Personal Finance Tracker

A Project Report

By

Sai Pranav Sripathi

saipranav.sripathi@sjsu.edu

ID:018188677

Kumari Shyama

kumari.shyama@sjsu.edu

ID: 018231512

Prasanna Sai Rohit Durbha

ID: 018230693

rohit.durbha@sjsu.edu

Abhishek Rasikbhai Shekhada

ID: 017609475

abhishekrasikbhai.shekhada@sjsu.edu

Varshan Kumar

varshan.kumar@sjsu.edu

ID: 017099368

6th December 2024

1. GOALS AND DESCRIPTION

The **Personal Finance Tracker** is a scalable web-based application designed with the aim to streamline and simplify personal financial management. The app will allow users to organize their financial accounts, including transactions, goals, budgets, investments, and loans leveraging a robust database management system.

The application provides essential features like organizing investments, loans, and daily transactions with ease. It helps in setting and tracking budgets, achieving financial goals, and categorizing expenses. Using MySQL database architecture with constraints, triggers, and indexes, and Python and Flask backend, the application ensures data integrity and consistency offering real-time updates and reliable performance with load times under 2 seconds.

A user can register and log-in into the system and record accounts and transactions which would be visible in real time on the application. The user can easily navigate through the entities and record their financial records. The application also helps in generating graphical reports in the form of pie charts to track total income, expenses, and savings. The app also caters to filtering transactions by date, type, account number and category.

The application offers a comprehensive solution tailored to individuals, families, and small business owners. It is built to scale seamlessly for over 100 users while maintaining 99% uptime reliability. This project demonstrates the practical implementation of database management systems to address challenges in financial tracking, presenting an efficient, scalable, and user-centric tool to manage and visualize personal finances effectively.

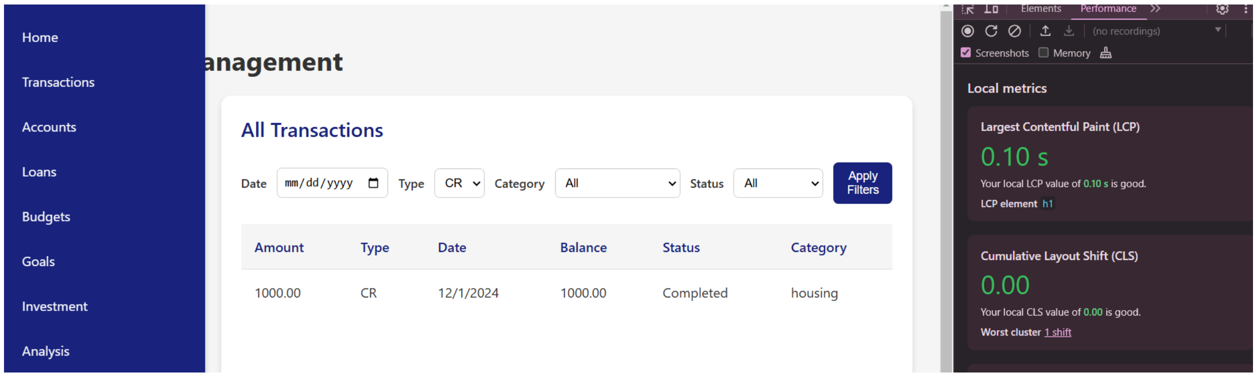
2. FUNCTIONAL & NON-FUNCTIONAL REQUIREMENTS AND ARCHITECTURE

Functional Requirements:

- **User Onboarding:**
  - Secure account registration and authentication with session support.
- **Financial Management Dashboard:**
  - Centralized platform to create, view, and manage financial accounts, transactions, loans, financial goals, investments, and budgets.
- **Transaction Management:**
  - Browse and categorize transactions by type (e.g., income, expenses, loans).
  - Add or update transactions for linked accounts with real-time syncing.
- **Goal Tracking:**
  - Create and manage financial goals, including saving targets or investment objectives.
  - Search and filter goals based on attributes like end date and amount.
- **Expense and Budget Analysis:**
  - Visualize spending patterns with dynamic charts and summaries.
  - Detailed breakdown of expenses by category or account.
- **Advanced Search and Analytics:**
  - Perform searches with advanced filters (e.g., date range, amount, or tags).

Non-Functional Requirements:

- **Performance:**
  - Search results/reports must load within 2 seconds.



anagement

All Transactions

Date mm/dd/yyyy Type CR Category All Status All

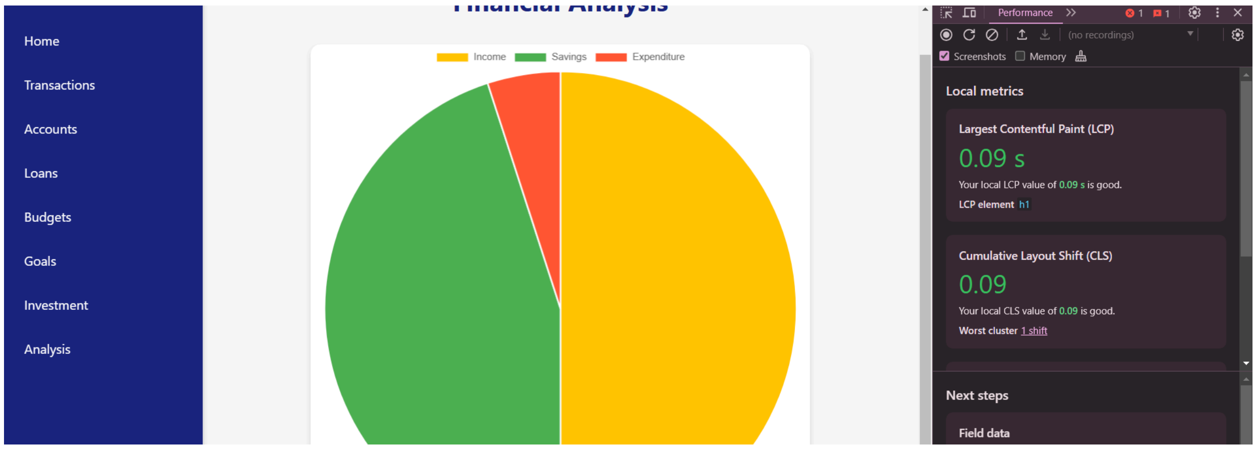
Amount	Type	Date	Balance	Status	Category
1000.00	CR	12/1/2024	1000.00	Completed	housing

Local metrics

Largest Contentful Paint (LCP)
0.10 s
Your local LCP value of 0.10 s is good.
LCP element h1

Cumulative Layout Shift (CLS)
0.00
Your local CLS value of 0.00 is good.
Worst cluster 1 shift

○ Daily expenditure calculations should be completed within 2–3 seconds.



Local metrics

Largest Contentful Paint (LCP)
0.09 s
Your local LCP value of 0.09 s is good.
LCP element h1

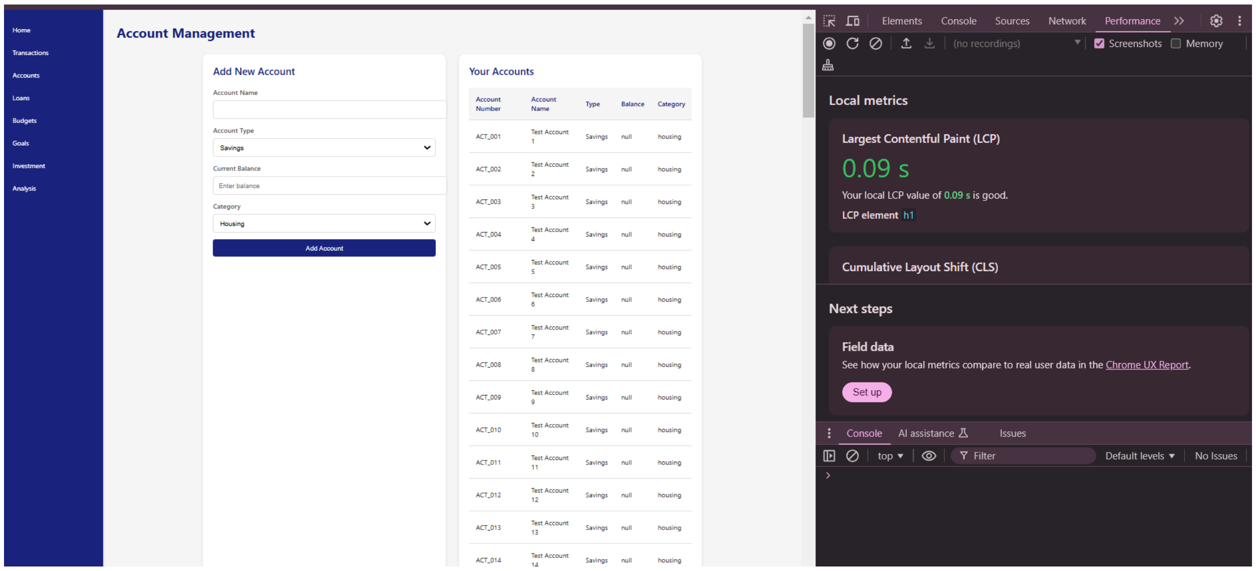
Cumulative Layout Shift (CLS)
0.09
Your local CLS value of 0.09 is good.
Worst cluster 1 shift

Next steps

Field data

○ Supports at least 100 accounts without performance degradation.

We inserted 100 accounts, but the performance of the application has not degraded.



Account Management

Add New Account

Account Name:

Account Type: Savings

Current Balance: Enter balance

Category: Housing

Your Accounts

Account Number	Account Name	Type	Balance	Category
ACT_001	Test Account 1	Savings	null	housing
ACT_002	Test Account 2	Savings	null	housing
ACT_003	Test Account 3	Savings	null	housing
ACT_004	Test Account 4	Savings	null	housing
ACT_005	Test Account 5	Savings	null	housing
ACT_006	Test Account 6	Savings	null	housing
ACT_007	Test Account 7	Savings	null	housing
ACT_008	Test Account 8	Savings	null	housing
ACT_009	Test Account 9	Savings	null	housing
ACT_010	Test Account 10	Savings	null	housing
ACT_011	Test Account 11	Savings	null	housing
ACT_012	Test Account 12	Savings	null	housing
ACT_013	Test Account 13	Savings	null	housing
ACT_014	Test Account 14	Savings	null	housing

Local metrics

Largest Contentful Paint (LCP)

0.09 s

Your local LCP value of 0.09 s is good.

LCP element h1

Cumulative Layout Shift (CLS)

Next steps

Field data

See how your local metrics compare to real user data in the Chrome UX Report.

- **Usability:**
  - Intuitive UI with clear and visible labels for all users.
  - Displays appropriate error messages on application failure.
- **Data Integrity and Availability:**
  - Database availability of 99% or higher.

Our SQL database has been 100% available. We have not faced any database performance issues.

  - Consistent data maintenance with cascading updates for transactions.

All the tuples are clearly visible in the database as well as the application

  - Historical data is stored and accessible at any time.

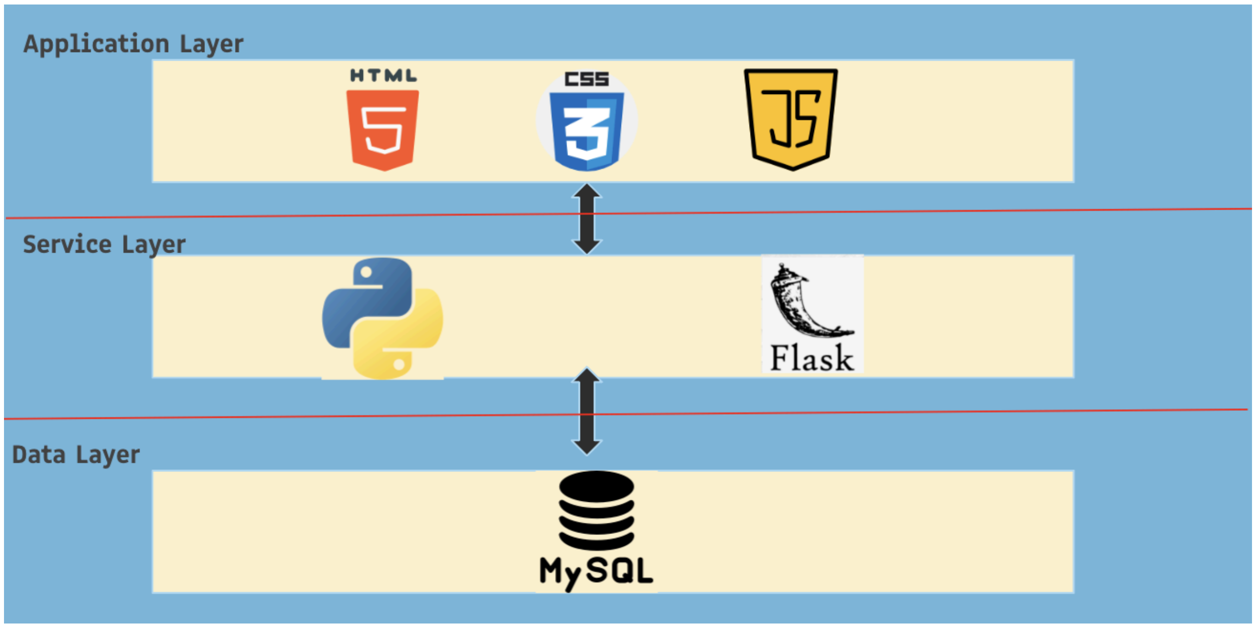
Historical data can be accessed through a date filter for a user anytime.

- **Database Design:**
  - Normalized schema to avoid redundancy, with primary keys and indexing.

The Schema is Normalized to BCNF form and the queries are running within

- Enforced constraints, including foreign key relationships.

- **Reliability:**
  - Ensures up-to-date and consistent data across the system.



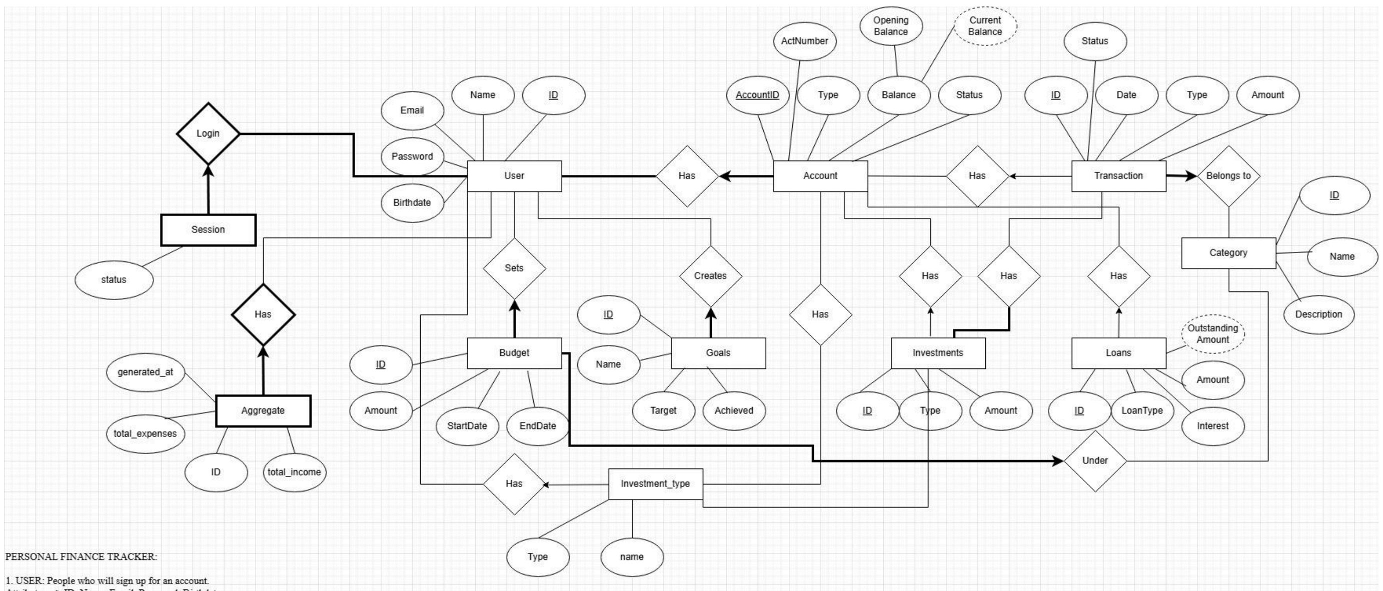
Application Architecture

The application adopts a three-tier architecture with the following components:

- **Frontend:**
  - Developed with HTML, CSS, and JavaScript libraries.
  - Manages UI and user interactions.
  - Communicates with the backend via RESTful API calls.
- **Backend:**
  - Built on the Flask (Python) framework.
  - Handles HTTP requests, business logic, and database interactions.
  - Provides RESTful APIs for CRUD operations.
- **Database:**
  - MySQL relational database for storing user data, accounts, transactions, and more.

- ○ Integrated with the backend using SQLAlchemy ORM.

3. ER DIAGRAM



PERSONAL FINANCE TRACKER:

1. USER: People who will sign up for an account.
Attributes --> ID, Name, Email, Password, Birthdate

2. ACCOUNT: Users can create multiple financial accounts like Savings, Credit Card, etc
Attributes --> AccountNumber, Type, Balance (Opening balance, Current Balance), Status
Current Balance = Opening Balance - Sum of all outgoing transactions + sum of all incoming transactions

3. TRANSACTIONS: Financial transactions, incoming/outgoing
Attributed --> ID, Date, Status, Type, Amount, Balance

4. AGGREGATE: Displays total expenses and income of any user
Attributes --> aggregate_id, total_income, total expenditure, generated_st

5. BUDGET: Amount constraints set by a user under different categories
Attributed --> ID, Amount, StartDate, EndDate

6. GOALS: Financial goals set by a user
Attributes --> ID, Name, Target, Deadline

7. Investments: Stocks, Mutual funds, etc, it is associated with transactions
Attributes --> ID, name

8. Investment_type: Stocks, Mutual funds, etc, it is associated with transactions
Attributes --> Type, name

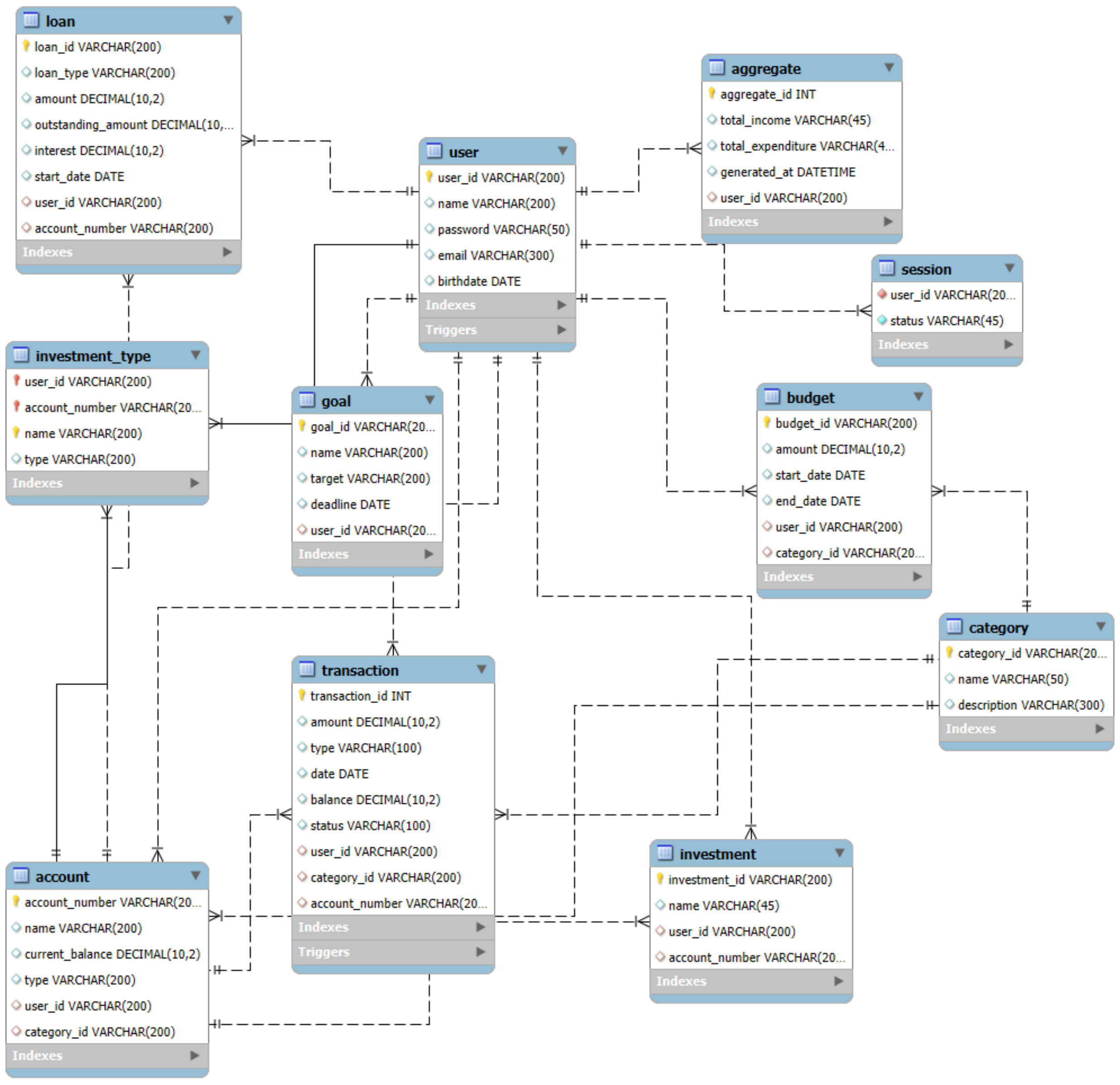
9. Loans: Any loans taken by a user can be tracked under an account
Attributes --> ID, LoanType, Amount, OutstandingAmount, Interest, Start_date
OutstandingAmount = Total Amount - Sum of the balance paid

10. CATEGORY: Categories to classify budgets and transactions
Attributes --> ID, Name, Description

11. SESSION: USER active session while logged in
Attributes --> status

4. DATABASE DESIGN

We used MYSQL workbench to design and implement our database.



Relational Database Schema:

1. User (<u>user_id: String</u>, name: String, password: String, email: String, birthdate: Date)

2. Category (category_id: String, name: String, description: String)

3. Transaction (transaction_id: String, amount: Integer, date: Date, type: String, status: String, user_id: String, category_id: String)

Transaction: user_id IS FK REFERENCE User: user_id

Transaction: category_id IS FK REFERENCE Category: category_id

4. Budget (budget_id: String, amount: Integer, start_date: Date, end_date: Date, user_id: String, category_id: String)

Budget: user_id IS FK REFERENCE User: user_id

Budget: category_id IS FK REFERENCE Category: category_id

5. Goal (goal_id: String, name: String, target: Integer, deadline: Date, user_id: String)

Goal: user_id IS FK REFERENCE User: user_id

6. Account (account_number: String, amount: Integer, balance: Integer, type: String, user_id: String, category_id: String)

Transaction: user_id IS FK REFERENCE User: user_id

Transaction: category_id IS FK REFERENCE Category: category_id

7. Loan (loan_id: String, loan_type: String, amount: Integer, outstanding_amount: Integer, interest: Integer, user_id: String, account_number: String)

Loan: user_id IS FK REFERENCE User: user_id

Loan: account_number IS FK REFERENCE Account: account_number

9. Investment (investment_id: String, name: String, type: String, user_id: String, account_number: String)

Investment: user_id IS FK REFERENCE User: user_id

Investment: account_number IS FK REFERENCE Account: account_number

10. Aggregate (<u>aggregate_id: String</u>, total_income: Integer, total_expenditure: Integer, generated_at: Datetime, user_id: String)

Aggregate: user_id IS FK REFERENCE User: user_id

11. Session (user_id: String, status: String)

Session: user_id IS FK REFERENCE User: user_id

BCNF Normalization:

1. User

Functional Dependencies:

user_id → name, password, email, birthdate

email → user_id, name, password, birthdate

2. Transaction

Functional Dependencies:

transaction_id → amount, date, type, status, user_id, category_id

3. Category

Functional Dependencies:

category_id → name, description

4. Budget

Functional Dependencies:

budget_id → amount, start_date, end_date, user_id, category_id

5. Goal

Functional Dependencies:

goal_id → name, target, deadline, user_id

6. Account

Functional Dependencies:

account_number → amount, balance, type, user_id, category_id

user_id, category_id → account_number, amount, balance, type (We are expecting each user will have at most one account in each cat.

7. Loan

Functional Dependencies:

loan_id → loan_type, amount, outstanding_amount, interest, user_id, account_number

user_id, account_number → loan_id, loan_type, amount, outstanding_amount, interest (optional, maybe not)

8. Investment

Functional Dependencies:

investment_id → name, type, user_id, account_number

user_id, account_number, name → type

BCNF: T1 (investment_id, name, user_id, account_number)

T2 (user_id, account_number, name, type)

9. Aggregate

Functional Dependencies:

aggregate_id → total_income, total_expenditure, generated_at, user_id

user_id, generated_at → aggregate_id, total_income, total_expenditure

10. Session

Functional Dependencies:

user_id → status

5. MAJOR DESIGN DECISIONS

The following design decisions were taken during the process of designing the database, and developing the backend and front end of the web application:

1. User Authentication and Session Management

To provide a seamless experience, mechanisms were developed for managing user authentication and maintaining session states of the user. The priority was to safely handle user credentials and ensure that users can transition smoothly between different interactions and pages without losing their session data.

2. Real-time Data Updates

The web application was designed to support real-time data synchronization between the frontend and backend. This was particularly critical for features like dynamic charts implemented on the “Analysis" page as it guarantees that users always have access to the most current financial information and transaction records and the ability to filter transactions as required on-demand.

3. Database Optimization

A robust and efficient database architecture was developed to manage the large volume of transaction data. This included designing optimal schemas, optimized queries, and leveraging relational data management techniques for smooth performance, even under heavy usage.

4. Cross-browser Compatibility

For a smooth and consistent user experience on the web application, it was tested and optimized to function efficiently across different web browsers and devices. This involved ensuring responsiveness in the design and addressing compatibility issues.

5. Integration of Frontend and Backend

Smooth communication between the frontend UI and the backend server was a critical priority. This was achieved by synchronizing the data flow using APIs (like Flask APIs), allowing for quick and responsive interactions between the user and the web application, without any latency involved.

6. IMPLEMENTATION

Activity	Date
DB schema design	9/22/2024
Normalizing DB Design	10/15/2024

Create MySQL Database	10/20/2024
Base code - virtual environment for backend	11/12/2024
Initialised Git repository	11/15/2024
Initialized with frontend pages	11/20/2024
Connecting backend with MySQL and executing ddl.sql file	11/22/2024
Developed routes for login and logout	11/25/2024
Website localhost for login/logout functions	11/30/2024
Finished with integrating all UI pages with the backend via API calls	12/01/2024
Updated filtering options	12/03/2024
UI Restructure	12/04/2024
Code Due	12/05/2024

Resources

Programming Languages-

JavaScript:

- For frontend and API calls into the backend.

Python:

- For the backend, to develop endpoints and connect to DB

CSS:

- For styling the frontend.

Frameworks-

Flask:

- To run generally server-side (backend/middleware)

Database - MySQL

- Where user, group, event, and item data lives

Folder Structure

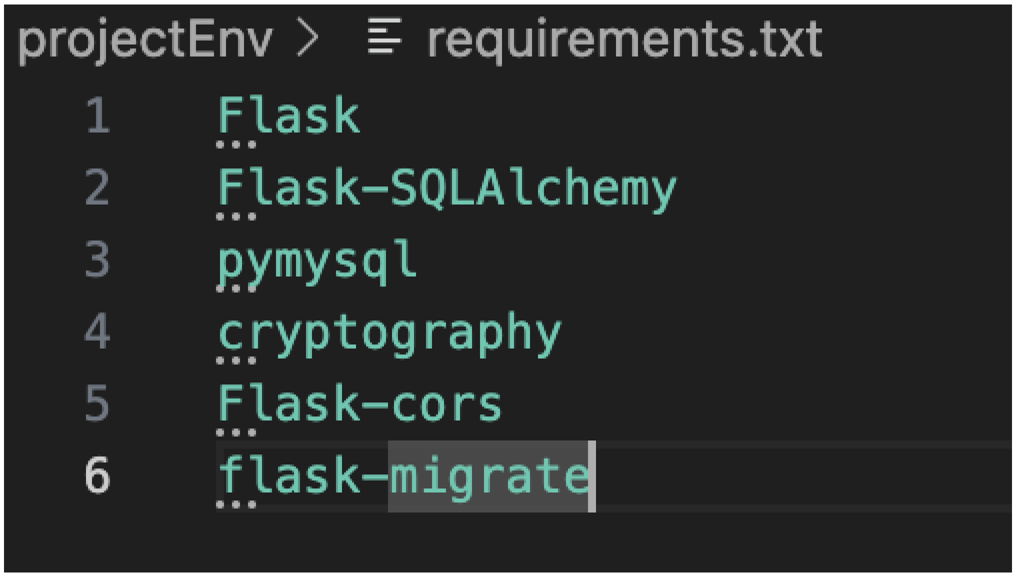
Backend (Server-side)

The backend is organized within the projectenv folder, comprising the implementation of the Flask server, route creation, database connection, and SQL files for database schema (DDL).

Python Dependencies:

- **Flask:** Initializes the backend server and handles HTTP requests.
- **Flask-CORS:** Middleware enabling seamless interaction between the client and backend server by handling Cross-Origin Resource Sharing (CORS).
- **Flask-SQLAlchemy and pymysql:** Facilitate connection and communication between the Flask server and the MySQL database.
- **Cryptography:** Ensures secure connections and data encryption during database interactions.

This setup ensures a secure, efficient, and scalable backend infrastructure for the application.



```
Flask
Flask-SQLAlchemy
pymysql
cryptography
Flask-cors
flask-migrate
```

Application Initialization:

- The __init__.py file initializes the Flask backend application, setting up the foundational structure for the server.
- Establishes a connection to the MySQL database by leveraging configurations from config.py, application extensions defined in extensions.py, and database initialization logic from init_db.py.
- Database schema is structured and managed using database_ddl.sql.

Routing and Logic Handling:

- The app.py file defines application routes and handles incoming HTTP requests.
- Routes are designed to process user interactions, communicate with the database via APIs, and return the required output or response.
- Implements business logic, ensuring secure and efficient data operations.

FrontEnd (Client-side)

The **index.html** file serves as the entry point for the client-side application. It dynamically renders the appropriate user interface based on the authentication status of the user:

Conditional Rendering:

- If the user's session and cookies contain valid user_id and user_name details, the application displays user-specific pages, such as account details, transactions, and settings.
- If the session or cookies are missing or invalid, the application restricts access to all other pages and redirects the user to the login/signup interface.

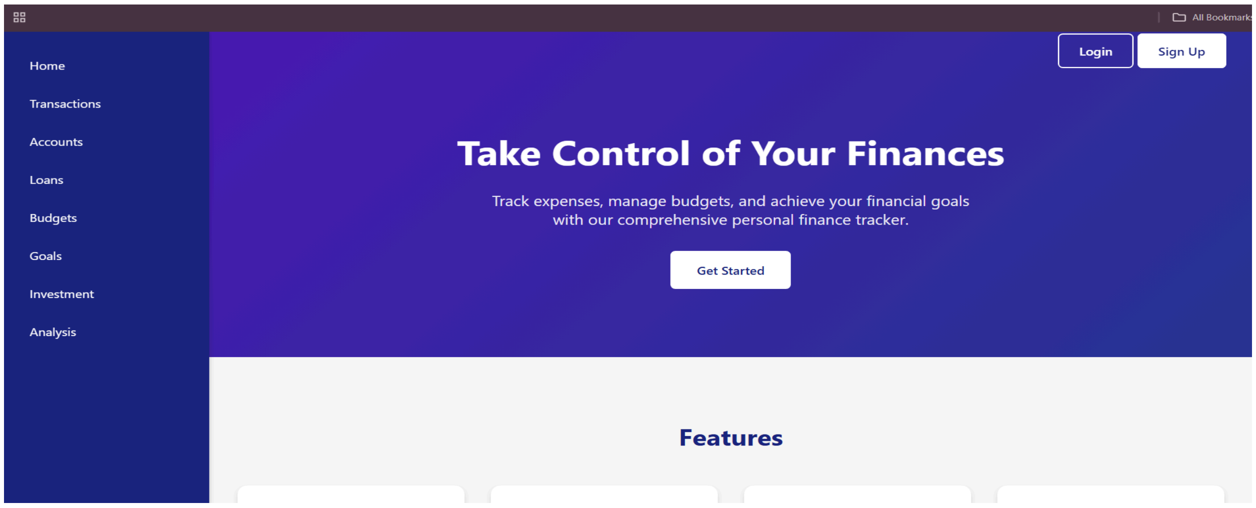
Authentication Workflow:

1. **Login/Signup Process:**
   - Users are required to provide valid credentials during login or complete the signup process to create a new account.
   - Upon successful authentication, the user_id and user_name are stored securely in session cookies.
2. **Session Management:**
   - The client checks for the presence and validity of session cookies on every page load.
   - If the session expires or is tampered with, the application automatically logs the user out and redirects them to the login page.
3. **Access Control:**
   - All subsequent API requests include the user_id and user_name from the session cookies to validate the user's identity.
   - Unauthorized users attempting to access restricted pages are blocked and prompted to log in.

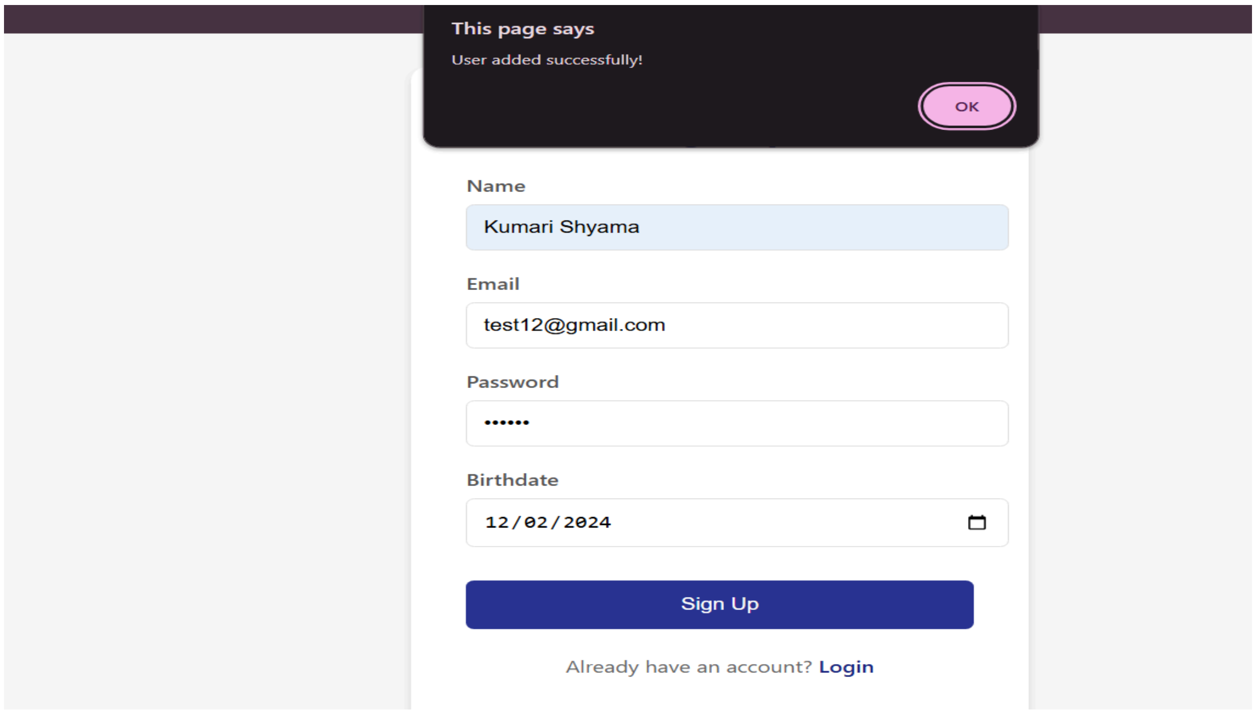
7. DEMONSTRATION OF SYSTEM RUN

1. User Registration and Login

The Personal Financial Tracker application front page describes the functionality of the features provided and also lets user register or login.


All Bookmarks
Login
Sign Up
Home
Transactions
Accounts
Loans
Budgets
Goals
Investment
Analysis
Take Control of Your Finances
Track expenses, manage budgets, and achieve your financial goals with our comprehensive personal finance tracker.
Get Started
Features

Any new user will be able to register in the application system using the “Sign-up” button. Once the user is registered the session table and aggregate table will be inserted with the user row. In the session table the User will show as INACTIVE till the user logs in.


This page says

User added successfully!

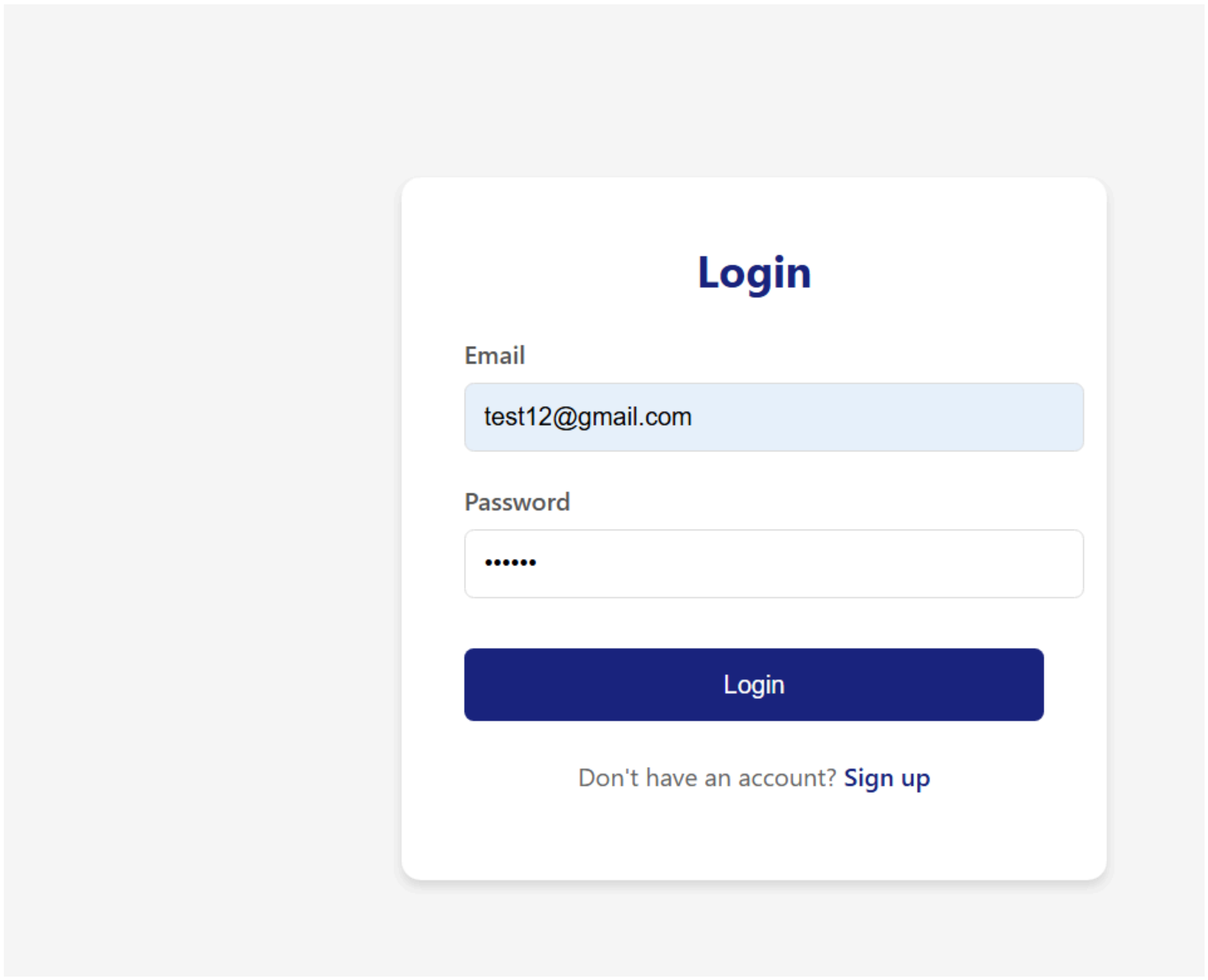
Name: Kumari Shyama

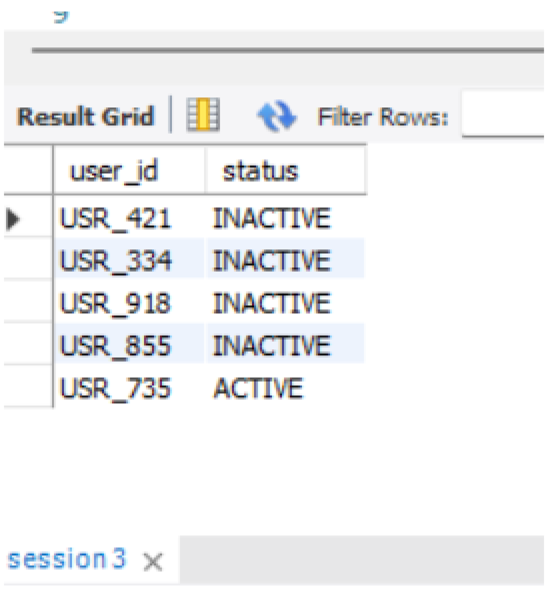
Email: test12@gmail.com

Password: ••••••

Birthdate: 12/02/2024

Next the user should be logged in for his session to be ACTIVE.

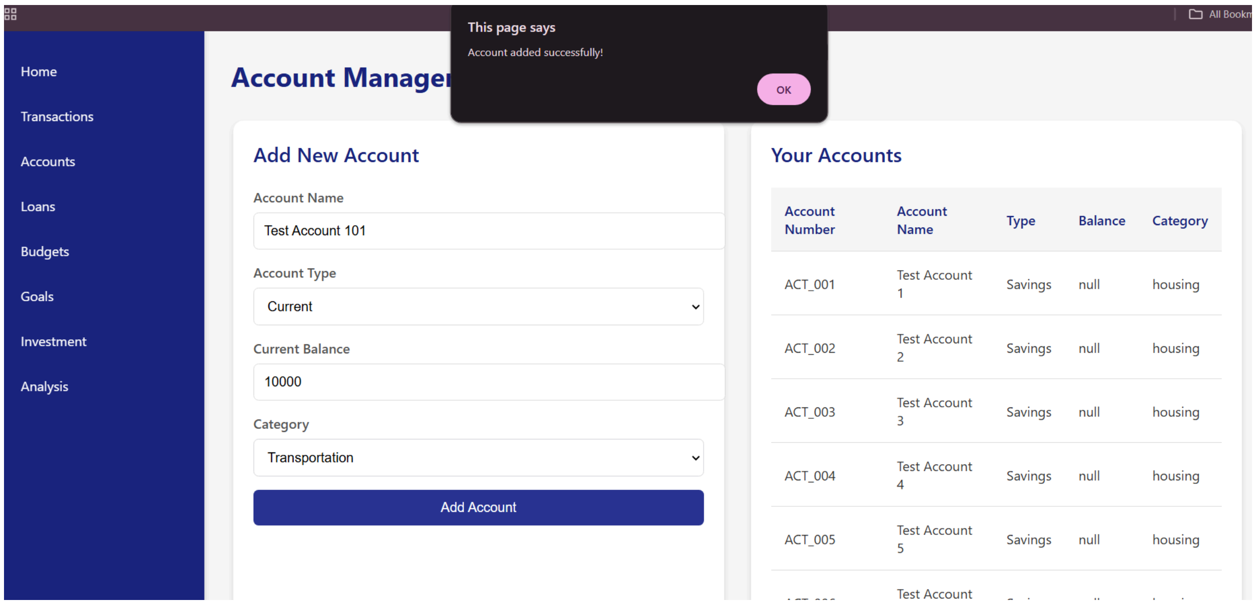




user_id	status
USR_421	INACTIVE
USR_334	INACTIVE
USR_918	INACTIVE
USR_855	INACTIVE
USR_735	ACTIVE

2. Accounts

Once a user has logged in they need to create an account or multiple accounts to log their transactions into.



This page says

Account added successfully!

Add New Account

Account Name: Test Account 101

Account Type: Current

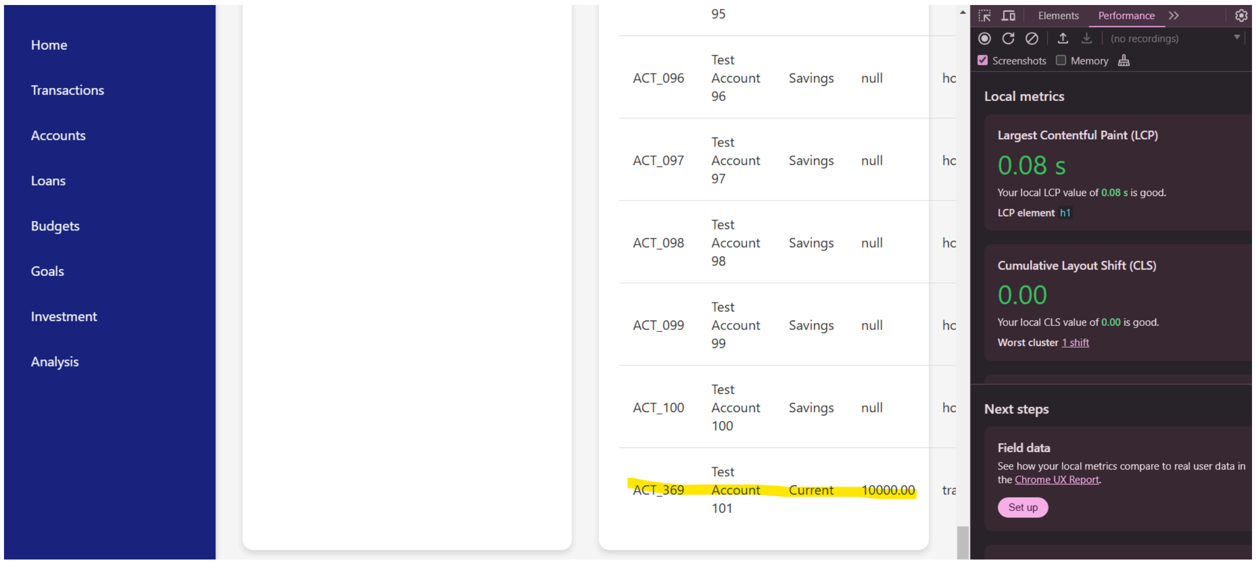
Current Balance: 10000

Category: Transportation

Your Accounts

Account Number	Account Name	Type	Balance	Category
ACT_001	Test Account 1	Savings	null	housing
ACT_002	Test Account 2	Savings	null	housing
ACT_003	Test Account 3	Savings	null	housing
ACT_004	Test Account 4	Savings	null	housing
ACT_005	Test Account 5	Savings	null	housing
	Test Account			

The added account is getting reflected on the application in under 0.10 seconds.



	95			
ACT_096	Test Account 96	Savings	null	hc
ACT_097	Test Account 97	Savings	null	hc
ACT_098	Test Account 98	Savings	null	hc
ACT_099	Test Account 99	Savings	null	hc
ACT_100	Test Account 100	Savings	null	hc
ACT_369	Test Account 101	Current	10000.00	tra

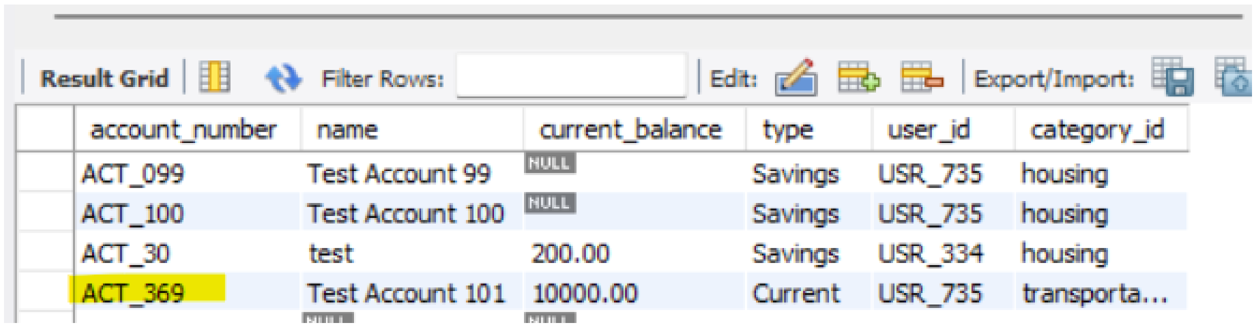
Local metrics

Largest Contentful Paint (LCP)
0.08 s
Your local LCP value of 0.08 s is good.
LCP element h1

Cumulative Layout Shift (CLS)
0.00
Your local CLS value of 0.00 is good.
Worst cluster 1 shift

Next steps

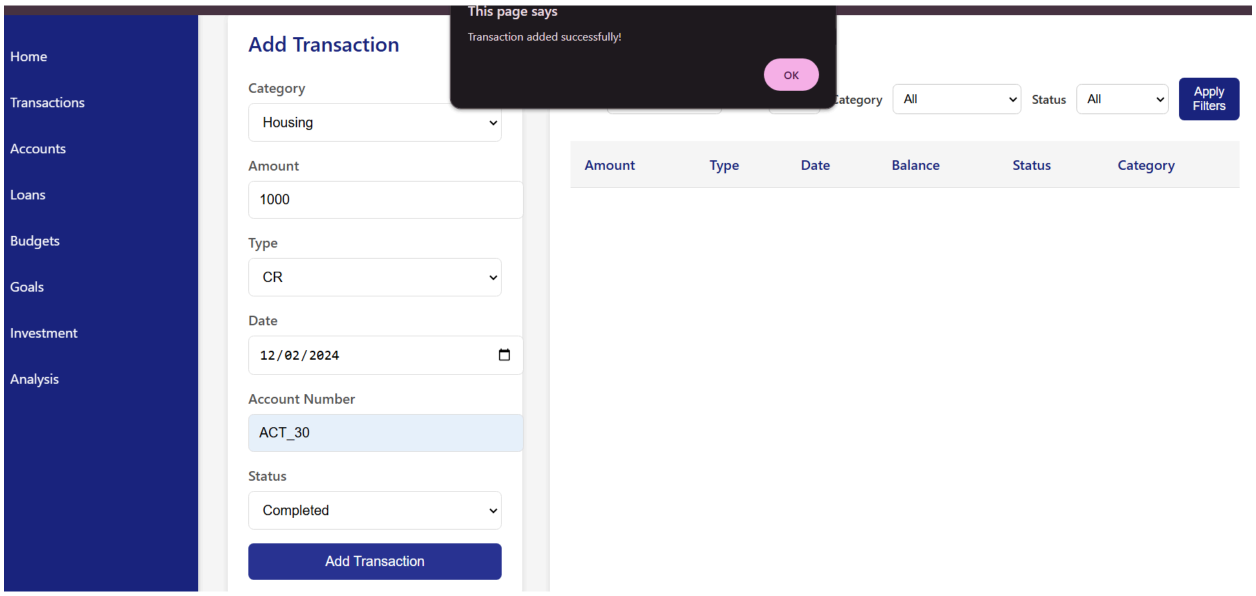
Field data
See how your local metrics compare to real user data in the Chrome UX Report.



	account_number	name	current_balance	type	user_id	category_id
	ACT_099	Test Account 99	NULL	Savings	USR_735	housing
	ACT_100	Test Account 100	NULL	Savings	USR_735	housing
	ACT_30	test	200.00	Savings	USR_334	housing
	ACT_369	Test Account 101	10000.00	Current	USR_735	transporta...

3. Transactions:

The application helps in adding and tracking user’s financial transactions to maintain accurate records of the income and expenses. The transactions can be categorised under multiple sections.


This page says

Transaction added successfully!

Add Transaction

Category: Housing

Amount: 1000

Type: CR

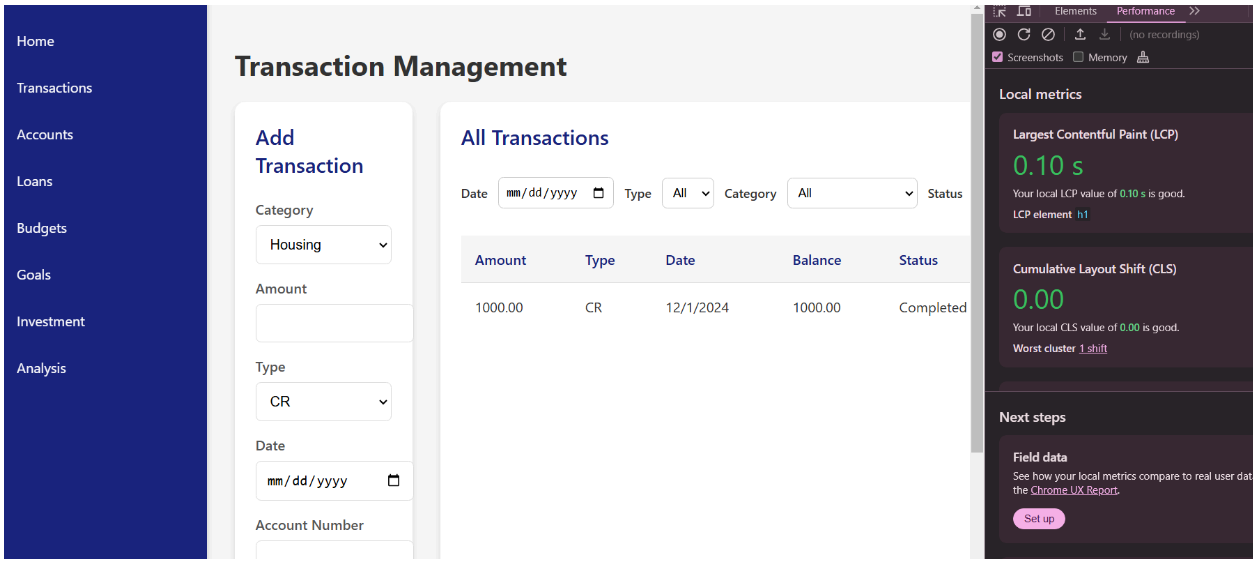
Date: 12/02/2024

Account Number: ACT_30

Status: Completed

Amount	Type	Date	Balance	Status	Category

Transactions are reflected with balance which is calculated using triggers in the database.



Transaction Management

Add Transaction

Category: Housing

Amount:

Type: CR

Date: mm/dd/yyyy

Account Number

All Transactions

Date mm/dd/yyyy Type All Category All Status

Amount	Type	Date	Balance	Status
1000.00	CR	12/1/2024	1000.00	Completed

Local metrics

Largest Contentful Paint (LCP)

0.10 s

Your local LCP value of 0.10 s is good.

LCP element h1

Cumulative Layout Shift (CLS)

0.00

Your local CLS value of 0.00 is good.

Worst cluster 1 shift

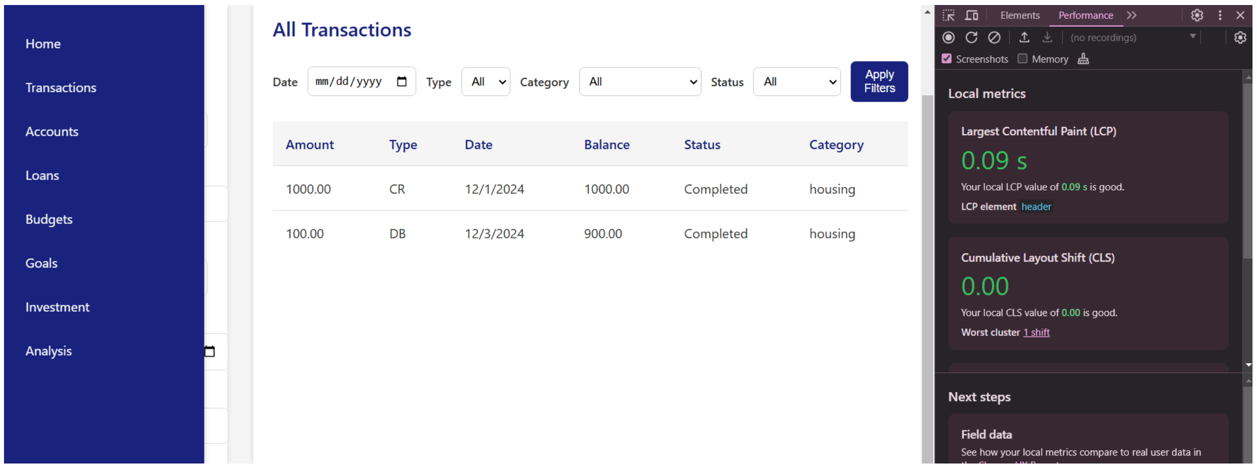
Next steps

Field data

See how your local metrics compare to real user dat

the Chrome UX Report.

If we add a debit transaction on the same category the balance reflects the updated balance.



All Transactions

Date mm/dd/yyyy Type All Category All Status All Apply Filters

Amount	Type	Date	Balance	Status	Category
1000.00	CR	12/1/2024	1000.00	Completed	housing
100.00	DB	12/3/2024	900.00	Completed	housing

Local metrics

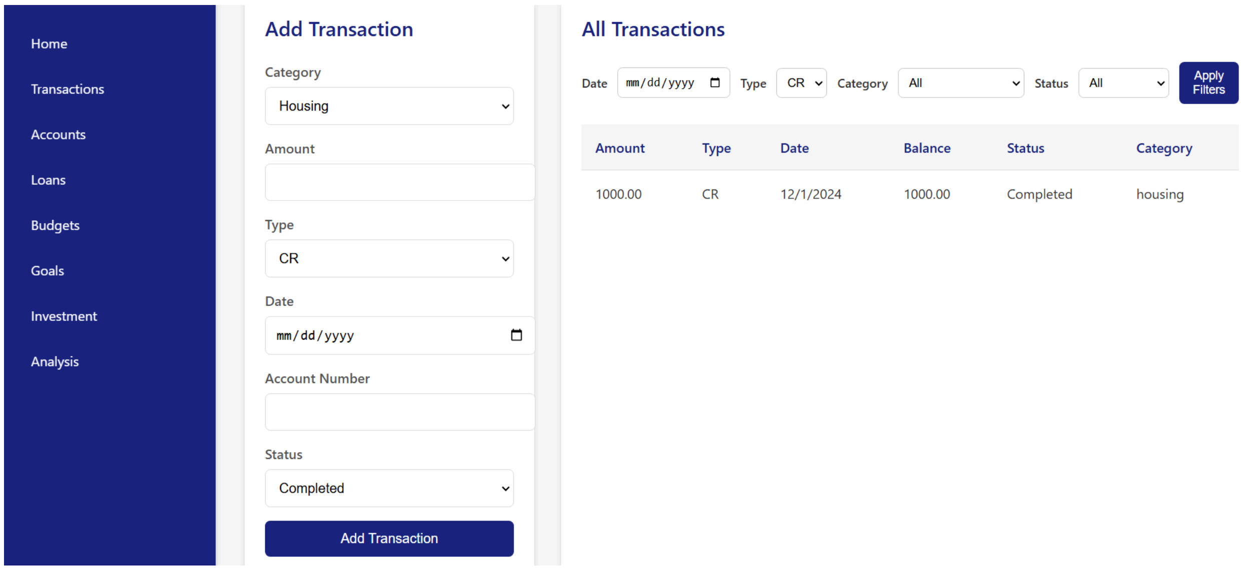
Largest Contentful Paint (LCP)
0.09 s
Your local LCP value of 0.09 s is good.
LCP element header

Cumulative Layout Shift (CLS)
0.00
Your local CLS value of 0.00 is good.
Worst cluster 1 shift

Next steps

Field data
See how your local metrics compare to real user data in

We can filter transactions by date, type, category and status. Here we have filtered using type (CR: credit, DB: debit)



Add Transaction

Category: Housing
Amount:
Type: CR
Date: mm/dd/yyyy
Account Number:
Status: Completed

Add Transaction

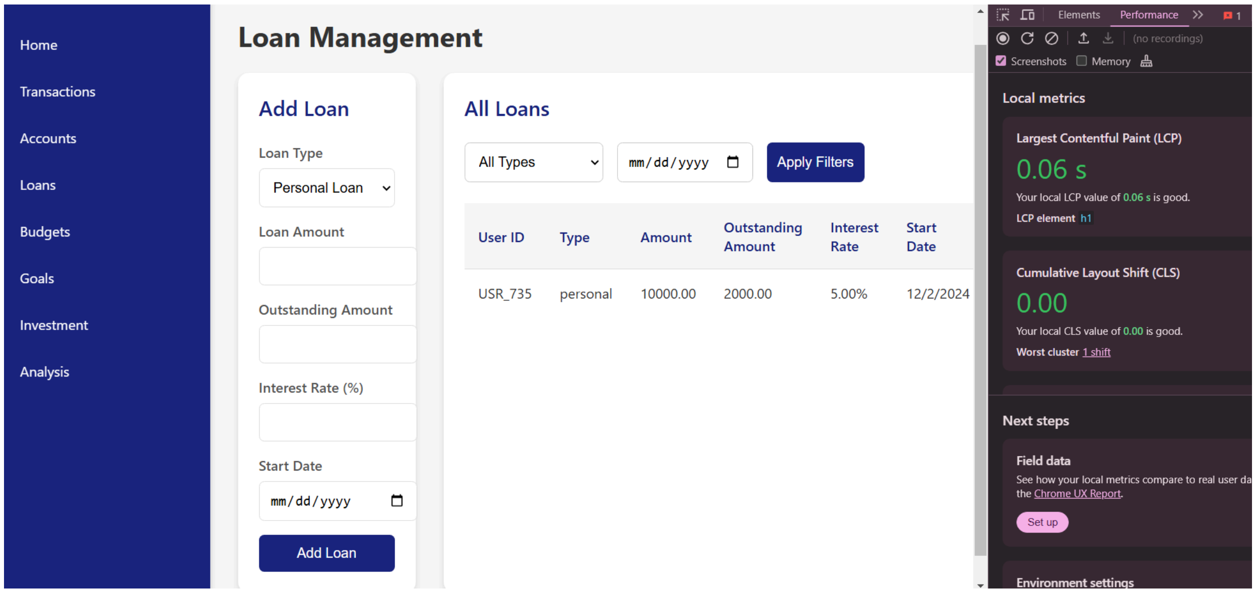
All Transactions

Date mm/dd/yyyy Type CR Category All Status All Apply Filters

Amount	Type	Date	Balance	Status	Category
1000.00	CR	12/1/2024	1000.00	Completed	housing

4. Loans

Users can add and track loans and outstanding balances using the loans section. Loans can be of multiple types. We are adding Personal Loan. Our application performance remains in good condition.


Loan Management

Add Loan

Loan Type: Personal Loan

Loan Amount

Outstanding Amount

Interest Rate (%)

Start Date: mm/dd/yyyy

All Loans

User ID	Type	Amount	Outstanding Amount	Interest Rate	Start Date
USR_735	personal	10000.00	2000.00	5.00%	12/2/2024

Local metrics

Largest Contentful Paint (LCP)

0.06 s

Your local LCP value of 0.06 s is good.

LCP element h1

Cumulative Layout Shift (CLS)

0.00

Your local CLS value of 0.00 is good.

Worst cluster 1 shift

Next steps

Field data

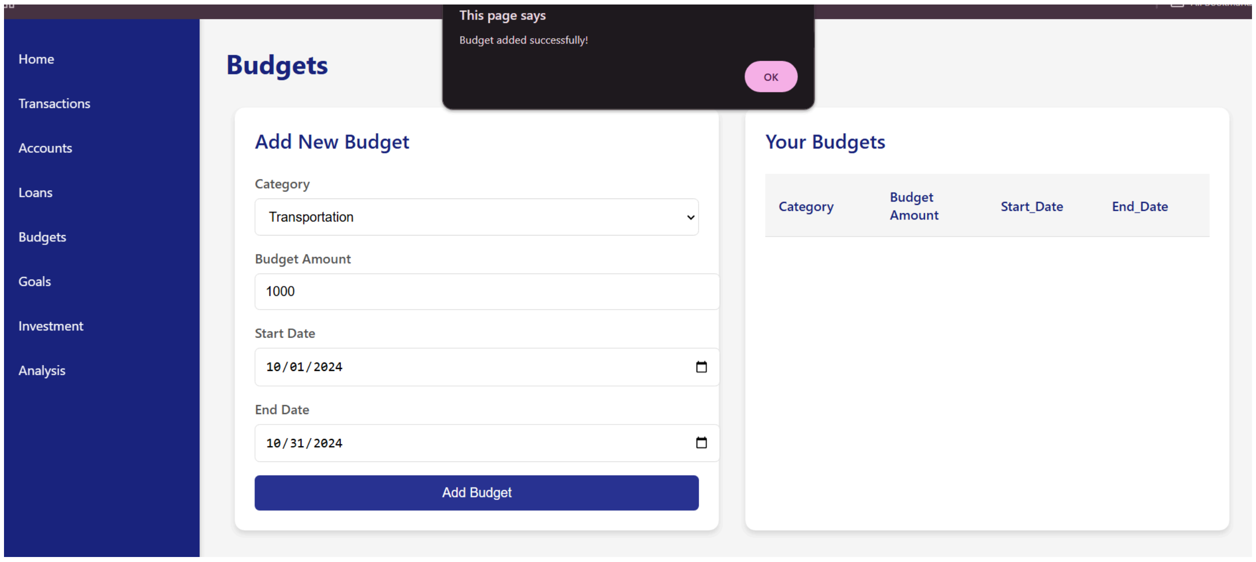
See how your local metrics compare to real user da the Chrome UX Report.

Environment settings

5. Budget, Goals and Investment

Similar to loans, user can add their specific budgets, goals and investment in the application and the financial tracker helps them track and see all the historical data as well.

a. Budget:


This page says

Budget added successfully!

Budgets

Add New Budget

Category: Transportation

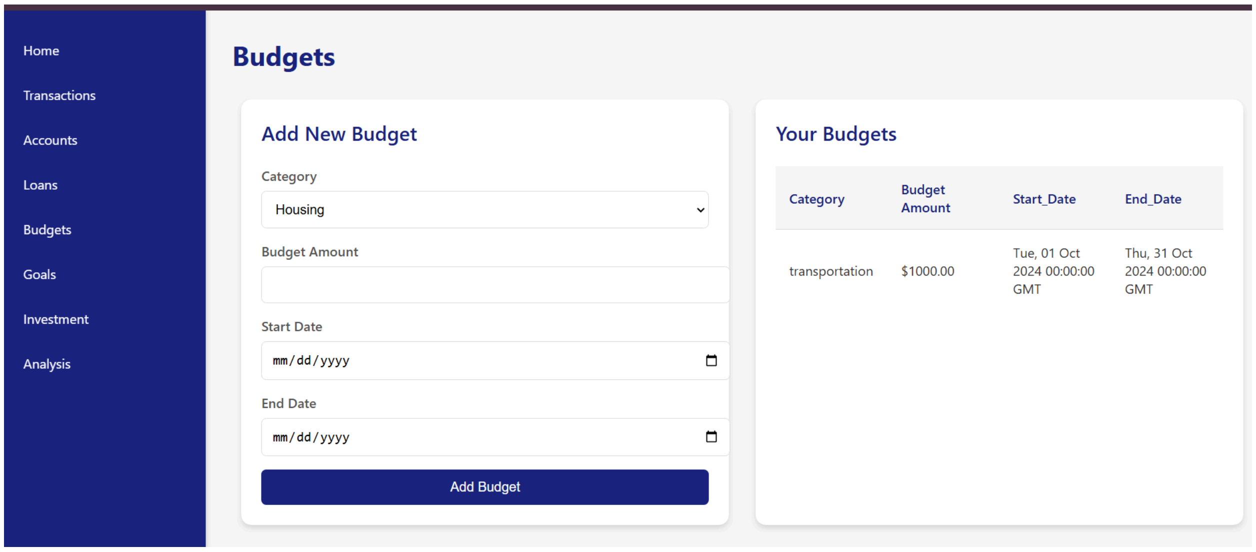
Budget Amount: 1000

Start Date: 10/01/2024

End Date: 10/31/2024

Your Budgets

Category	Budget Amount	Start_Date	End_Date



Budgets

Add New Budget

Category: Housing

Budget Amount:

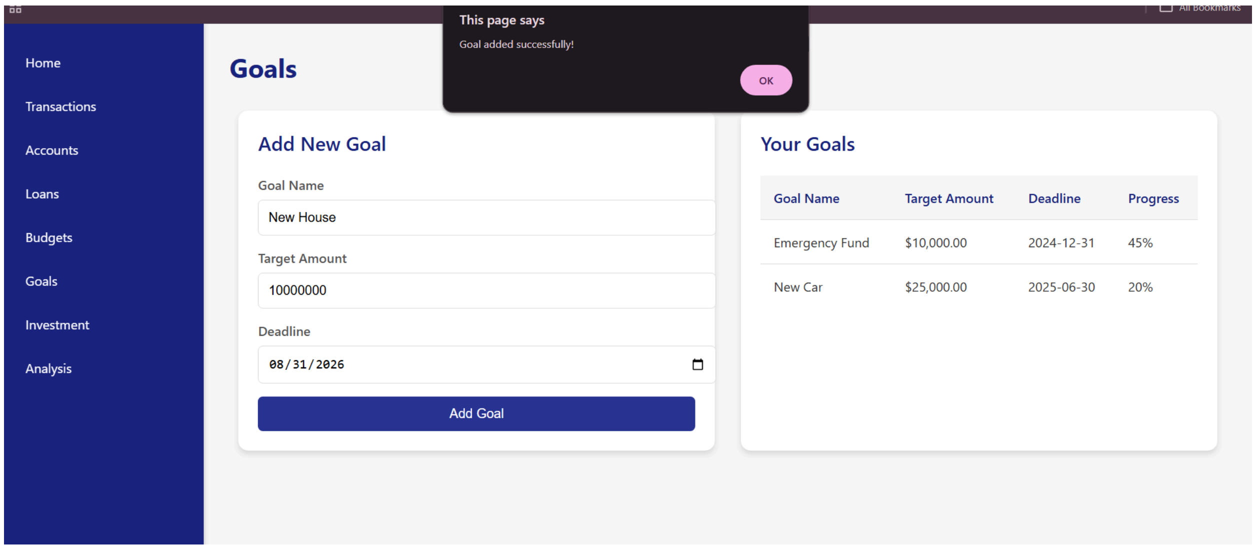
Start Date: mm/dd/yyyy

End Date: mm/dd/yyyy

Your Budgets

Category	Budget Amount	Start_Date	End_Date
transportation	$1000.00	Tue, 01 Oct 2024 00:00:00 GMT	Thu, 31 Oct 2024 00:00:00 GMT

b. Goals:



This page says

Goal added successfully!

Goals

Add New Goal

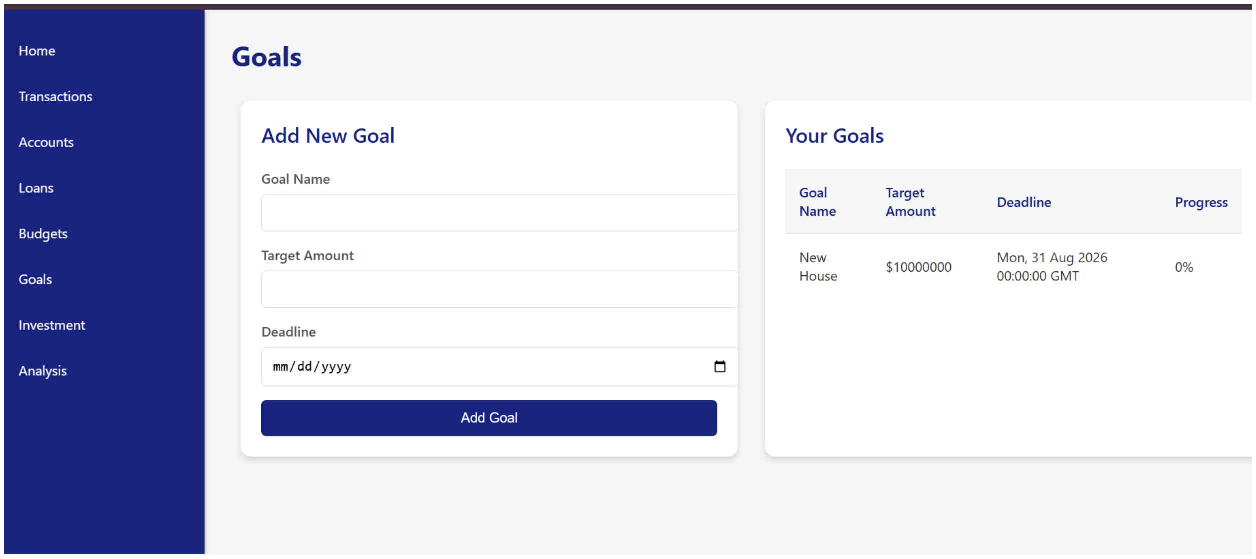
Goal Name: New House

Target Amount: 10000000

Deadline: 08/31/2026

Your Goals

Goal Name	Target Amount	Deadline	Progress
Emergency Fund	$10,000.00	2024-12-31	45%
New Car	$25,000.00	2025-06-30	20%



Goals

Add New Goal

Goal Name

Target Amount

Deadline

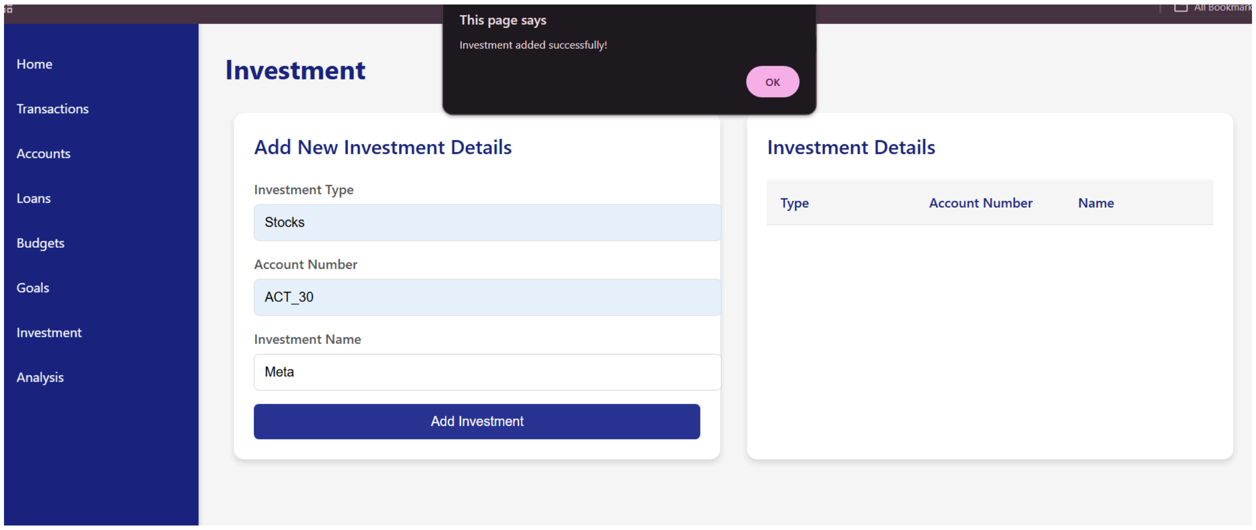
mm/dd/yyyy

Add Goal

Your Goals

Goal Name	Target Amount	Deadline	Progress
New House	$10000000	Mon, 31 Aug 2026 00:00:00 GMT	0%

c. Investment:



This page says

Investment added successfully!

OK

Investment

Add New Investment Details

Investment Type: Stocks

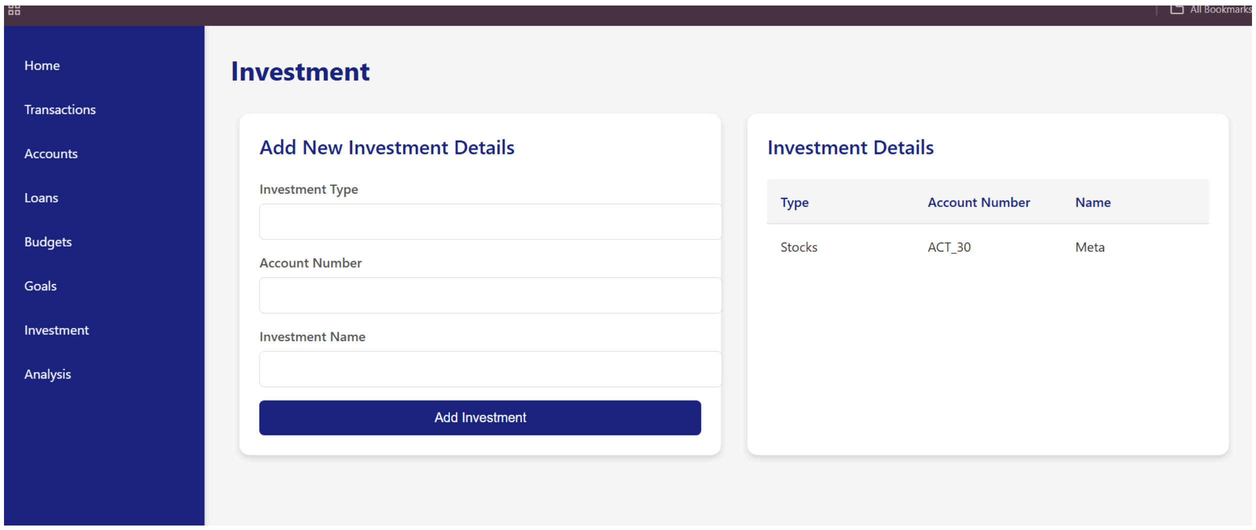
Account Number: ACT_30

Investment Name: Meta

Add Investment

Investment Details

Type	Account Number	Name



Investment

Add New Investment Details

Investment Type

Account Number

Investment Name

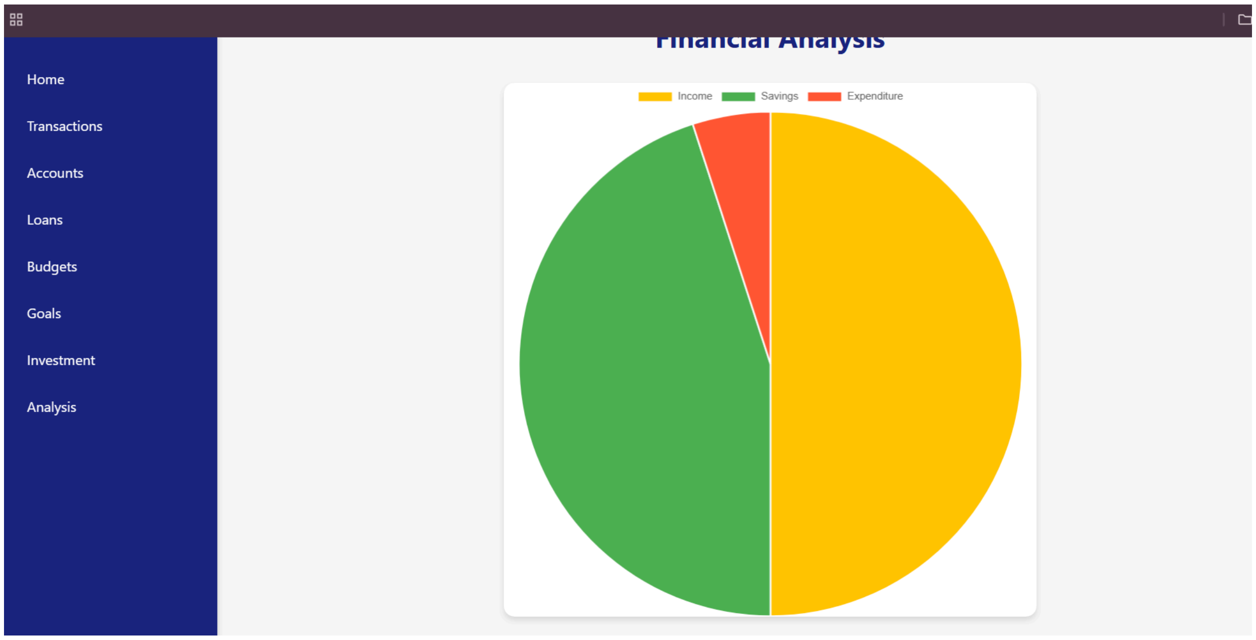
Investment Details

Type	Account Number	Name
Stocks	ACT_30	Meta

6. Analysis:

Once a user adds a transaction, the total incoming amount and total expenditure as well as savings is calculated through triggers. This data is reflected in the Analysis tab on the application in the form of a pie chart for better understanding of the user.

Here we have two transactions with income as $1000, expenditure as $100 and savings as $900



Overall our application is working beyond expectation and the performance time, refresh rate, query time remains in the green.

8. CHALLENGES

During the course of our project we faced several challenges in our application. The majority of our hurdles involved the HTML pages and linking them to the data APIs. We addressed these issues by writing error messages to a log and reading them through the console and backtracking to see where the API went wrong.

We had a few issues in calculation of field values using triggers which were investigated and resolved before implementing the backend.

Another issue we faced was synchronizing the database fetching with the content on the screen. It was a challenge to make changes appear after fetching from the database without doing a hard reload on the page. We researched action listeners to solve this problem and implemented them in our website. Although we had great success settling up and querying from the database, integrating it with a Flask application posed a learning curve for all of us where we furthered our knowledge.

9. CONCLUSION

The Personal Finance Tracker is a robust, scalable, and user-friendly application designed to address the financial management needs of individuals, students, and small business owners. By leveraging a carefully chosen set of tools and technologies, the project ensures seamless functionality, data integrity, and a superior user experience.The frontend of the application, developed with HTML, CSS, and JavaScript, offers an intuitive and responsive interface. These technologies ensure that users can easily interact with features like transaction recording, budget tracking, and financial goal monitoring across devices. JavaScript enhances interactivity, enabling dynamic visualizations such as pie charts and bar graphs for expense tracking and goal analysis.

The backend utilizes Flask, a lightweight Python framework, to handle server-side logic and connect the frontend with the MySQL database. Flask's modular design facilitates the efficient implementation of CRUD operations, recurring payment modules, and advanced filtering options. MySQL serves as the backbone of the database, ensuring secure and consistent management of financial data. With features like primary and foreign key constraints, triggers, and normalization, MySQL maintains data accuracy and minimizes redundancy, supporting the application's real-time updates and reliability. The design phase employs tools like MySQL Workbench for creating detailed Entity-Relationship (ER) diagrams, which model the database schema and relationships between entities such as users, transactions, and budgets. Wireframing tools are used to design a user-focused interface, emphasizing clarity and ease of use. Testing and development are conducted on a localhost server, allowing for the simulation of real-world scenarios and ensuring the system meets performance benchmarks such as fast load times and scalability to support at least 100 users. Through the integration of these technologies, the Personal Finance Tracker meets its objectives of delivering a reliable, efficient, and comprehensive financial management solution tailored to diverse user needs.

10. FUTURE SCOPE

There are several ways our financial application can be improved and have more features added. The first feature that could be added is AI powered financial insights. We can integrate machine learning models to give application users spending forecasting insights, spending advice, and investment recommendations. One important feature to integrate is data encryption. Since financial information is sensitive information, the data should be encrypted in the database to ensure data security for the users. Another feature that could be added is a mobile version of the application using Flutter. Having an independent mobile app is beneficial to have an effective user experience on the go. Multi Currency support is also a feature that would be worthwhile to implement in order to expand our user base overseas. By supporting multiple currencies, we can also integrate bank APIs to allow users to import and export financial information. More optional features that would be nice to have are better graphics for the website. A better layout and more appealing dashboard would make the user experience more enjoyable and attract more users for the application.