Project Deliverable 3

1. DESCRIPTION OF IMPLEMENTATION

Database Schema

- Tables Created:
- User: Stores user details such as SSN, name, and primary contact information.
- BankAccount: Contains linked bank account details for each user.
- Transaction: Records all transactions with sender, receiver, amount, date, and type (send/request).
- EmailAddress: Stores multiple email addresses for each user.
- PhoneNumber: Stores multiple phone numbers for each user.
- Statement: Aggregates user transactions into summaries.
- Primary Keys:

  SSN for User, AccountID for BankAccount, TransactionID for Transaction, EmailID for EmailAddress, PhoneID for PhoneNumber, StatementID for MonthlyStatement.
- Foreign Keys:

  SSN in BankAccount, Transaction, EmailAddress, PhoneNumber referencing User.

  TransactionID in MonthlyStatement referencing Transaction.

Data Population

- SQL command files populated tables with sample data, maintaining relationships:
- 10 users, each with at least one email, phone number, and bank account.
- 50 transactions distributed across users.
- Monthly statements calculated from the Transaction table.

Application System

Developed a menu-driven application with the following functionality:

Main Menu

1. Account Info: Displays user details and linked accounts.
2. Send Money:

- Inputs: Recipient phone/email, amount.

- Validations: Verify sufficient balance, existing recipient details.
- Actions: Deduct from sender, credit to recipient.

3. Request Money:

- Inputs: Requester details, amount.
- Validations: Ensure the recipient exists in the system.
- Actions: Record the request and notify the user.

4. Statements:

- summaries of sent/received amounts.

5. Search Transactions:

- Query filters: SSN, email, phone, type, date/time range.

6. Sign Out: Ends the session.

Account Functions

1. Modify user details like name or address.
2. Add/Remove linked emails and phone numbers.
3. Manage linked bank accounts (add/remove).

Statement Functions

1. Total money sent/received within a date range.
2. Aggregate statistics (monthly totals and averages).
3. Identify largest transaction amounts.
4. Highlight best users by transaction value.

2. PROBLEMS FACED AND SOLUTIONS

Problem 1: Data Integrity

- Issue: Referential integrity violations when populating tables.
- Solution: Used cascading updates/deletes in foreign key constraints and validated data sequences during population scripts.

Problem 2: Complex Queries

- Issue: Generating monthly summaries and identifying top users required multi-level aggregation.

- Solution: Optimized queries using GROUP BY, ORDER BY, and subqueries. Created indexed views for performance improvement.

Problem 3: Application Menu Navigation

- Issue: Handling menu inputs dynamically for multiple operations was complex.
- Solution: Implemented a switch-case structure in the code for clear navigation and created reusable functions for repeated actions.

Problem 4: Transaction Validations

- Issue: Ensuring real-time validation for balances, valid accounts, and recipient details.
- Solution: Implemented stored procedures with necessary checks to execute transactional updates atomically.

Problem 5: Concurrent Access

- Issue: Simultaneous operations by multiple users caused deadlock errors.
- Solution: Used database transaction isolation levels and locks to prevent conflicts.

3) COMMANDS TO CREATE TABLE

1)users table

```
CREATE TABLE IF NOT EXISTS users (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    email TEXT UNIQUE NOT NULL,
    phone TEXT UNIQUE NOT NULL,
    ssn TEXT UNIQUE NOT NULL,
    password TEXT NOT NULL
  )
```

2) additional emails table

```
CREATE TABLE IF NOT EXISTS user_emails (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    user_id INTEGER NOT NULL,
    email TEXT NOT NULL UNIQUE,
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE
  )
  ''')
```

3)phones table

```
CREATE TABLE IF NOT EXISTS user_phones (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    user_id INTEGER NOT NULL,
```

```
        phone TEXT NOT NULL UNIQUE,
        FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE
    )
```

4) Bank Account

```
CREATE TABLE IF NOT EXISTS bank_accounts (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        user_id INTEGER NOT NULL,
        account_number TEXT NOT NULL UNIQUE,
        routing_number TEXT NOT NULL,
        phone TEXT NOT NULL,
        FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE
    )
    ''')
```

5) transactions

```
CREATE TABLE IF NOT EXISTS transactions (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        sender_id INTEGER NOT NULL,
        receiver_id INTEGER NOT NULL,
        amount REAL NOT NULL,
        date TEXT NOT NULL,
        transaction_type TEXT NOT NULL,
        FOREIGN KEY (sender_id) REFERENCES users(id) ON DELETE CASCADE,
        FOREIGN KEY (receiver_id) REFERENCES users(id) ON DELETE CASCADE
    )
    ''')
```

6) best users table

```
CREATE TABLE IF NOT EXISTS best_users (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        user_id INTEGER NOT NULL,
        total_sent REAL DEFAULT 0,
        total_received REAL DEFAULT 0,
        FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE
    )
    ''')
```

7) creating indices

```
cursor.execute('CREATE INDEX IF NOT EXISTS idx_sender ON transactions (sender_id)')
    cursor.execute('CREATE INDEX IF NOT EXISTS idx_receiver ON transactions (receiver_id)')
    cursor.execute('CREATE INDEX IF NOT EXISTS idx_date ON transactions (date)')
```

4) SQL COMMANDS TO POPULATE TABLE

1) populate best users table

```
INSERT OR IGNORE INTO best_users (user_id, total_sent, total_received)
    SELECT id, 0, 0 FROM users
    ''')
```

2) Sign up

```
conn.execute('INSERT INTO users (name, email, phone, ssn, password) VALUES (?, ?, ?, ?, ?)',
                    (f"{first_name} {last_name}", email, phone, ssn, password))
```

3)modify details

```
conn.execute('UPDATE users SET name = ?, email = ?, phone = ? WHERE id = ?',
                (name, email, phone, session['user_id']))
        conn.commit()
```

4) modify bank account

```
existing_account = conn.execute('SELECT * FROM bank_accounts WHERE user_id = ?',
(session['user_id'],)).fetchone()
        if existing_account:
            conn.execute('UPDATE bank_accounts SET account_number = ?, routing_number = ?, phone = ?
WHERE user_id = ?',
                    (account_number, routing_number, phone, session['user_id']))
            flash("Bank account updated successfully.")
        else:
            conn.execute('INSERT INTO bank_accounts (user_id, account_number, routing_number, phone)
VALUES (?, ?, ?, ?)',
                    (session['user_id'], account_number, routing_number, phone))
            flash("Bank account added successfully.")
```

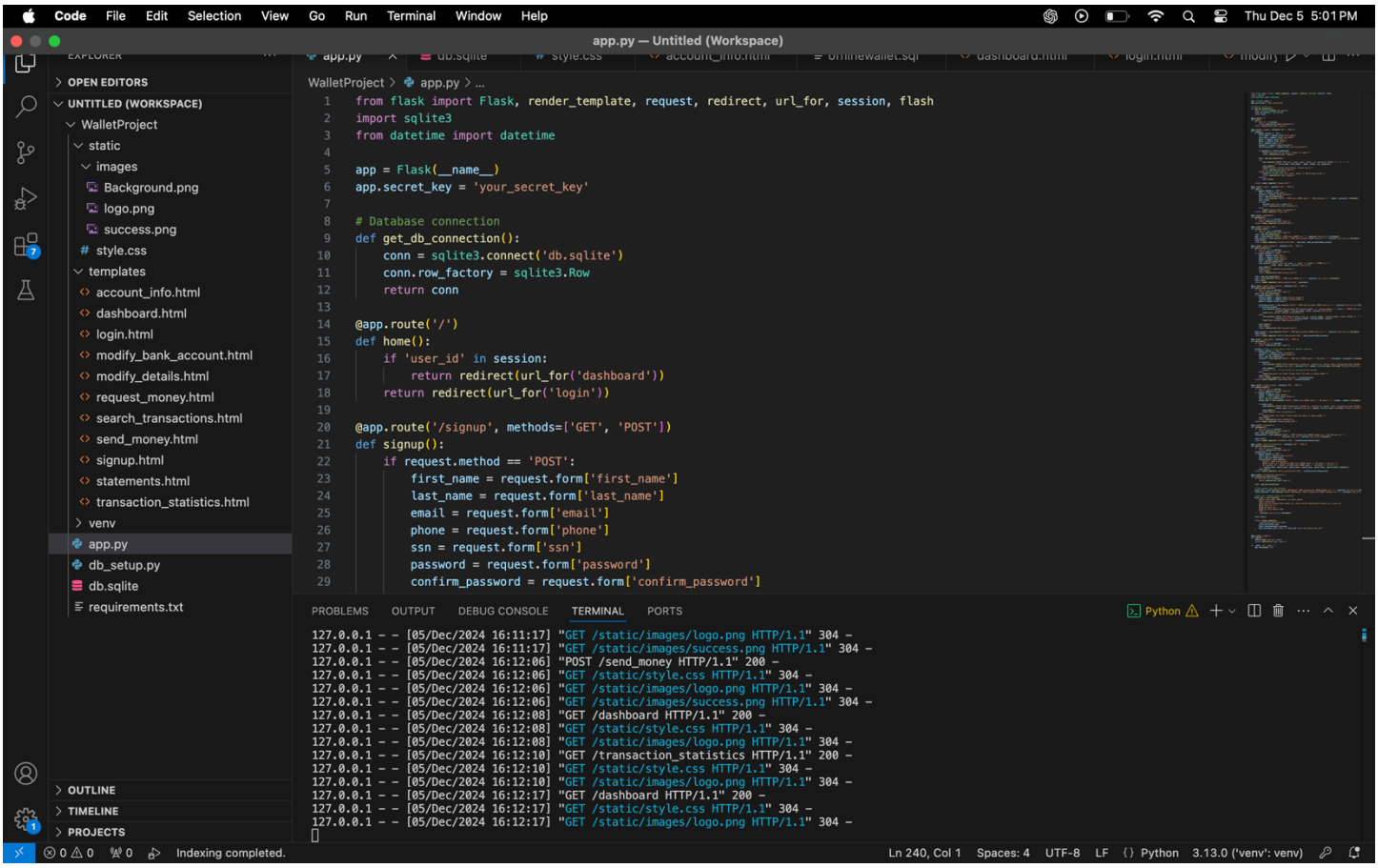
5) Send money

```
if receiver:
            conn.execute('INSERT INTO transactions (sender_id, receiver_id, amount, date, transaction_type)
VALUES (?, ?, ?, ?, ?)',
                    (session['user_id'], receiver['id'], amount, datetime.now().strftime('%Y-%m-%d %H:%M:%S'),
'send'))
```

6) Request money

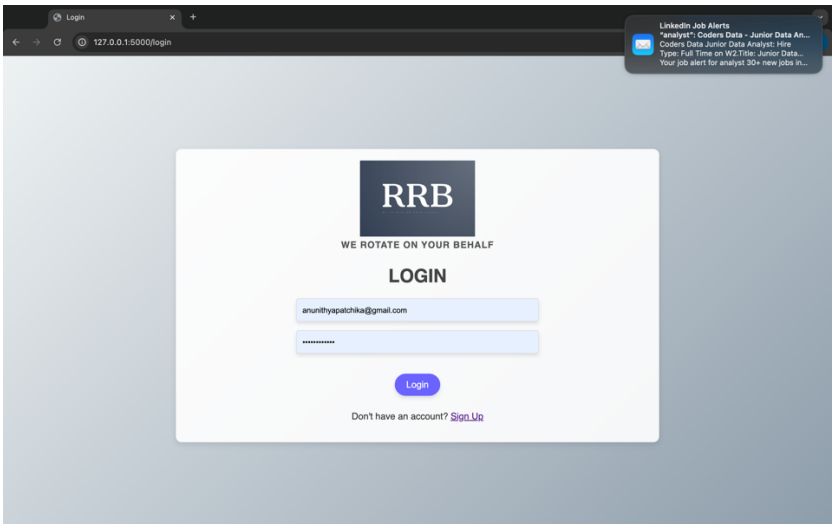
```
f sender_user:
            conn.execute('INSERT INTO transactions (sender_id, receiver_id, amount, date, transaction_type)
VALUES (?, ?, ?, ?, ?)',
                    (sender_user['id'], session['user_id'], amount, datetime.now().strftime('%Y-%m-%d
%H:%M:%S'), 'request'))
```

SCREENSHOT OF USE OF THE PROGRAM

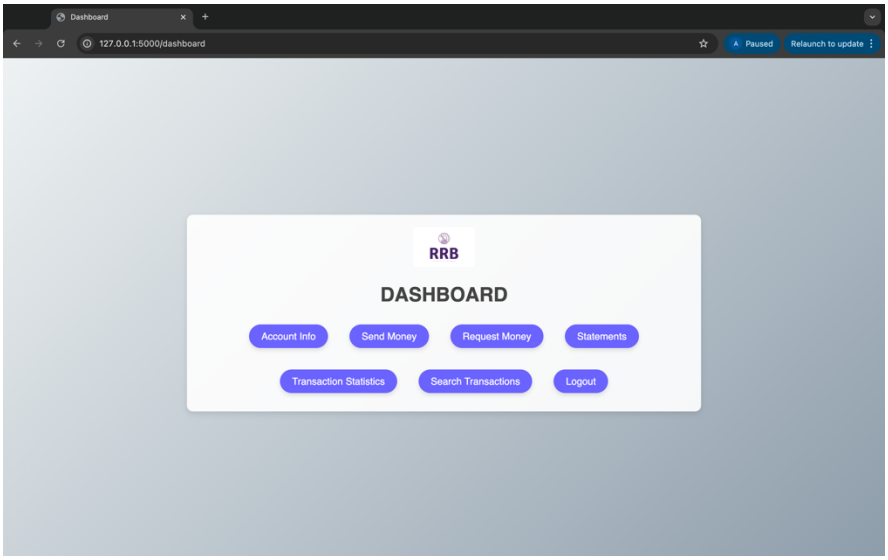


SCREENSHOTS OF APPLICATION RUNNING

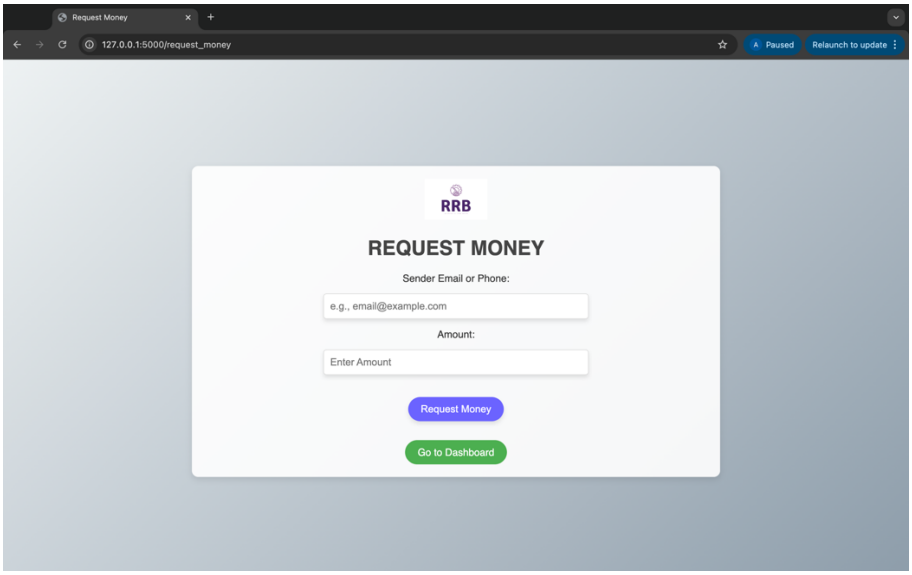
1) LOGIN PAGE



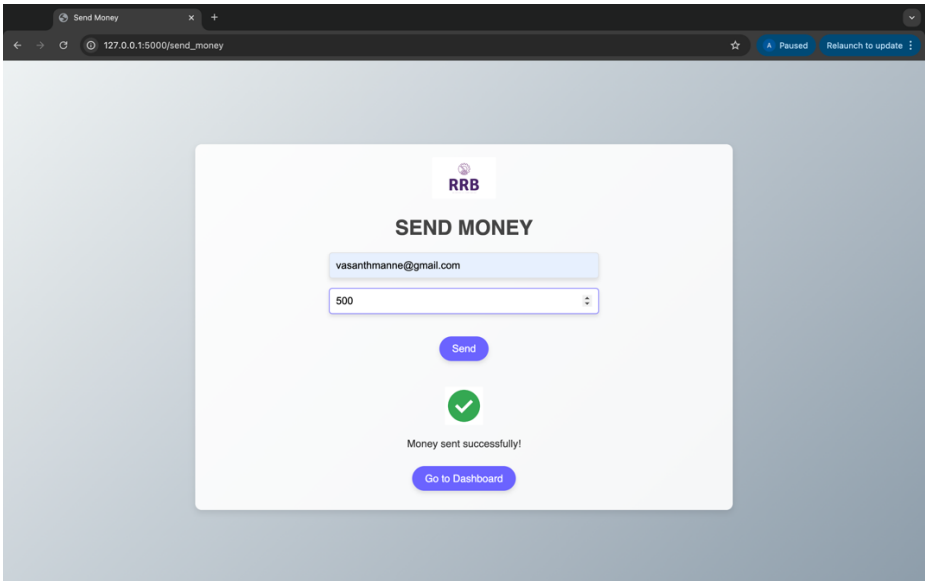
2) DASHBOARD



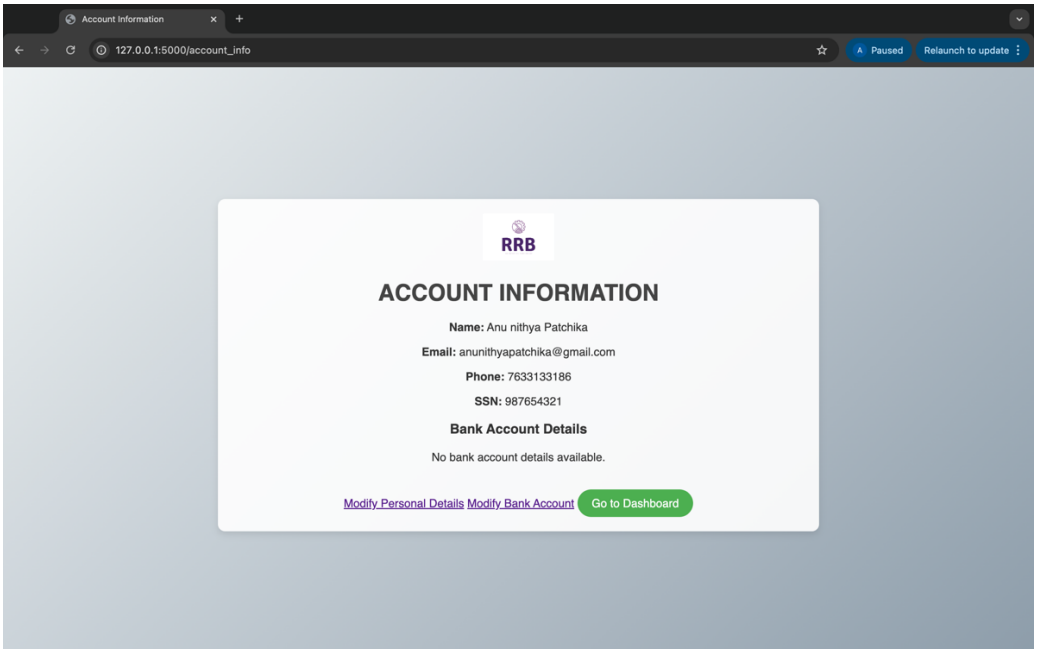
3) REQUEST MONEY



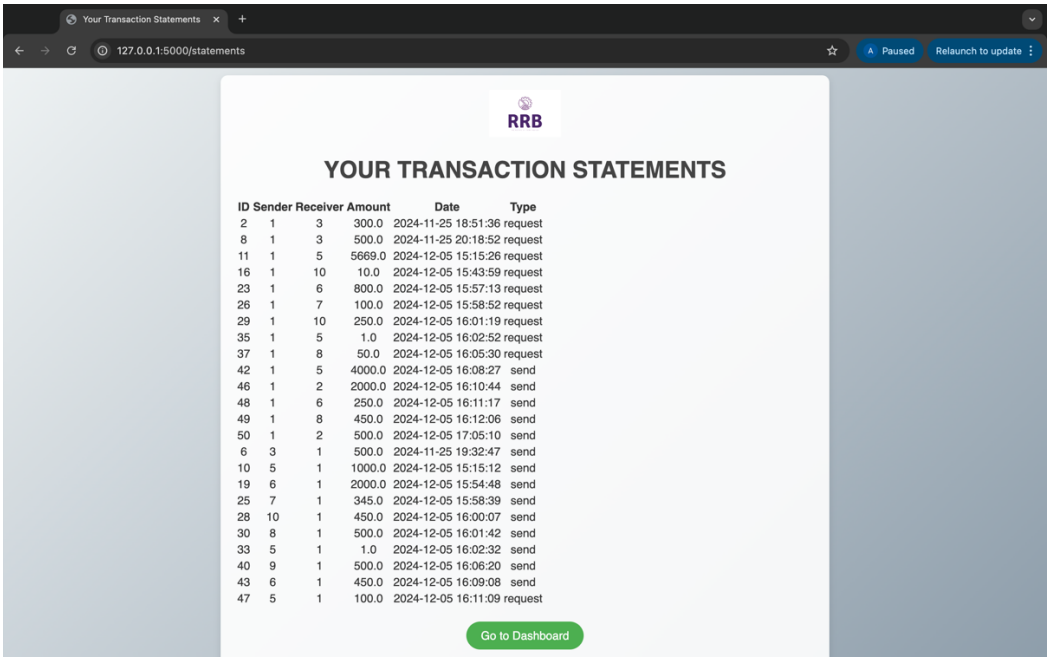
4)SEND MONEY



5)ACCOUNT INFORMATION

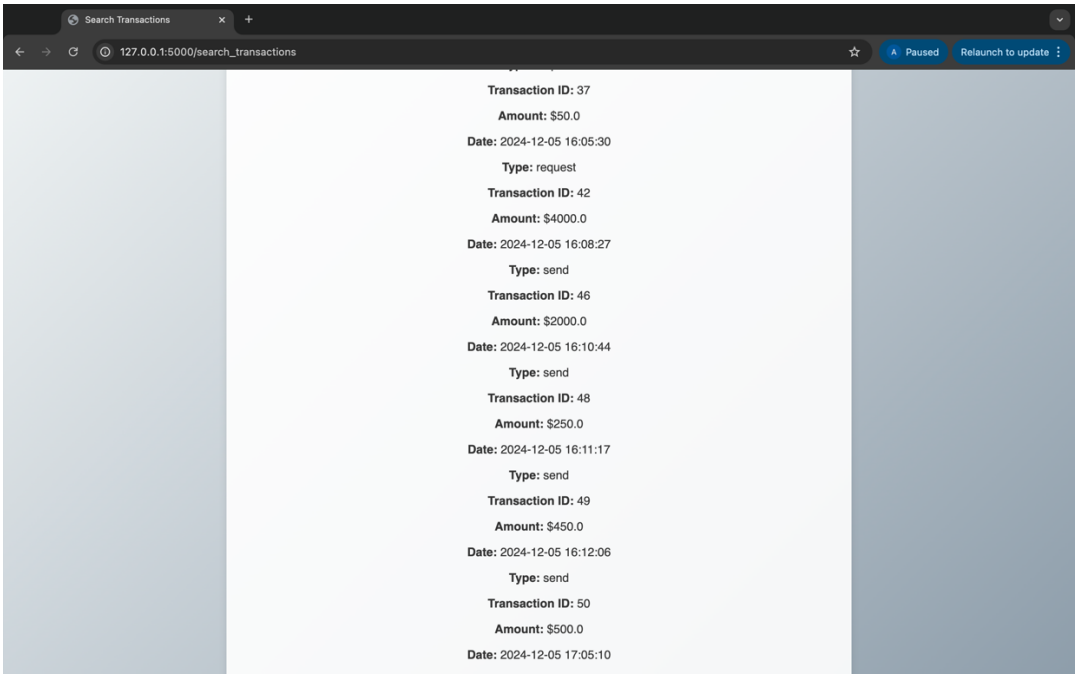


6) TRANSACTION STATEMENTS

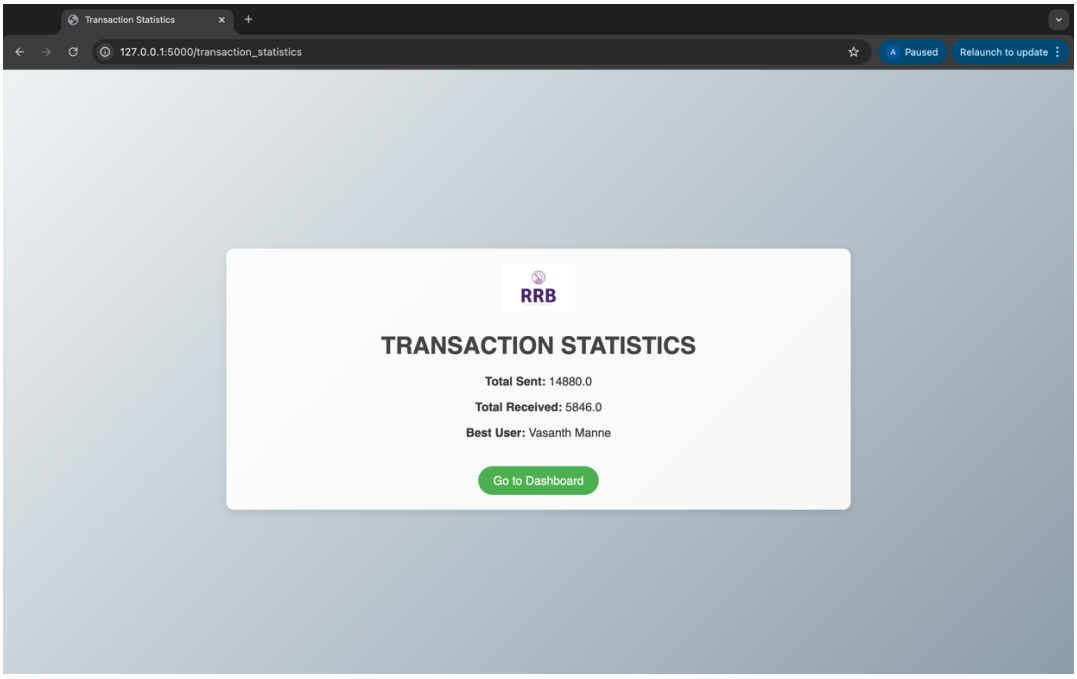


ID	Sender	Receiver	Amount	Date	Type
2	1	3	300.0	2024-11-25 18:51:36	request
8	1	3	500.0	2024-11-25 20:18:52	request
11	1	5	5669.0	2024-12-05 15:15:26	request
16	1	10	10.0	2024-12-05 15:43:59	request
23	1	6	800.0	2024-12-05 15:57:13	request
26	1	7	100.0	2024-12-05 15:58:52	request
29	1	10	250.0	2024-12-05 16:01:19	request
35	1	5	1.0	2024-12-05 16:02:52	request
37	1	8	50.0	2024-12-05 16:05:30	request
42	1	5	4000.0	2024-12-05 16:08:27	send
46	1	2	2000.0	2024-12-05 16:10:44	send
48	1	6	250.0	2024-12-05 16:11:17	send
49	1	8	450.0	2024-12-05 16:12:06	send
50	1	2	500.0	2024-12-05 17:05:10	send
6	3	1	500.0	2024-11-25 19:32:47	send
10	5	1	1000.0	2024-12-05 15:15:12	send
19	6	1	2000.0	2024-12-05 15:54:48	send
25	7	1	345.0	2024-12-05 15:58:39	send
28	10	1	450.0	2024-12-05 16:00:07	send
30	8	1	500.0	2024-12-05 16:01:42	send
33	5	1	1.0	2024-12-05 16:02:32	send
40	9	1	500.0	2024-12-05 16:06:20	send
43	6	1	450.0	2024-12-05 16:09:08	send
47	5	1	100.0	2024-12-05 16:11:09	request

7) TRANSACTION



8)TRANSACTION STATISTICS



SOURCE CODE: https://drive.google.com/file/d/1I5g3lEXzEHZlgiKRTVl-Ws5vxBrj6bs8/view?usp=sharing