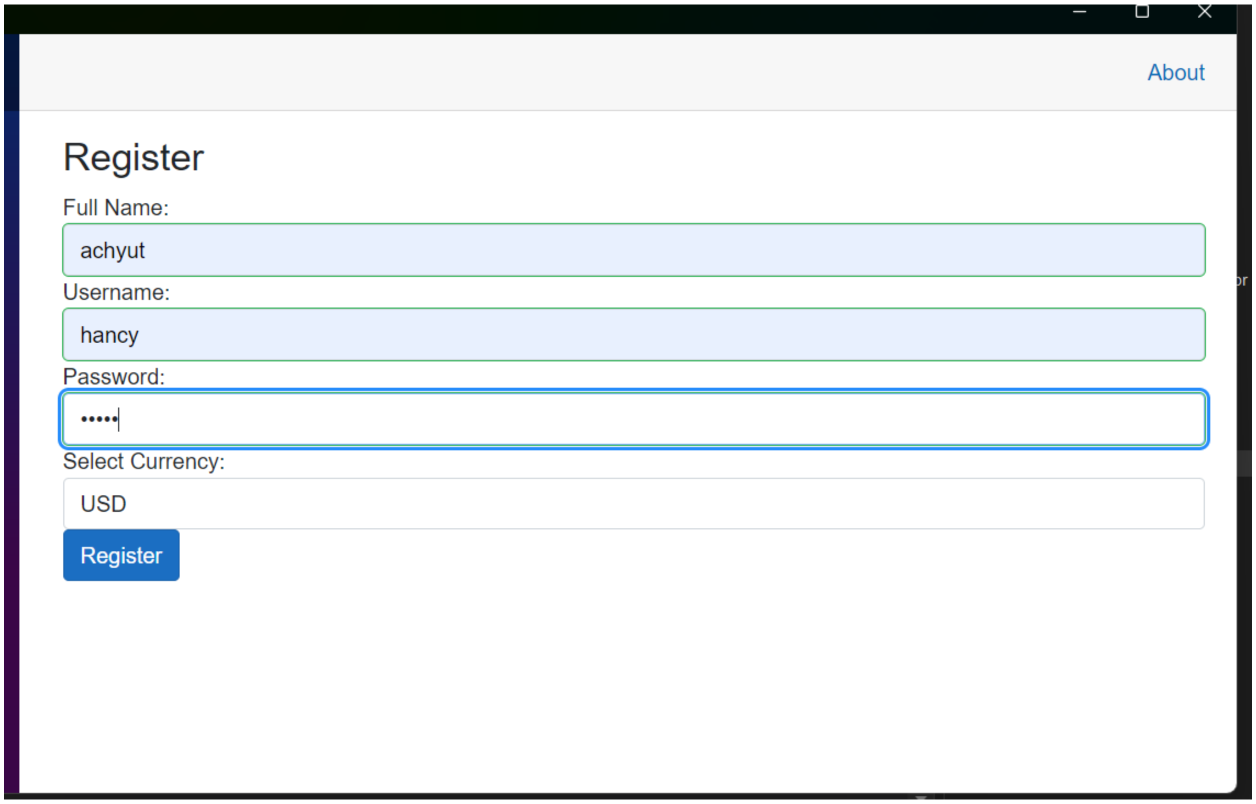
Select Currency:

USD

Register

All fields are required!

Figure 33: Invalid Fileds



Register

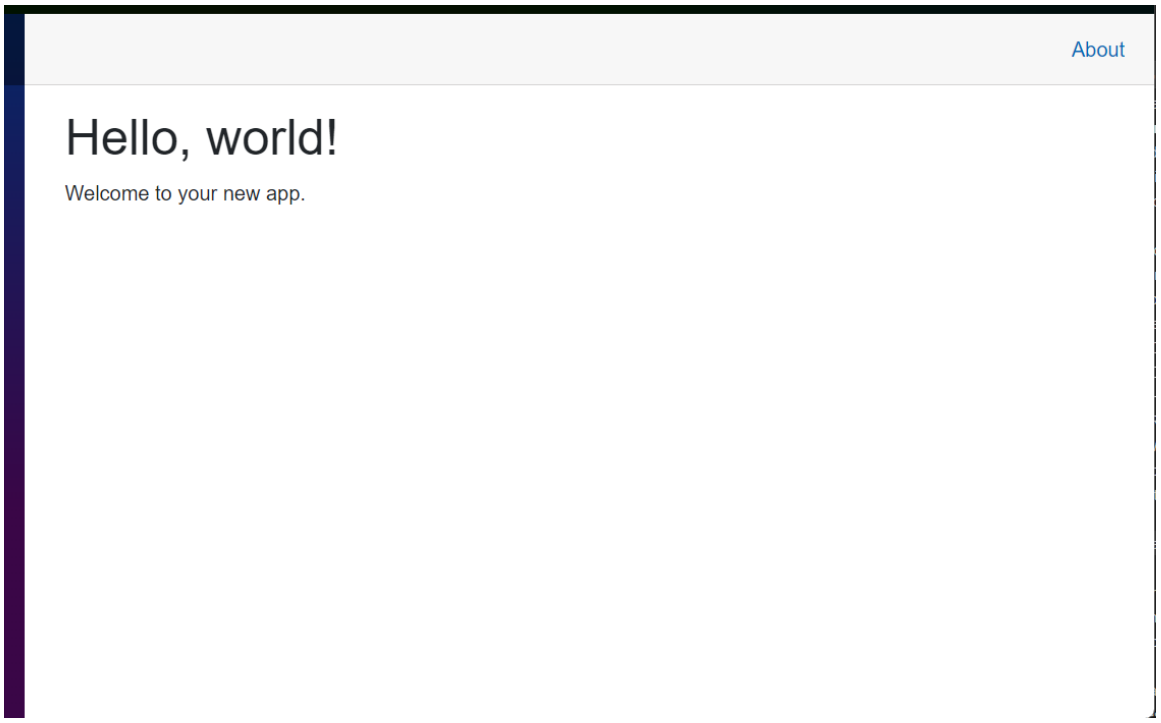
Full Name: achyut

Username: hancy

Password: •••••

Select Currency: USD

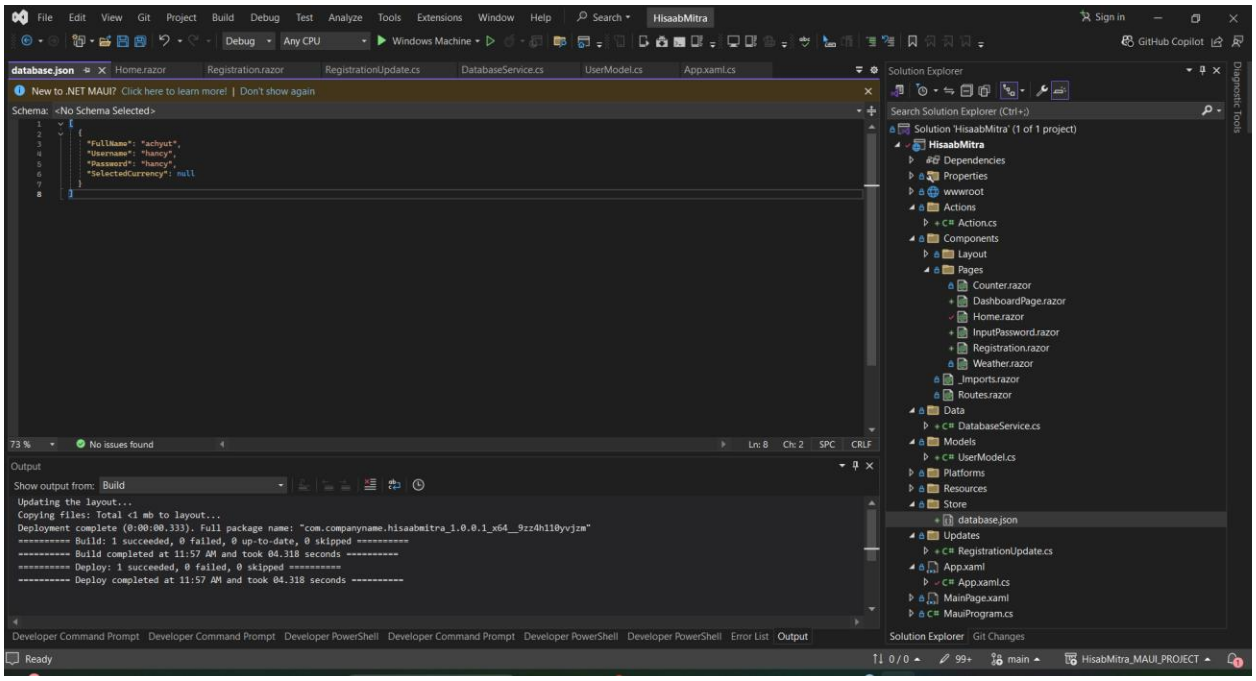
Figure 34: Inserting details



Hello, world!

Welcome to your new app.

Figure 35: Redirected to home page after registration



```
[
  {
    "FullName": "achyut",
    "Username": "hancy",
    "Password": "hancy",
    "SelectedCurrency": null
  }
]
```

```
Updating the layout...
Copying files: Total <1 mb to layout...
Deployment complete (0:00:00.333). Full package name: "com.companyname.hisaabmitra_1.0.0.1_x64__9zz4h110yvjzm"
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
---------- Build completed at 11:57 AM and took 04.318 seconds ----------
========== Deploy: 1 succeeded, 0 failed, 0 skipped ==========
---------- Deploy completed at 11:57 AM and took 04.318 seconds ----------
```

Figure 36: Insert data store in json file

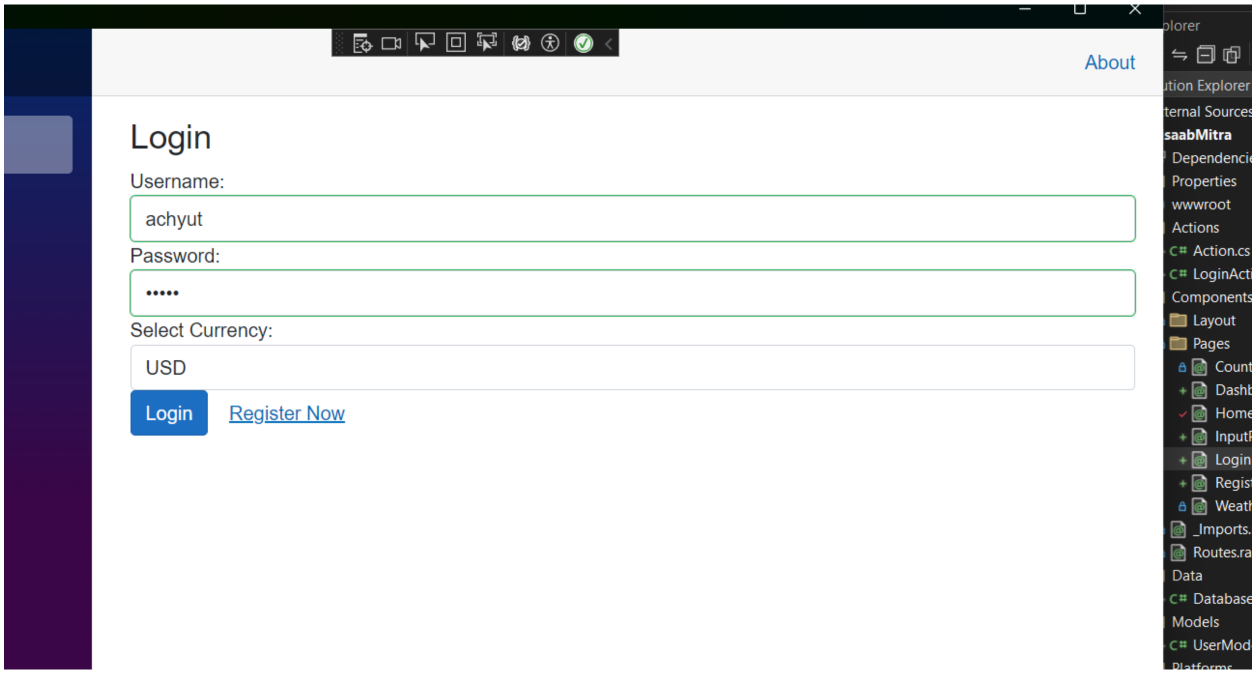
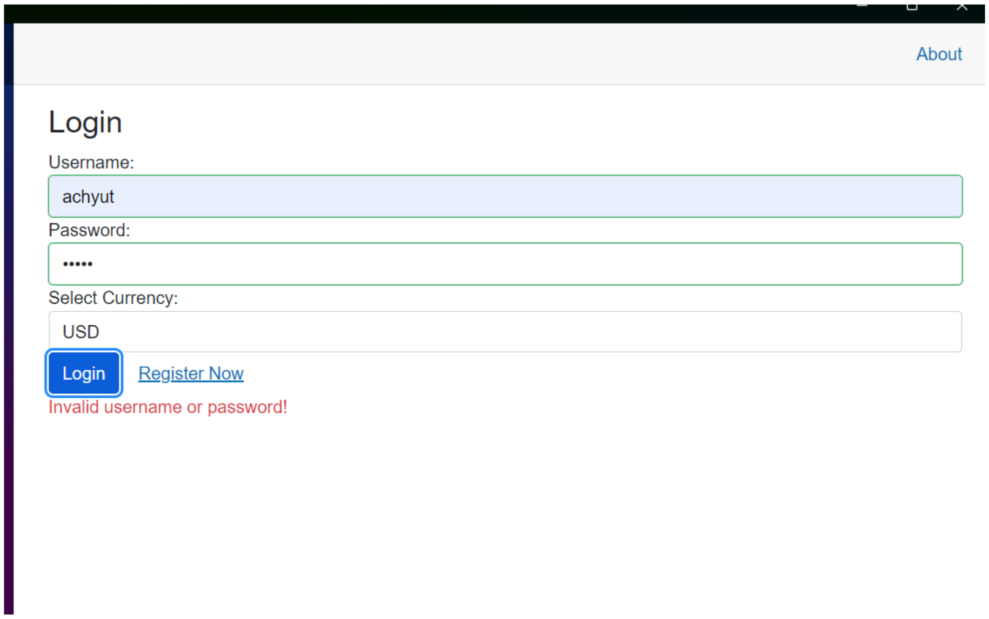
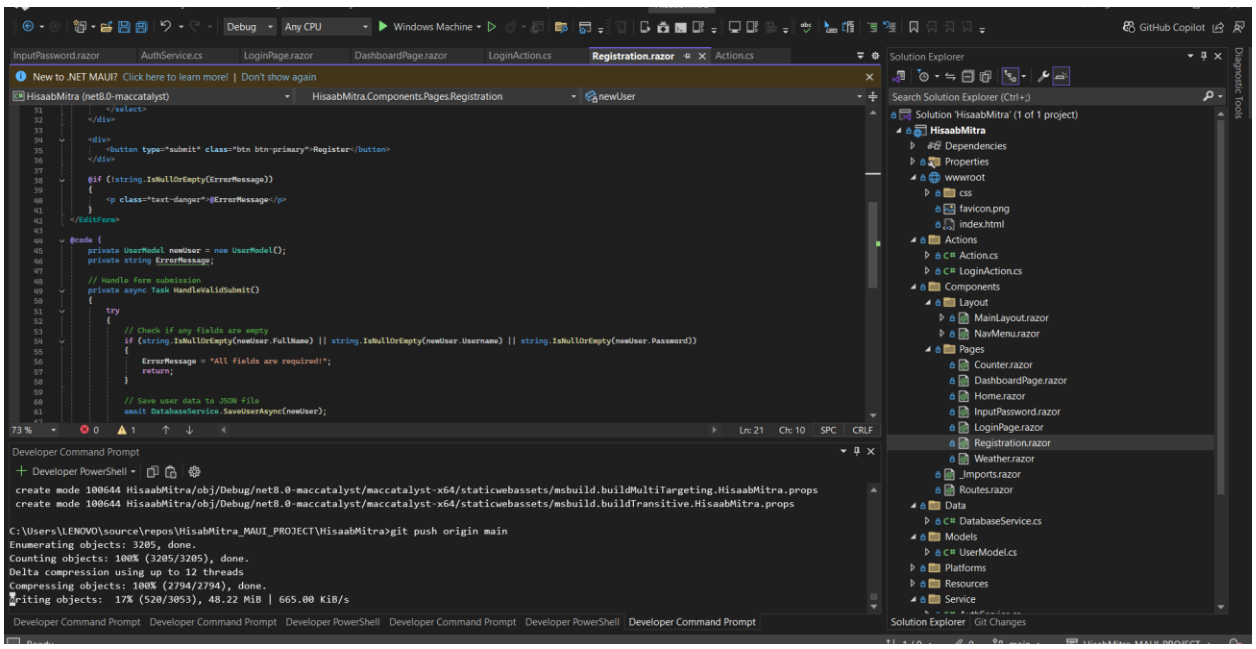


Figure 37: Login Page



Invalid username or password!

Figure 38: Entering wrong username



```
            </select>
        </div>

        <div>
            <button type="submit" class="btn btn-primary">Register</button>
        </div>

        @if (!string.IsNullOrEmpty(ErrorMessage))
        {
            <p class="text-danger">@ErrorMessage</p>
        }
    </EditForm>

@code {
    private UserModel newUser = new UserModel();
    private string ErrorMessage;

    // Handle form submission
    private async Task HandleValidSubmit()
    {
        try
        {
            // Check if any fields are empty
            if (string.IsNullOrEmpty(newUser.FullName) || string.IsNullOrEmpty(newUser.Username) || string.IsNullOrEmpty(newUser.Password))
            {
                ErrorMessage = "All fields are required!";
                return;
            }

            // Save user data to JSON file
            await DatabaseService.SaveUserAsync(newUser);
```

```
 create mode 100644 HisaabMitra/obj/Debug/net8.0-maccatalyst/maccatalyst-x64/staticwebassets/msbuild.buildMultiTargeting.HisaabMitra.props
 create mode 100644 HisaabMitra/obj/Debug/net8.0-maccatalyst/maccatalyst-x64/staticwebassets/msbuild.buildTransitive.HisaabMitra.props

C:\Users\LENOVO\source\repos\HisabMitra_MAUI_PROJECT\HisaabMitra>git push origin main
Enumerating objects: 3205, done.
Counting objects: 100% (3205/3205), done.
Delta compression using up to 12 threads
Compressing objects: 100% (2794/2794), done.
Writing objects:  17% (520/3053), 48.22 MiB | 665.00 KiB/s
```

Figure 39: Pushing login and register function in github main branch