DBMS Project - UE22CS351A

Akash C (PES1UG22AM016) & Anuraag A Srivatsa (PES1UG22AM026)

Split

September 12, 2024

Overview

Split is a Bill Splitting Application which is a user-friendly and convenient tool designed to simplify the process of splitting bills fairly and efficiently among a group of people. It aims to reduce the hassle of manual calculations, ensuring a seamless and transparent experience for all participants.

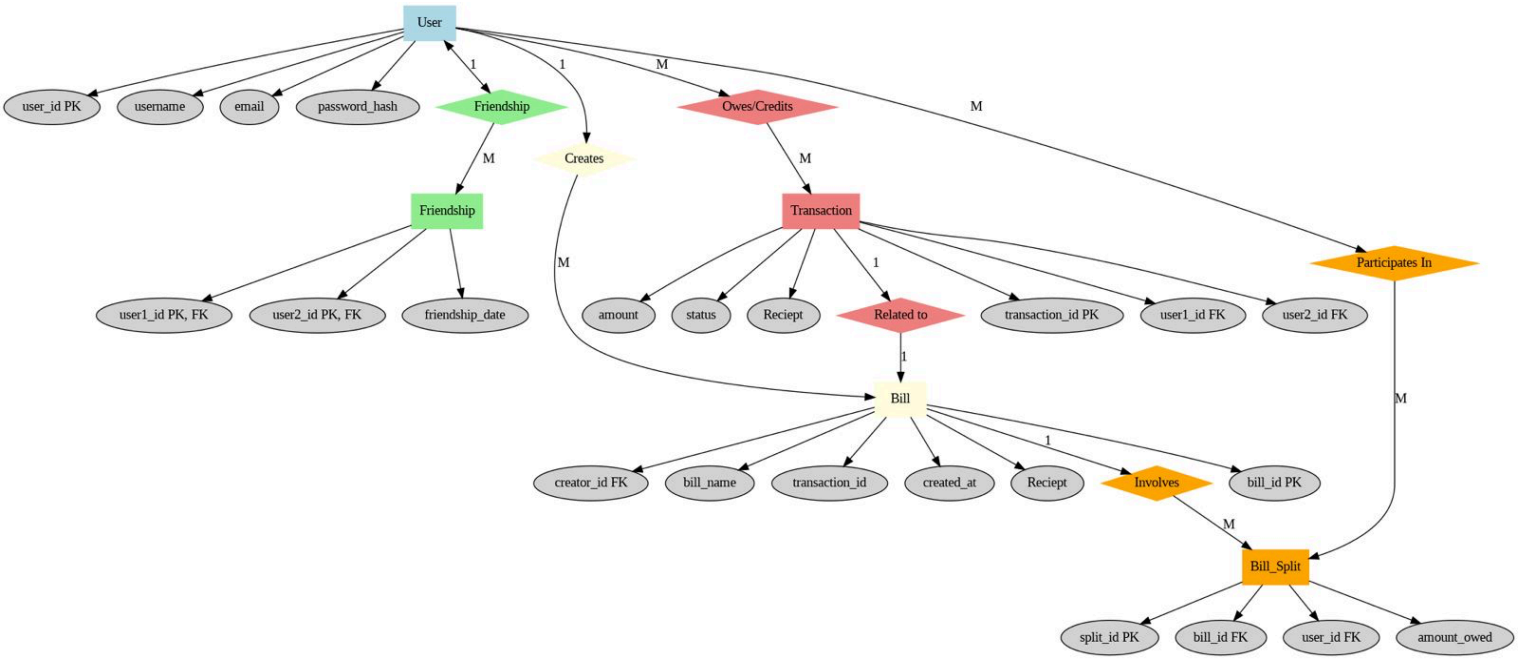
Goals

- Expense Splitting
  - Allows equal or unequal expense division, accurately calculating each participant's share based on their contribution or usage.
- User Authentication and Profile Management
  - Secure sign-up and login process with multi-factor authentication, including third-party authentication options for added convenience.
- Transaction History
  - Tracks all financial transactions, providing a comprehensive record of past expenses, payments, and settlements.
- Debt Settlement
  - Enables users to settle debts conveniently by marking them as paid or adjusting them against future expenses.

Technical Specifications

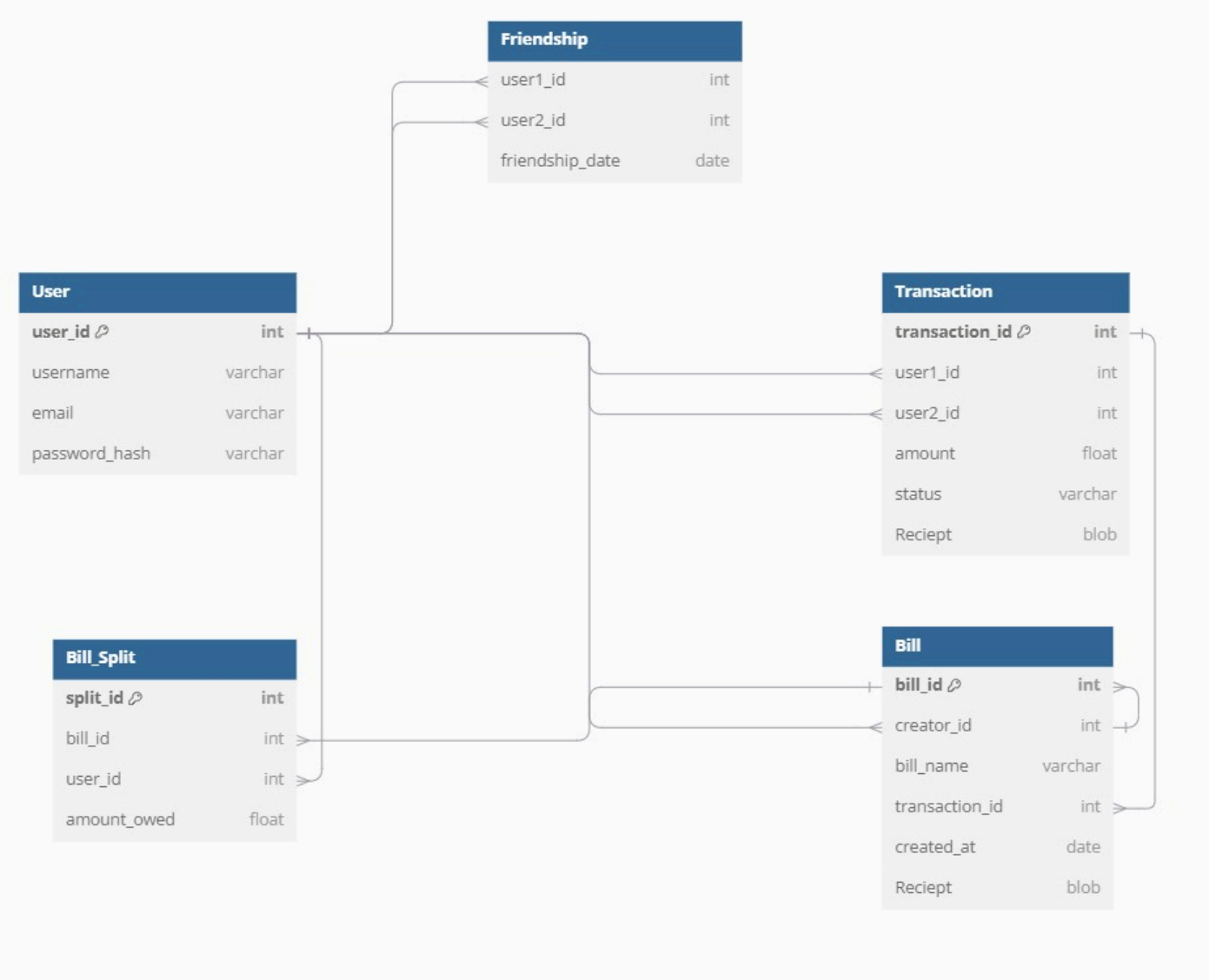
- Frontend
  - Built using cutting-edge React framework **NEXT.JS**, presenting a modern and responsive user interface.
- Backend
  - Powered by **Node.js** and integrated with a robust **PostgreSQL database**, ensuring efficient data storage and retrieval.
- Hosting
  - Hosted on **Vercel** for reliable performance, fast data access, and scalability.

ER Diagram



3

Relational Schema



DDL Commands

Below are the DDL Commands used for creating the Database

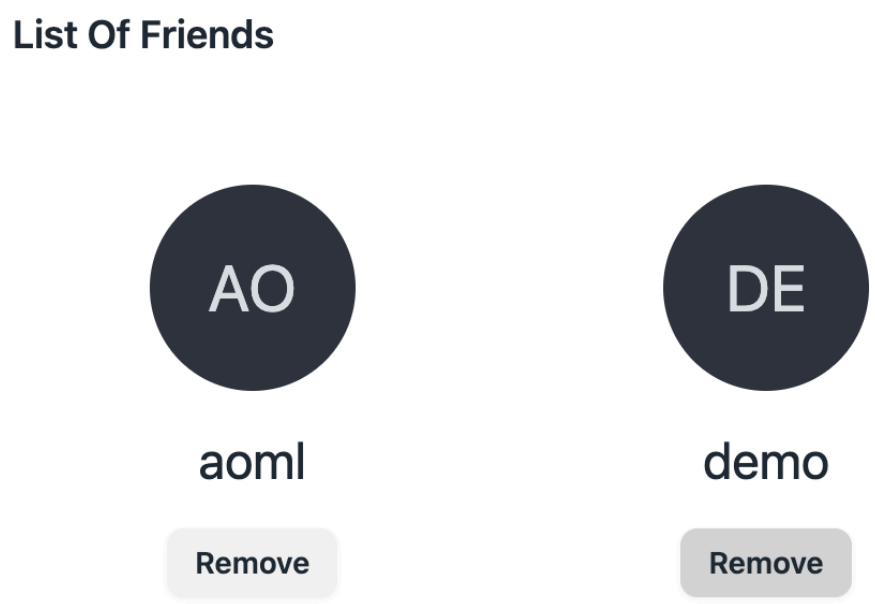
```
CREATE TABLE users (
            user_id SERIAL PRIMARY KEY,
            token_version integer DEFAULT 1,
            username varchar NOT NULL UNIQUE,
            email varchar NOT NULL UNIQUE,
            password varchar NOT NULL
        );
CREATE TABLE bills (
            bill_id SERIAL PRIMARY KEY,
            bill_name varchar(255) NOT NULL,
            amount DECIMAL(10, 2) NOT NULL,
            created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
            user_id INTEGER REFERENCES users(user_id)

        );
CREATE TABLE friends (
            user_id INT NOT NULL,
            friend_id INT NOT NULL,
            added_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
            PRIMARY KEY (user_id, friend_id),
            FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE,
            FOREIGN KEY (friend_id) REFERENCES users(user_id) ON DELETE CASCADE
        );
CREATE TABLE bill_participants (
            id SERIAL PRIMARY KEY,
            bill_no INT REFERENCES bills(bill_no),
            user_id INT REFERENCES users(user_id),
            amount_owed DECIMAL(10, 2),
            settled BOOLEAN DEFAULT FALSE
        );
CREATE TABLE expenses (
            expense_id SERIAL PRIMARY KEY,
            bill_id INT NOT NULL,
            split_id INT,
            user_id INT NOT NULL,
            amount DECIMAL(10, 2) NOT NULL,
            created_at TIMESTAMP DEFAULT NOW(),
```

```
            CONSTRAINT fk_bill FOREIGN KEY (bill_id) REFERENCES
bills(bill_id) ON DELETE CASCADE,
            CONSTRAINT fk_user FOREIGN KEY (user_id) REFERENCES users(id) ON
DELETE CASCADE
        );
```

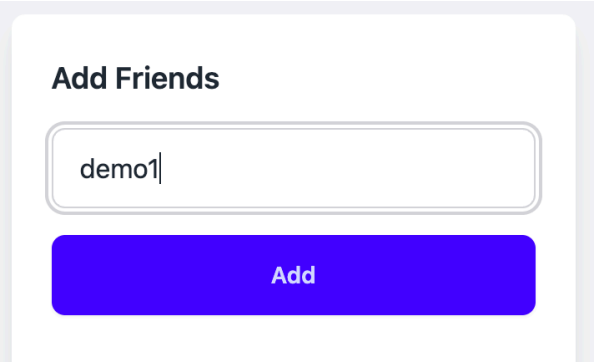
CRUD Operations Screenshot

1. Friends

Delete Friends

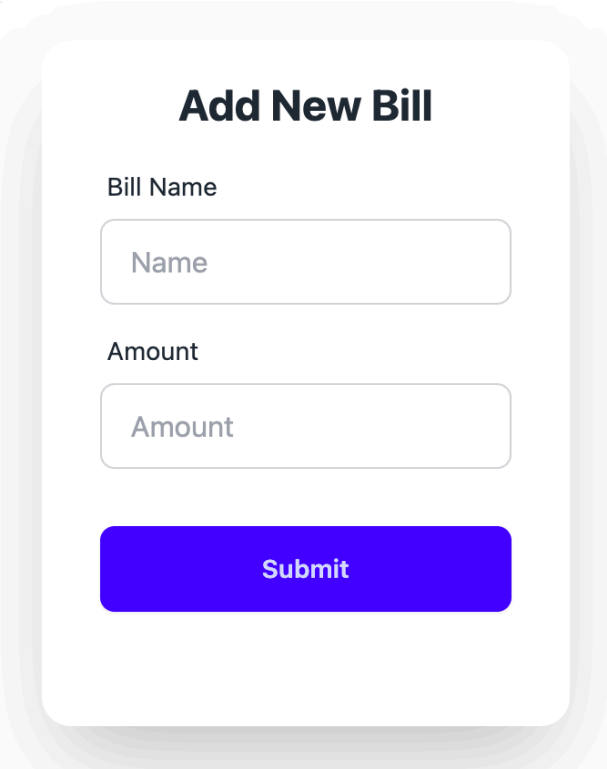


Add friends

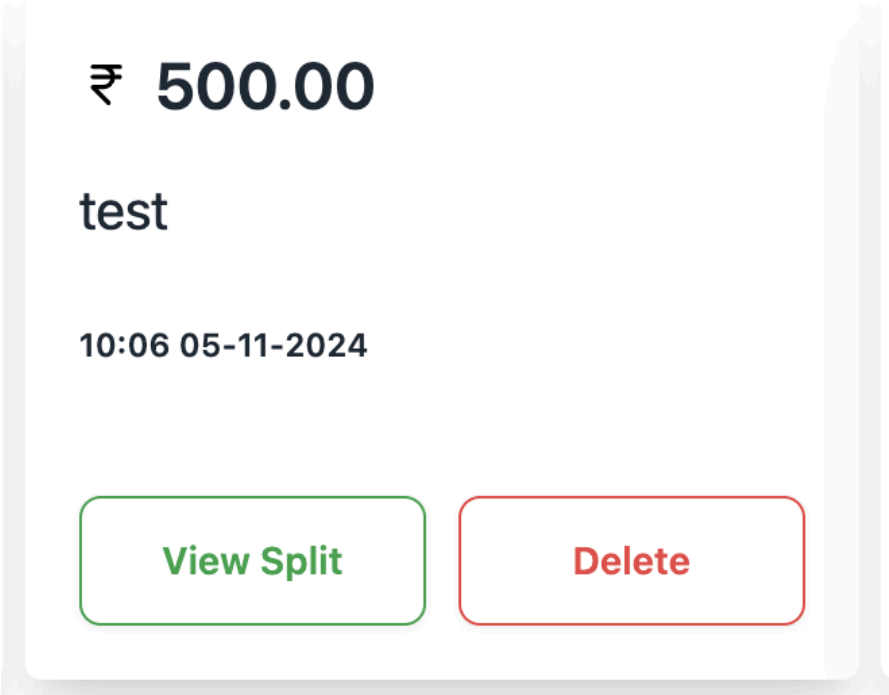


2. Bills

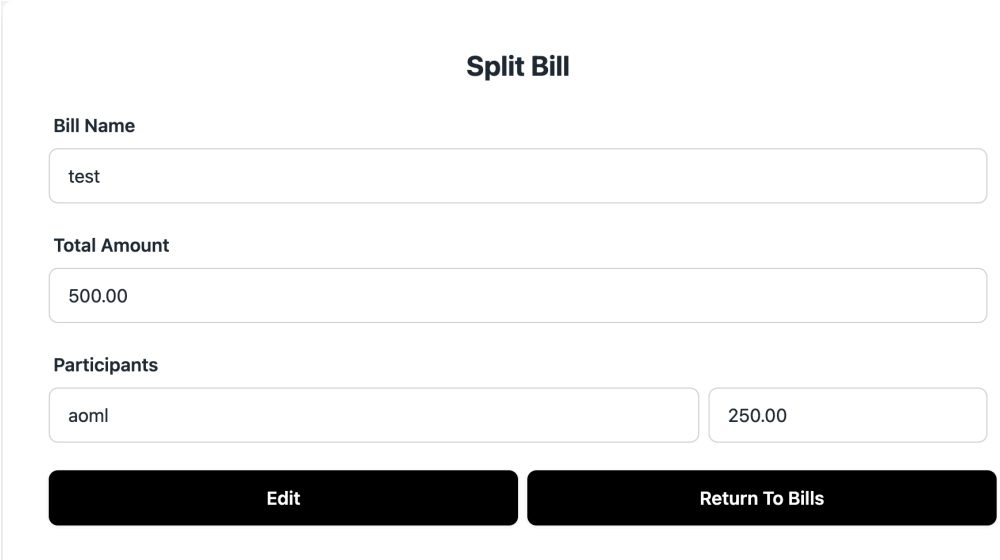
Create Bill



Delete Bill

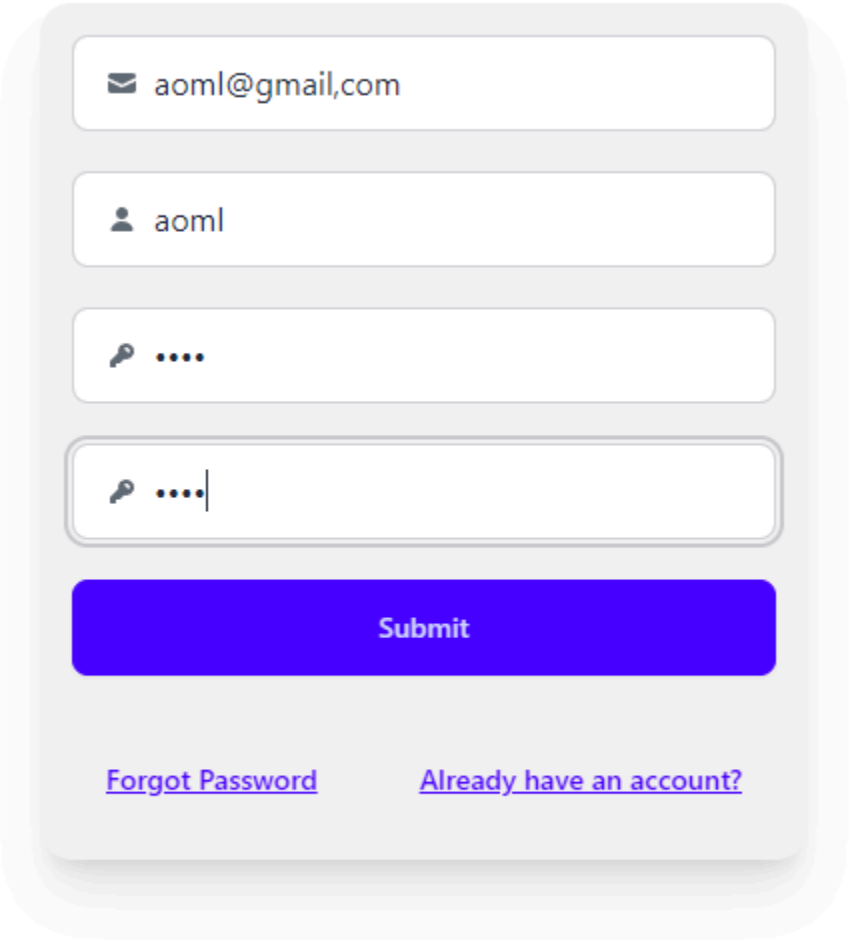


Update Bill

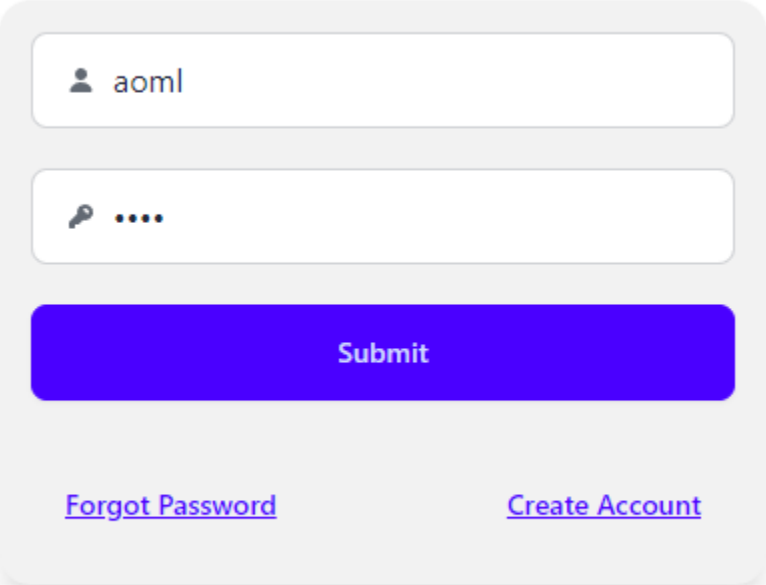


Functionalities of Application

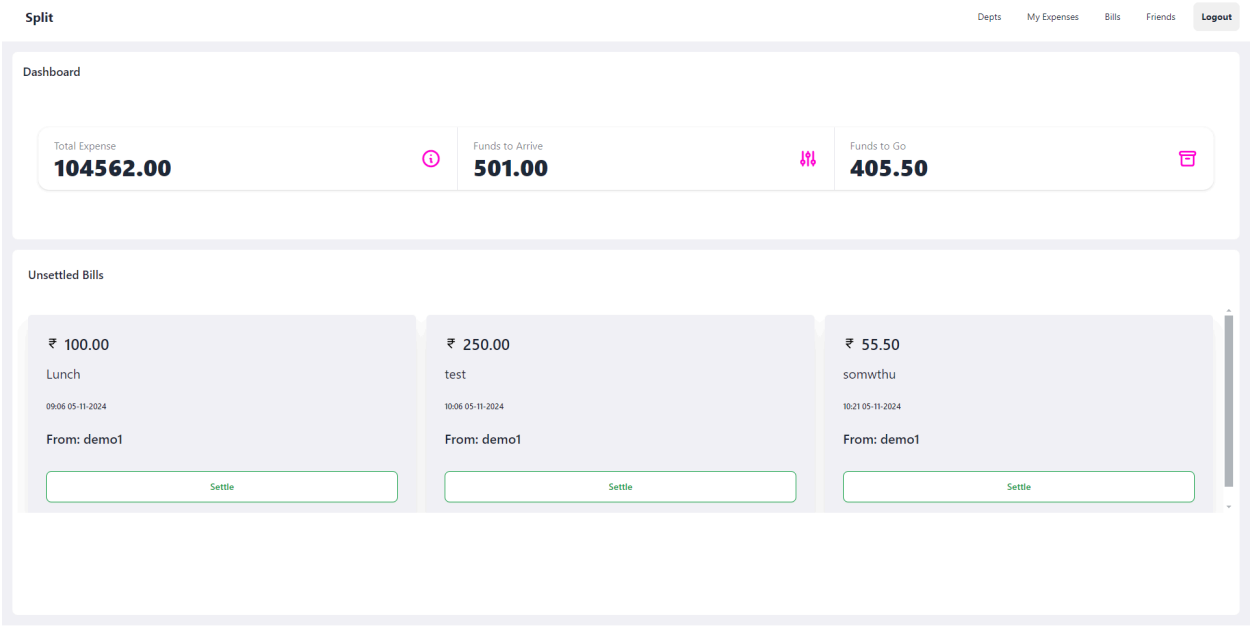
Signup : Page to create a user



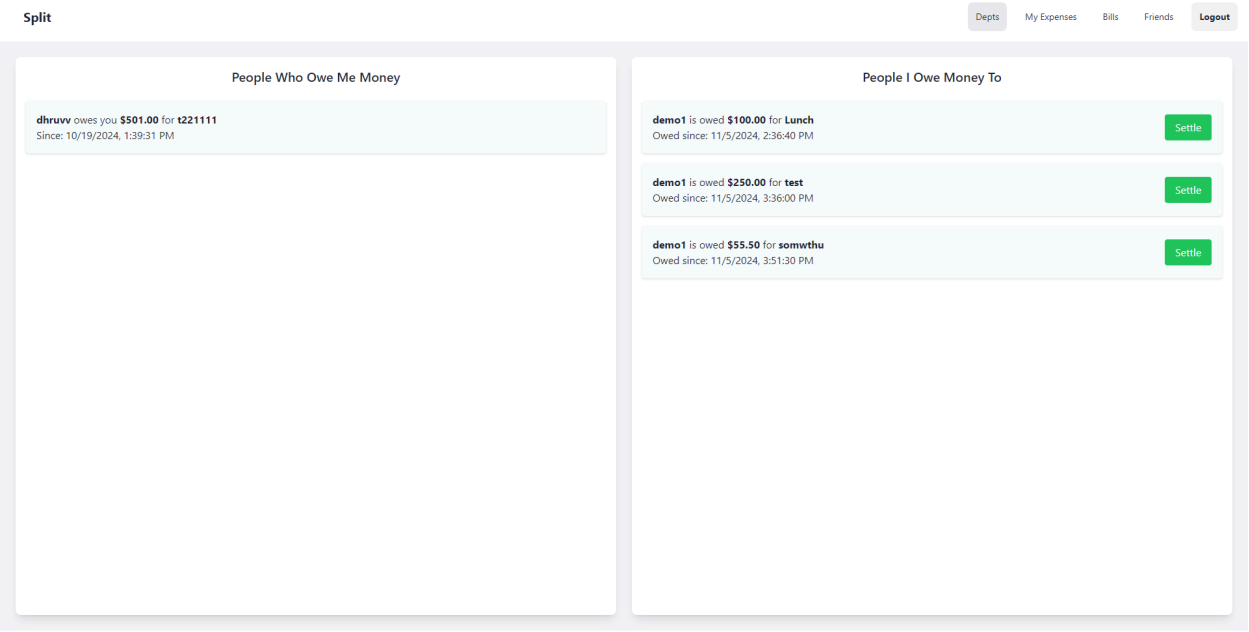
Login : Page to login



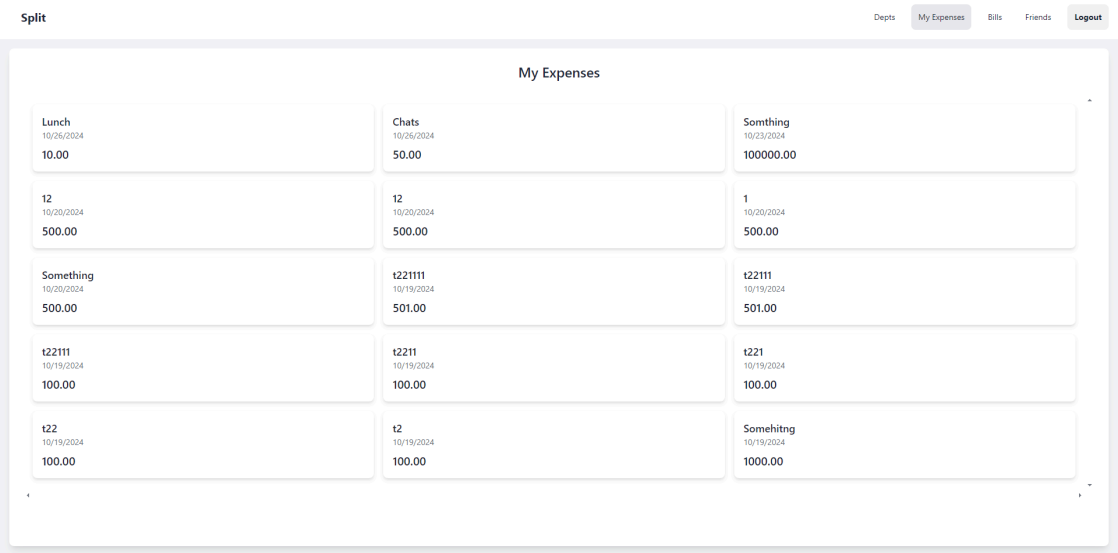
Dashboard : Page with aggregation of all the information about debts and expenses to the user



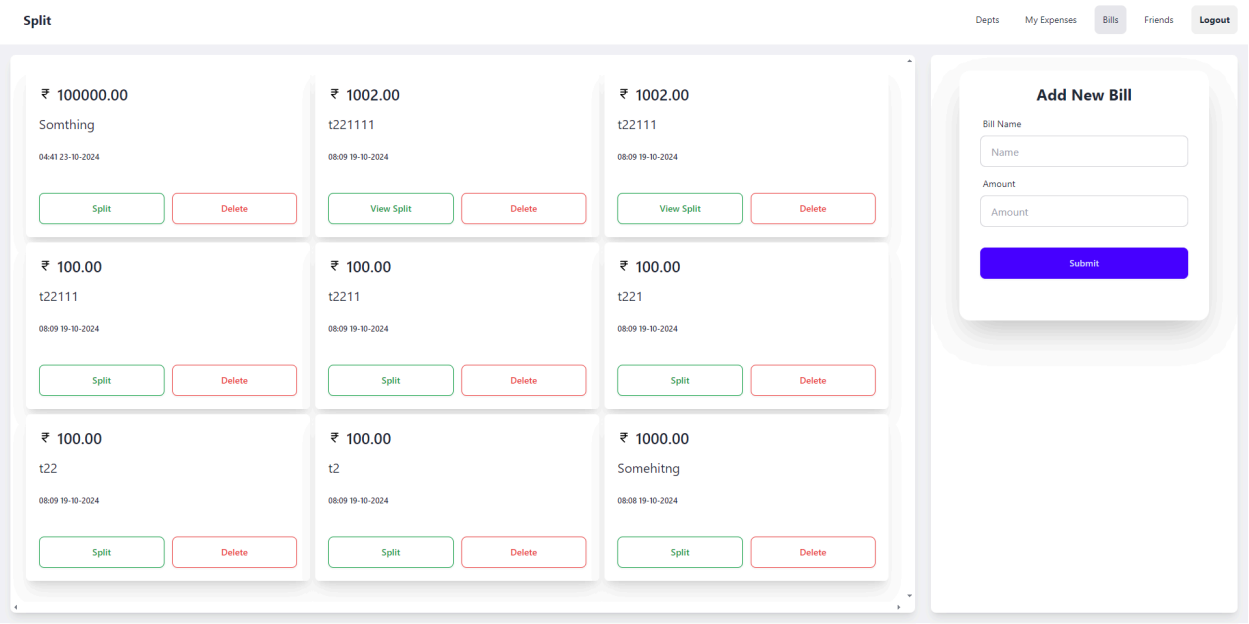
Debts : Page to view existing debts and can be settled here



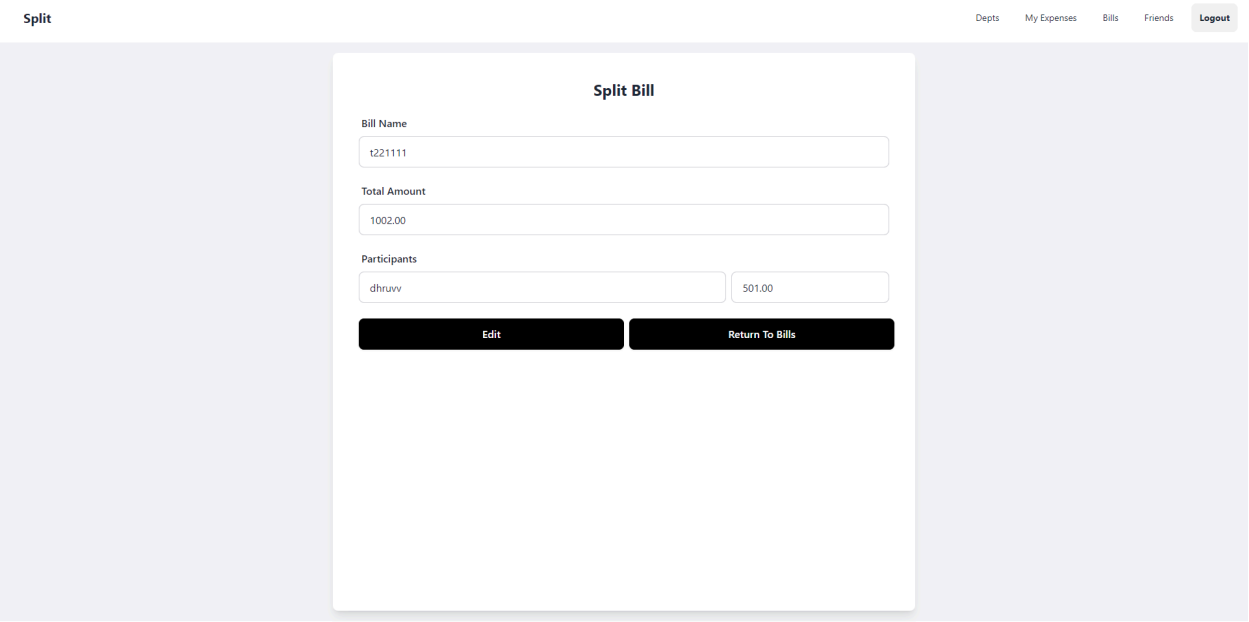
Expense History : Page to track user's expense history



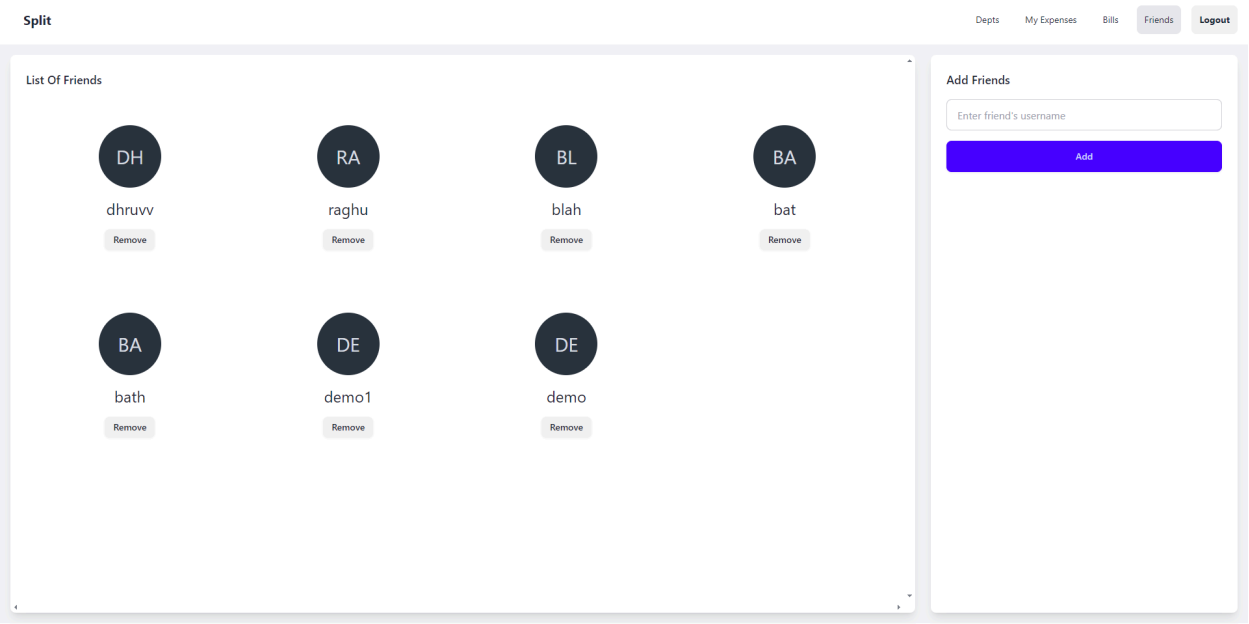
Bills : Page where you can see the existing bills and add new bills



Split Bill : Page where you can create split for a bill, can also be used to see the existing split and updating it



Friends : Page where you can add or remove friends using the userid



Triggers, Procedures / Functions, Nested Query, Join, Aggregate queries

```
CREATE OR REPLACE FUNCTION insert_into_expense_on_settle()
RETURNS TRIGGER AS $$
BEGIN

          IF NEW.settled = true THEN
             INSERT INTO expenses (bill_id, split_id, user_id, amount)
             VALUES (
             NEW.bill_id,
             NEW.id,
             NEW.user_id,
             NEW.amount_owed
             );
          END IF;

RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER after_settle_update
AFTER UPDATE OF settled ON bill_participants
FOR EACH ROW
WHEN (NEW.settled = true)
EXECUTE FUNCTION insert_into_expense_on_settle();
```

Created a Function to manage expenses table and a Trigger calling this function whenever there are changes made to bill participants (Ie, bills being settled)

```
CREATE OR REPLACE FUNCTION update_expenses_on_bills()
RETURNS TRIGGER AS $$
BEGIN
    IF TG_OP = 'INSERT' THEN
        INSERT INTO expenses (bill_id, user_id, amount)
        VALUES (NEW.bill_id, NEW.user_id, NEW.amount);
        RETURN NEW;

    ELSIF TG_OP = 'DELETE' THEN
```

```
        DELETE FROM expenses WHERE bill_id = OLD.bill_id;
        RETURN OLD;

    END IF;

    RETURN NULL;
    EXCEPTION
    WHEN OTHERS THEN
        INSERT INTO public.trigger_logs (log_message)
        VALUES (FORMAT('Error: %s', SQLERRM));
        RETURN NULL;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER bills_trigger
AFTER INSERT OR DELETE ON bills
FOR EACH ROW
EXECUTE FUNCTION update_expenses_on_bills();
```

Created a Function to manage expenses table and a Trigger calling this function whenever there are changes made to bills

```
SELECT SUM(bp.amount_owed) AS owed
FROM bill_participants bp
WHERE bp.bill_id IN (
    SELECT b.bill_id
    FROM bills b
    JOIN users u ON b.user_id = u.user_id
    WHERE bp.user_id = $1 AND bp.settled = false
) AND bp.settled = false;
```

Join / Aggregation

```
        SELECT SUM(e.amount) AS total_amount
        FROM expenses e
        JOIN bills b ON e.bill_id = b.bill_id
        WHERE e.user_id = $1
```

```
        SELECT SUM(bp.amount_owed) as sum
        FROM bills b
        JOIN bill_participants bp ON b.bill_id = bp.bill_id
        JOIN users u ON u.user_id = bp.user_id
        WHERE b.user_id = $1 AND bp.amount_owed > 0 AND bp.settled = false
```

```
        SELECT
            bill_participants.id,
            bill_participants.user_id,
            users.username,
            bill_participants.amount_owed,
            bill_participants.settled
        FROM bill_participants
        JOIN users ON bill_participants.user_id = users.user_id
        WHERE bill_participants.bill_id = $1
```

```
        SELECT u.username AS "From", bp.amount_owed AS amount, b.bill_name
AS bill, b.created_at AS time, bp.id
        FROM bill_participants bp
        JOIN bills b ON bp.bill_id = b.bill_id
        JOIN users u ON b.user_id = u.user_id
        WHERE bp.user_id = $1 AND bp.settled = false
        ORDER BY b.created_at;
```

These are only some of the queries

All queries are in the split.sql file submitted seperately

Code Snippets for invoking functions and trigger

1.

```
UPDATE bill_participants SET settled = true WHERE id = $1 AND user_id = $2
```

This code will settle the bill, which will trigger a function call to update information on the expense history table

2.

```
DELETE FROM bills WHERE bill_id = $1
```

This code will delete an existing bill , which will trigger a function call to update information on the expense history table. (Ie, removing that expense from the expense)

Github Repo Link : https://github.com/Split-Simplify-Group-Expenses/split

Live hosted Website : https://expensesplit.vercel.app/

Conclusion

The Bill Splitting Application is a comprehensive solution for managing shared expenses, catering to a wide range of users. Its focus on simplicity, security, and reliability makes it an invaluable tool for individuals and groups looking to streamline their financial management and promote financial transparency.