Decentralized Transaction Verification System

Mini Project Report -Database Lab (DSE 2241)

Department of Data Science & Computer Applications

B. Tech Data Science

4th Semester – Batch: A1 - Group: A15

Submitted By

Piyush Verma	230968114
Kewal Thacker	230968126
Anij Mehta	230968116
Purav Goyal	230968118
Kilaru Pardha Sai	230968134
Aditya Jain	230968132

Mentored By

Shreenidhi B
Assistant Professor
DSCA, MIT

Archana H
Assistant Professor-Senior
DSCA, MIT

MANIPAL INSTITUTE OF TECHNOLOGY

MANIPAL

(A constituent unit of MAHE, Manipal)

Date: 13th April, 2025

CERTIFICATE

This is to certify that the Piyush Verma (230968114), Kewal Thacker (230968126), Anij Mehta (230968116), Purav Goyal (230968118), Kilaru Pardha Sai (230968134), Aditya Jain (230968132), have successfully executed a mini project titled "Decentralized Transaction Verification System" rightly bringing fore the competencies and skill sets they have gained during the course- Database Lab (DSE 2241), thereby resulting in the culmination of this project.

Shreenidhi B
Assistant Professor
DSCA, MIT

Archana H
Assistant Professor-Senior
DSCA, MIT

ABSTRACT

The Decentralized Transaction Verification System presents a secure and tamper-resistant approach to managing financial transactions within a relational database environment. Traditional centralized database systems are susceptible to data tampering, fraud, and single points of failure. This project addresses these vulnerabilities by simulating blockchain-like features using MySQL, PL/SQL and SHA2 functions. Core functionalities include cryptographic hashing (SHA-256), sequential block linking, and a Proof-of-Work (PoW) simulation for transaction validation.

The system architecture consists of four key tables: Users, Wallets, Transactions, and Blocks, each designed to enforce data integrity and traceability. Transactions are cryptographically hashed and linked in a chain-like structure, ensuring immutability. Triggers and stored procedures validate each transaction, check for sufficient wallet balance, and prevent double-spending. Additionally, a basic frontend and hosted with streamlit, facilitates user interaction.

This SQL-based solution demonstrates how robust security, verifiability, and data consistency can be achieved without relying on external blockchain frameworks. Future enhancements include integrating Merkle trees for optimized querying and automated auditing features. The project showcases an innovative method for secure transaction management entirely within the bounds of relational database systems.

Contents

Chapter 1

Introduction

Database systems form the backbone of modern digital infrastructure, enabling the structured storage, retrieval, and management of vast volumes of data. Among their most critical applications is in the handling of financial transactions, where accuracy, consistency, and security are paramount. Traditional Relational Database Management Systems (RDBMS) such as Oracle SQL are designed to ensure data integrity through well-defined schemas, constraints, and transaction control mechanisms. However, these centralized systems are inherently vulnerable to risks such as unauthorized access, data tampering, and single points of failure.

As digital transactions continue to grow in scale and sensitivity, there is a pressing need for more robust solutions that go beyond conventional database safeguards. This has given rise to decentralized models like blockchain, which introduce features such as cryptographic hashing, data immutability, and consensus mechanisms to secure transactions. However, integrating such advanced features directly into a traditional SQL-based system remains an area of active exploration.

In this context, our project, Decentralized Transaction Verification System, bridges the gap between conventional relational databases and decentralized security principles. By leveraging MySQL, PL/SQL and SHA2 we design and implement a tamper-proof transaction management system that simulates core blockchain features—such as hashing, linked blocks, and a Proof-of-Work (PoW) mechanism—within a familiar SQL environment. This innovative approach ensures that transactions are validated, securely stored, and immutable, all while maintaining the structure and efficiency of a traditional database system.

Through this project, we aim to demonstrate how relational databases can be enhanced with cryptographic techniques to offer blockchain-level security, providing a scalable and practical solution for secure transaction verification in real-world applications.

Chapter 2

Synopsis

2.1 Proposed System

The proposed system simulates a simplified blockchain using MySQL to model essential blockchain functions such as transaction validation, mining, balance updates, and activity logging.

2.1.1. Transaction Validation

Transactions are validated through a `BEFORE INSERT` trigger on the `blockchain_transactions` table. The trigger ensures:

- The sender has sufficient balance (including transaction fees).
- Transactions are not self-directed.
- Amounts and fees are positive.

Invalid transactions are blocked and logged with appropriate messages for audit purposes.

2.1.2. Mining and Block Formation

A stored procedure handles the mining process by:

- Selecting valid transactions from the mempool.
- Grouping them into a new block.
- Crediting the miner with a fixed reward and the total transaction fees.
- Updating balances of all involved parties.

A system-generated reward transaction is added to the block for transparency.

2.1.3. Balance Calculation

Balances are computed dynamically by aggregating all incoming and outgoing transactions for each address, ensuring real-time consistency and eliminating the need for manual updates.

2.1.4. Logging and Automation

All valid and invalid actions are logged with timestamps and memos. Triggers ensure automation of:

- Transaction validation,
- Reward distribution,
- And balance enforcement.

2.1.5. Extensibility

Though implemented centrally, the system mimics key blockchain principles and can be extended to support cryptographic hashing, timestamping, and smart contracts in future iterations.

2.2 Objectives

2.2.1 Simulate Blockchain Functionality using SQL: To create a simplified yet functional model of a blockchain using MySQL database features like tables, triggers, and stored procedures.

2.2.2 Implement Block Mining Mechanism: To design a procedure that simulates the mining of blocks, including timestamping and chaining through hash references.

2.2.3 Enable Transaction Handling with Fees and Memos: To allow users to make transactions that include custom memos and fees, mimicking real-world use cases of blockchain.

2.2.4 Distribute Mining Rewards Automatically: To automate the allocation of rewards to miners after a block is successfully mined.

2.2.5 Ensure Data Integrity and Security: To enforce transaction validation and balance checks using triggers, ensuring the system mimics the immutability and reliability of an actual blockchain.

2.2.6 Track and Log All Activities: To maintain logs of all transactions and block creation events for audit and analysis purposes.

Chapter 3

Functional Requirements

3.1 Database Tablese

3.1.1 Users Table: Stores user account details including user ID and username.

3.1.2 Transactions Table: Stores transaction data such as sender, receiver, amount, fee, memo, status, and timestamp.

3.1.3 Blocks Table: Maintains block metadata such as block ID, miner ID, timestamp and total reward.

3.1.4 Wallets Table: Tracks wallets of every user to make transactions.

3.2. Stored Procedures

3.2.1 make_transaction(sender, receiver, amount, fee, memo): Adds a transaction request to the transactions table.

3.2.2 mine_block(miner_id): Groups unconfirmed transactions into a block, distributes rewards and fees to the miner, and confirms transactions.

3.3 Triggers

3.3.1 validate_transaction_before_insert: Validates transactions upon insertion to ensure:

- Sender exists
- Receiver exists
- Sufficient balance
- Positive amount and fee
- Prevents invalid or malicious transactions

3.4 Functions

3.4.1 generate_hash: Generates hash codes to encrypt all transactions using the SHA256 function.

3.5 Security & Logging

3.5.1 Status Tracking: Each transaction has a status field to denote success or failure.

3.5.2 Auditability: Failed transactions are retained with a reason, ensuring transparency and accountability.

Chapter 4

Detailed Design

4.1 ER Diagram

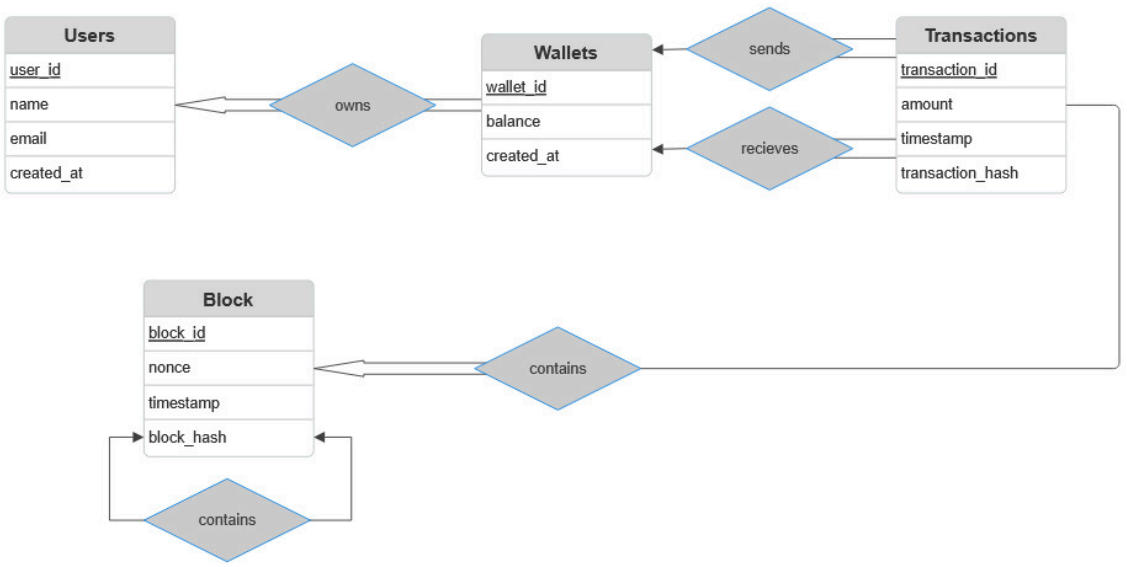

Figure 4.1 ER Diagram

4.2 Schema Diagram

Users (user_id, name, email, created_at)

Wallets (wallet_id, user_id, balance, created_at)

Transactions (transaction_id, block_id, amount, sender_wallet_id, reciever_wallet_id, timestamp, transaction_hash)

Blocks (block_id, block_hash, previous_block_id, timestamp, nonce)

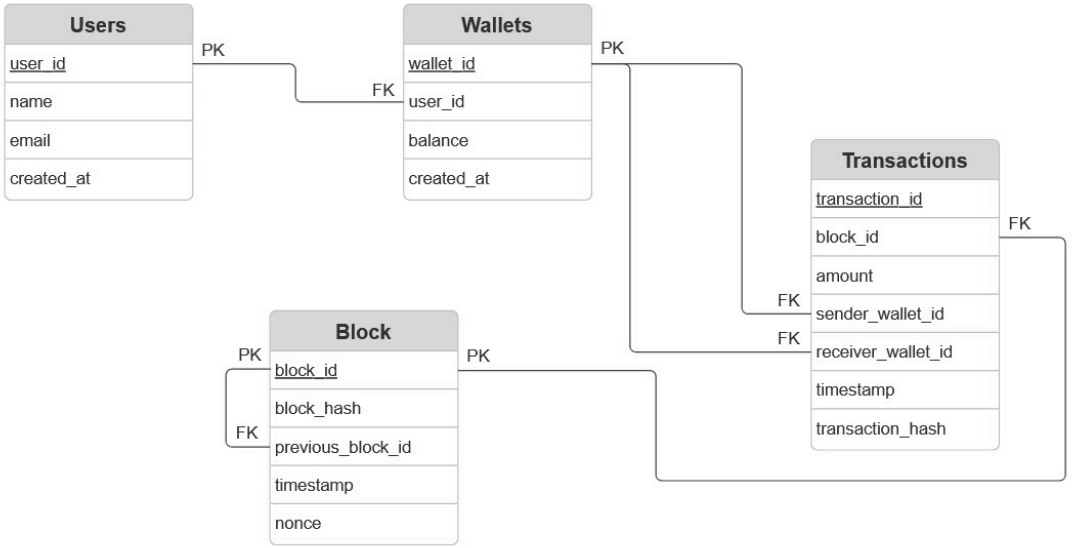

Figure 4.2 Schema Diagram

4.3 Data Dictionary

USERS

Column	Data type (size)	Constraint	Constraint Name
user_id	int	Primary Key, AUTO_INCREEMENT	
name	varchar(100)	NOT NULL	
email	varchar(255)	NOT NULL, UNIQUE	
password	varchar(255)	NOT NULL	
created_at	TIMESTAMP	DEFAULT, CURRENT_TIMESTAMP	

WALLETS

Column	Data type (size)	Constraint	Constraint Name
wallet_id	int	Primary Key, AUTO_INCREEMENT	
user_id	int	NOT NULL, REFERENCES USERS	
balance	decimal(20.8)	NOT NULL, DEFAULT 0	
created_at	TIMESTAMP	DEFAULT, CURRENT_TIMESTAMP	

BLOCKS

Column	Data type (size)	Constraint	Constraint Name
block_id	int	Primary Key, AUTO_INCREEMENT	
block_hash	int	NOT NULL,UNIQUE	
previous_block_id	int	DEFAULT NULL, REFERENCES BLOCKS	
timestamp	TIMESTAMP	DEFAULT, CURRENT_TIMESTAMP	
nonce	int	NOT NULL	

TRANSACTIONS

Column	Data type (size)	Constraint	Constraint Name
transaction_tid	int	Primary Key, AUTO_INCREEMENT	
block_id	int	NOT NULL, REFERENCES Blocks ON DELETE SET NULL	
amount	decimal(20,8)	CHECK (amount > 0), NOT NULL	
sender_wallet_id	int	NOT NULL, REFERENCES Wallets ON DELETE CASCADE	
receiver_wallet_id	int	NOT NULL, REFERENCES Wallets ON DELETE CASCADE	
timestamp	TIMESTAMP	DEFAULT, CURRENT_TIMESTAMP	
transaction_hash	varchar(64)	UNIQUE, NOT NULL	

4.4 Relational Model Implementation

CREATE TABLE Users (user_id INT AUTO_INCREMENT PRIMARY KEY, name VARCHAR(100) NOT NULL, email VARCHAR(255) UNIQUE NOT NULL, password VARCHAR(255) NOT NULL, created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP);

CREATE TABLE Wallets (wallet_id INT AUTO_INCREMENT PRIMARY KEY, user_id INT NOT NULL, balance DECIMAL(20,8) DEFAULT 0 NOT NULL, created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP, FOREIGN KEY (user_id) REFERENCES Users(user_id) ON DELETE CASCADE);

CREATE TABLE Blocks (block_id INT AUTO_INCREMENT PRIMARY KEY, block_hash VARCHAR(64) UNIQUE NOT NULL, previous_block_id INT DEFAULT NULL, timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP, nonce INT NOT NULL, FOREIGN KEY (previous_block_id) REFERENCES Blocks(block_id) ON DELETE CASCADE);

CREATE TABLE Transactions (transaction_id INT AUTO_INCREMENT PRIMARY KEY, block_id INT DEFAULT NULL, amount DECIMAL(20,8) CHECK (amount > 0) NOT NULL, sender_wallet_id INT NOT NULL, receiver_wallet_id INT NOT NULL, timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP, transaction_hash VARCHAR(64) UNIQUE NOT NULL, FOREIGN KEY (block_id) REFERENCES Blocks(block_id) ON DELETE SET NULL, FOREIGN KEY (sender_wallet_id) REFERENCES Wallets(wallet_id) ON DELETE CASCADE, FOREIGN KEY (receiver_wallet_id) REFERENCES Wallets(wallet_id) ON DELETE CASCADE);

Chapter 5

5. Implementation

5.1 Triggers

```
CREATE TRIGGER before_transaction_insert
BEFORE INSERT ON Transactions
FOR EACH ROW
BEGIN
   DECLARE sender_balance DECIMAL(20,8);

   -- Check sender wallet balance
    SELECT balance INTO sender_balance FROM Wallets WHERE wallet_id =
NEW.sender_wallet_id;

   IF sender_balance < NEW.amount THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Insufficient
balance';
   END IF;

   -- Deduct balance from sender
   UPDATE Wallets SET balance = balance - NEW.amount WHERE wallet_id =
NEW.sender_wallet_id;

   -- Add balance to receiver
   UPDATE Wallets SET balance = balance + NEW.amount WHERE wallet_id =
NEW.receiver_wallet_id;

   -- Generate transaction hash
SET NEW.transaction_hash = generate_hash(CONCAT(NEW.sender_wallet_id,
NEW.receiver_wallet_id, NEW.amount, NEW.timestamp));
END$$

CREATE TRIGGER validate_transaction BEFORE INSERT ON Transactions
FOR EACH ROW
BEGIN
   DECLARE sender_exists INT;
   DECLARE receiver_exists INT;

    SELECT COUNT(*) INTO sender_exists FROM Wallets WHERE wallet_id =
NEW.sender_wallet_id;
```

```
    SELECT COUNT(*) INTO receiver_exists FROM Wallets WHERE wallet_id =
NEW.receiver_wallet_id;

   IF sender_exists = 0 OR receiver_exists = 0 THEN
         SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Invalid sender or
receiver wallet';
   END IF;
END$$


CREATE TRIGGER after_transaction_update
AFTER UPDATE ON Transactions
FOR EACH ROW
BEGIN
   -- Update block size when a transaction is added to a block
   IF NEW.block_id IS NOT NULL AND OLD.block_id IS NULL THEN
       UPDATE Blocks
       SET size = size + 1
       WHERE block_id = NEW.block_id;

       -- Log the transaction confirmation
       INSERT INTO TransactionLogs (transaction_id, action, details)
        VALUES (NEW.transaction_id, 'confirmed', CONCAT('Added to block
#', NEW.block_id));

       -- Update transaction status
       UPDATE Transactions
       SET status = 'confirmed'
       WHERE transaction_id = NEW.transaction_id;

       -- Create alerts for transaction participants
       -- For sender
       INSERT INTO Alerts (user_id, title, message)
       SELECT u.user_id, 'Transaction Confirmed',
                 CONCAT('Your transaction of ', NEW.amount, ' has been
confirmed in block #', NEW.block_id)
       FROM Wallets w
       JOIN Users u ON w.user_id = u.user_id
       WHERE w.wallet_id = NEW.sender_wallet_id;

       -- For receiver
       INSERT INTO Alerts (user_id, title, message)
       SELECT u.user_id, 'Payment Received',
```

```
                CONCAT('You received ', NEW.amount, ' in a transaction
confirmed in block #', NEW.block_id)
       FROM Wallets w
       JOIN Users u ON w.user_id = u.user_id
       WHERE w.wallet_id = NEW.receiver_wallet_id;
   END IF;
END$$


CREATE TRIGGER before_transaction_insert
BEFORE INSERT ON Transactions
FOR EACH ROW
BEGIN
   DECLARE sender_balance DECIMAL(20,8);
   DECLARE total_amount DECIMAL(20,8);

   -- Skip validation for mining rewards (system transactions)
   IF NEW.sender_wallet_id IS NOT NULL THEN
       -- Calculate total amount including fee
       SET total_amount = NEW.amount + COALESCE(NEW.fee, 0);

       -- Check sender wallet balance
        SELECT balance INTO sender_balance FROM Wallets WHERE wallet_id
= NEW.sender_wallet_id;

       -- Validate sufficient balance
       IF sender_balance < total_amount THEN
              SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Insufficient
balance';
       END IF;

       -- Deduct balance from sender (including fee)
           UPDATE Wallets SET balance = balance - total_amount WHERE
wallet_id = NEW.sender_wallet_id;
   END IF;

    -- Add balance to receiver (just the transaction amount, not the
fee)
   IF NEW.receiver_wallet_id IS NOT NULL THEN
            UPDATE Wallets SET balance = balance + NEW.amount WHERE
wallet_id = NEW.receiver_wallet_id;
   END IF;
```

```
    -- Set initial status
    SET NEW.status = 'pending';

    -- Generate transaction hash if not provided
    IF NEW.transaction_hash IS NULL OR NEW.transaction_hash = '' THEN
        SET NEW.transaction_hash = generate_hash(CONCAT(
            COALESCE(NEW.sender_wallet_id, 'system'),
            NEW.receiver_wallet_id,
            NEW.amount,
            NEW.timestamp,
            COALESCE(NEW.memo, '')
        ));
    END IF;
END$$
```

....

5.2 Stored Procedures

```
CREATE PROCEDURE mine_block()
BEGIN
    DECLARE prev_block_hash VARCHAR(64);
    DECLARE new_block_hash VARCHAR(64);
    DECLARE mined_nonce INT DEFAULT 0;
    DECLARE curtime TIMESTAMP;
    DECLARE difficulty VARCHAR(4) DEFAULT '0000';

    -- Get the latest block hash
    SELECT block_hash INTO prev_block_hash FROM Blocks ORDER BY block_id
DESC LIMIT 1;

    -- If there is no previous block, set a default value
    IF prev_block_hash IS NULL THEN
SET                          prev_block_hash                          =
'0000000000000000000000000000000000000000000000000000000000000000';
    END IF;

    SET curtime = NOW();

    -- Simulated Proof-of-Work loop
    REPEAT
              SET  new_block_hash  =  generate_hash(CONCAT(prev_block_hash,
mined_nonce, curtime));

        -- Check if the block meets the difficulty requirement
```

```
        IF LEFT(new_block_hash, 4) = difficulty THEN
            LEAVE;
        ELSE
            SET mined_nonce = mined_nonce + 1;
        END IF;
    UNTIL FALSE END REPEAT;

    -- Insert new block
    INSERT INTO Blocks (block_hash, previous_block_id, timestamp, nonce)
    VALUES (new_block_hash, (SELECT MAX(block_id) FROM Blocks), curtime,
mined_nonce);

END$$

CREATE PROCEDURE make_transaction(
    IN sender_username VARCHAR(100),
    IN receiver_username VARCHAR(100),
    IN amount DECIMAL(20,8)
)
BEGIN
    DECLARE sender_wallet_id INT;
    DECLARE receiver_wallet_id INT;

    -- Get sender wallet ID
    SELECT wallet_id INTO sender_wallet_id FROM Wallets WHERE user_id =
(SELECT user_id FROM Users WHERE name = sender_username) LIMIT 1;

    -- Get receiver wallet ID
    SELECT wallet_id INTO receiver_wallet_id FROM Wallets WHERE user_id
= (SELECT user_id FROM Users WHERE name = receiver_username) LIMIT 1;

    -- Insert transaction
    INSERT INTO Transactions (block_id, amount, sender_wallet_id,
receiver_wallet_id, timestamp, transaction_hash)
    VALUES (NULL, amount, sender_wallet_id, receiver_wallet_id, NOW(),
SHA2(CONCAT(sender_wallet_id, receiver_wallet_id, amount, NOW()),
256));

    COMMIT;
END$$


CREATE PROCEDURE mine_block(IN force_difficulty VARCHAR(5))
```

```
BEGIN
   DECLARE prev_block_hash VARCHAR(64);
   DECLARE new_block_hash VARCHAR(64);
   DECLARE mined_nonce INT DEFAULT 0;
   DECLARE curtime TIMESTAMP;
   DECLARE difficulty VARCHAR(5);
   DECLARE block_count INT;
   DECLARE avg_mining_time INT;

   -- Get current block count
   SELECT COUNT(*) INTO block_count FROM Blocks;

   -- Calculate appropriate difficulty based on blockchain size
   -- This provides automatic difficulty adjustment
   IF force_difficulty IS NOT NULL THEN
       SET difficulty = force_difficulty;
   ELSE
       -- Determine difficulty dynamically
       IF block_count < 100 THEN
           SET difficulty = '0000';  -- Easy
       ELSEIF block_count < 1000 THEN
           SET difficulty = '00000'; -- Medium
       ELSE
           SET difficulty = '000000'; -- Hard
       END IF;
   END IF;

   -- Get the latest block hash
   SELECT block_hash INTO prev_block_hash FROM Blocks ORDER BY block_id
DESC LIMIT 1;

   -- If there is no previous block, set a default value
   IF prev_block_hash IS NULL THEN
                                          SET      prev_block_hash      =
'0000000000000000000000000000000000000000000000000000000000000000';
   END IF;

   SET curtime = NOW();

   -- Simulated Proof-of-Work loop (improved for performance)
   REPEAT
            SET  new_block_hash  =  generate_hash(CONCAT(prev_block_hash,
mined_nonce, curtime));
```

```
        -- Check if the block meets the difficulty requirement
        IF LEFT(new_block_hash, LENGTH(difficulty)) = difficulty THEN
            LEAVE;
        ELSE
            SET mined_nonce = mined_nonce + 1;
        END IF;

        -- Safety exit for infinite loop prevention
        IF mined_nonce > 10000 THEN
                    -- If we exceed 10000 attempts, reduce difficulty
temporarily
            SET difficulty = LEFT(difficulty, LENGTH(difficulty) - 1);
            SET mined_nonce = 0;
        END IF;
    UNTIL FALSE END REPEAT;

    -- Insert new block
    INSERT INTO Blocks (block_hash, previous_block_id, timestamp, nonce,
difficulty)
    VALUES (new_block_hash, (SELECT MAX(block_id) FROM Blocks), curtime,
mined_nonce, difficulty);

    -- Process pending transactions (up to 10 per block)
    UPDATE Transactions
    SET block_id = LAST_INSERT_ID()
    WHERE block_id IS NULL
    ORDER BY timestamp
    LIMIT 10;

END$$


CREATE PROCEDURE make_transaction(
    IN sender_username VARCHAR(100),
    IN receiver_username VARCHAR(100),
    IN amount DECIMAL(20,8),
    IN memo VARCHAR(255)
)
BEGIN
    DECLARE sender_wallet_id INT;
    DECLARE receiver_wallet_id INT;
    DECLARE transaction_fee DECIMAL(20,8);
```

```
   DECLARE sender_user_id INT;
   DECLARE receiver_user_id INT;
   DECLARE tx_hash VARCHAR(64);

    -- Set transaction fee (0.1% of transaction amount with minimum of
0.01)
   SET transaction_fee = GREATEST(amount * 0.001, 0.01);

   -- Start transaction
   START TRANSACTION;

   -- Get sender user ID
   SELECT user_id INTO sender_user_id
   FROM Users
   WHERE name = sender_username
   LIMIT 1;

   -- Get receiver user ID
   SELECT user_id INTO receiver_user_id
   FROM Users
   WHERE name = receiver_username
   LIMIT 1;

   -- Validate users exist
   IF sender_user_id IS NULL OR receiver_user_id IS NULL THEN
         SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Invalid sender or
receiver username';
       ROLLBACK;
   END IF;

   -- Get sender wallet ID
   SELECT wallet_id INTO sender_wallet_id
   FROM Wallets
   WHERE user_id = sender_user_id
   ORDER BY balance DESC
   LIMIT 1;

   -- Get receiver wallet ID
   SELECT wallet_id INTO receiver_wallet_id
   FROM Wallets
   WHERE user_id = receiver_user_id
   LIMIT 1;
```

```
    -- Validate wallets exist
    IF sender_wallet_id IS NULL OR receiver_wallet_id IS NULL THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Wallet not found';
        ROLLBACK;
    END IF;

    -- Generate transaction hash
      SET tx_hash = SHA2(CONCAT(sender_wallet_id, receiver_wallet_id,
amount, NOW(), RAND()), 256);

    -- Insert transaction
      INSERT INTO Transactions (block_id, amount, sender_wallet_id,
receiver_wallet_id, timestamp, transaction_hash, fee, memo)
    VALUES (NULL, amount, sender_wallet_id, receiver_wallet_id, NOW(),
tx_hash, transaction_fee, memo);

    -- Commit transaction
    COMMIT;
END$$



CREATE PROCEDURE distribute_mining_reward(IN miner_user_id INT)
BEGIN
    DECLARE miner_wallet_id INT;
    DECLARE block_count INT;
    DECLARE mining_reward DECIMAL(20,8);

    -- Get miner's wallet
    SELECT wallet_id INTO miner_wallet_id
    FROM Wallets
    WHERE user_id = miner_user_id
    LIMIT 1;

    -- Calculate current mining reward (starts high and decreases over
time)
    SELECT COUNT(*) INTO block_count FROM Blocks;
    SET mining_reward = 50 / POWER(2, FLOOR(block_count / 100000));

    -- Add reward to miner's wallet
    UPDATE Wallets
    SET balance = balance + mining_reward
    WHERE wallet_id = miner_wallet_id;
```

```
   -- Record reward as a system transaction
      INSERT INTO Transactions (block_id, amount, sender_wallet_id,
receiver_wallet_id, timestamp, transaction_hash, fee, memo)
   VALUES (
       (SELECT MAX(block_id) FROM Blocks),
       mining_reward,
       NULL, -- System transaction (no sender)
       miner_wallet_id,
       NOW(),
       SHA2(CONCAT('mining_reward', miner_wallet_id, NOW()), 256),
       0,
       'Mining Reward'
   );
END$$
```

…

5.3 Stored Functions

```
CREATE FUNCTION generate_hash(input_string VARCHAR(255)) RETURNS
VARCHAR(64) DETERMINISTIC
RETURN SHA2(input_string, 256);
```

…..

5.4 Frontend / Hosting

```
import streamlit as st
import database
import pandas as pd
import plotly.express as px
import plotly.graph_objects as go
import hashlib
import time
from datetime import datetime, timedelta
import base64
from PIL import Image
import io
import re

# Page config with favicon and expanded layout
st.set_page_config(
   page_title="Decentralized Transaction Verification System",
   page_icon="💎",
```

```
    layout="wide",
    initial_sidebar_state="expanded"
)

# Apply custom CSS
st.markdown("""
<style>
    .main {
        background-color: black;
    }
    .block-container {
        padding-top: 1rem;
        padding-bottom: 1rem;
    }
    .stButton>button {
        background-color: white;
        color: black;
        border-radius: 8px;
        border: none;
        padding: 0.5rem 1rem;
        font-weight: bold;
    }
    .stButton>button:hover {
        background-color: #00FFFF;
        color : black;
    }
    .card {
        background-color: #00FFFF;
        color : black;
        border-radius: 10px;
        padding: 1.5rem;
        box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
        margin-bottom: 1rem;
    }
    .metric-value {
        font-size: 2rem;
        font-weight: bold;
        color: #333;
    }
    .metric-label {
        color: #666;
        font-size: 0.9rem;
    }
```

```
    .hash-text {
        font-family: monospace;
        overflow-wrap: break-word;
    }
    .footer {
        text-align: center;
        padding: 1rem;
        font-size: 0.8rem;
        color: black;
    }
</style>
""", unsafe_allow_html=True)

# Initialize database connection
conn = database.get_db_connection()
cursor = conn.cursor(dictionary=True)

# Initialize session state variables
if "user_id" not in st.session_state:
    st.session_state.user_id = None
if "theme" not in st.session_state:
    st.session_state.theme = "light"
if "notification" not in st.session_state:
    st.session_state.notification = None
if "last_activity" not in st.session_state:
    st.session_state.last_activity = datetime.now()

# Check session timeout (15 minutes)
if       st.session_state.user_id       and       (datetime.now()       -
st.session_state.last_activity) > timedelta(minutes=15):
    st.session_state.user_id = None
     st.session_state.notification = "Session timed out. Please log in
again."

# Update last activity timestamp
st.session_state.last_activity = datetime.now()

# Function to hash passwords
def hash_password(password):
    return hashlib.sha256(password.encode()).hexdigest()

# Function to validate email
def is_valid_email(email):
```

```
    pattern = r"^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$"
    return re.match(pattern, email) is not None

# Function to export transactions to CSV
def get_download_link(df, filename, text):
    csv = df.to_csv(index=False)
    b64 = base64.b64encode(csv.encode()).decode()
                href      =      f'<a      href="data:file/csv;base64,{b64}"
download="{filename}">{text}</a>'
    return href

# Function to get blockchain statistics
def get_blockchain_stats():
    cursor.execute("SELECT COUNT(*) as block_count FROM Blocks")
    block_count = cursor.fetchone()['block_count']

    cursor.execute("SELECT COUNT(*) as tx_count FROM Transactions")
    tx_count = cursor.fetchone()['tx_count']

    cursor.execute("SELECT COUNT(*) as user_count FROM Users")
    user_count = cursor.fetchone()['user_count']

    cursor.execute("SELECT SUM(amount) as volume FROM Transactions")
    result = cursor.fetchone()
    volume = result['volume'] if result['volume'] else 0

    return {
        "blocks": block_count,
        "transactions": tx_count,
        "users": user_count,
        "volume": volume
    }

# Sidebar for navigation
with st.sidebar:
    st.title("Decentralized Transaction Verification System")

    st.markdown("---")

    # Menu options based on login state
    if st.session_state.user_id is None:
        menu = ["Home", "Login", "Register"]
    else:
```

```
        # Get username for display
            cursor.execute("SELECT name FROM Users WHERE user_id = %s",
(st.session_state.user_id,))
        user = cursor.fetchone()
        st.markdown(f"### Welcome, {user['name']}!")

        menu = [
            "Dashboard",
            "My Transactions",
            "Make Transaction",
            "My Wallets",
            "Block Explorer",
            "Profile Settings",
            "Logout"
        ]

    choice = st.radio("Navigation", menu)

    st.markdown("---")

    # Display some blockchain stats in sidebar
    if st.session_state.user_id is not None:
        stats = get_blockchain_stats()
        st.markdown("### Network Statistics")
        st.markdown(f"**Blocks:** {stats['blocks']}")
        st.markdown(f"**Transactions:** {stats['transactions']}")
        st.markdown(f"**Total Users:** {stats['users']}")
        st.markdown(f"**Transaction Volume:** ${stats['volume']:,.2f}")

    st.markdown("---")
        st.markdown("<div class='footer'>© Decentralized Transaction
Verification System</div>", unsafe_allow_html=True)

# Display notification if exists
if st.session_state.notification:
    st.warning(st.session_state.notification)
    st.session_state.notification = None

# HOME PAGE
if choice == "Home":
    st.title("Welcome to the Decentralized Transaction Verification
System")
```

```
    col1, col2 = st.columns([2, 1])

    with col1:
        st.markdown("""
        <div class="card">
            <h2>Secure Blockchain Transactions</h2>
                <p>Our platform offers a secure and transparent way to
manage your digital transactions using blockchain technology.</p>
            <p>Key features:</p>
            <ul>
                <li>Real-time transaction verification</li>
                <li>Secure wallet management</li>
                <li>Detailed blockchain explorer</li>
                <li>Analytics and reporting</li>
            </ul>
        </div>
        """, unsafe_allow_html=True)

        st.markdown("""
        <div class="card">
            <h3>Recent Transactions</h3>
        </div>
        """, unsafe_allow_html=True)

        # Display recent transactions
        cursor.execute("""
            SELECT t.transaction_hash, t.amount,
                   s.name as sender, r.name as receiver,
                   t.timestamp
            FROM Transactions t
            JOIN Wallets ws ON t.sender_wallet_id = ws.wallet_id
            JOIN Users s ON ws.user_id = s.user_id
            JOIN Wallets wr ON t.receiver_wallet_id = wr.wallet_id
            JOIN Users r ON wr.user_id = r.user_id
            ORDER BY t.timestamp DESC LIMIT 5
        """)
        recent_txs = cursor.fetchall()

        if recent_txs:
            tx_df = pd.DataFrame(recent_txs)
                                            tx_df['transaction_hash']    =
tx_df['transaction_hash'].apply(lambda x: x[:8] + '...' + x[-8:])
            st.dataframe(tx_df, hide_index=True)
```

```
        else:
            st.info("No transactions available")

    with col2:
        st.markdown("""
        <div class="card">
            <h3>Get Started</h3>
                <p>Create an account or login to start exploring the
blockchain.</p>
        </div>
        """, unsafe_allow_html=True)

        st.markdown("""
        <div class="card">
            <h3>Block Height</h3>
        </div>
        """, unsafe_allow_html=True)

        # Display current block height
        cursor.execute("SELECT MAX(block_id) as height FROM Blocks")
        height = cursor.fetchone()['height'] or 0
               st.markdown(f"<div  class='metric-value'>{height}</div>",
unsafe_allow_html=True)

        # Display latest block time
          cursor.execute("SELECT timestamp FROM Blocks ORDER BY block_id
DESC LIMIT 1")
        latest = cursor.fetchone()
        if latest:
            latest_time = latest['timestamp']
                  st.markdown(f"<div  class='metric-label'>Latest block:
{latest_time}</div>", unsafe_allow_html=True)


# LOGIN PAGE
elif choice == "Login":
    st.title("Login to Your Account")

    with st.form("login_form"):
        email = st.text_input("Email")
        password = st.text_input("Password", type="password")
        submit = st.form_submit_button("Login")
```

```
        if submit:
            if not email or not password:
                st.error("Please fill in all fields")
            else:
                hashed_pw = hash_password(password)
                   cursor.execute("SELECT user_id, name FROM Users WHERE
email=%s AND password=%s", (email, hashed_pw))
                user = cursor.fetchone()

                if user:
                    st.session_state.user_id = user['user_id']
                    st.success(f"Welcome back, {user['name']}!")
                    time.sleep(1)
                    st.rerun()
                else:
                    st.error("Invalid email or password")

    st.markdown("Don't have an account? Navigate to Register from the
sidebar.")

# REGISTER PAGE
elif choice == "Register":
   st.title("Create a New Account")

   with st.form("register_form"):
        col1, col2 = st.columns(2)

        with col1:
            name = st.text_input("Username")
            email = st.text_input("Email")

        with col2:
            password = st.text_input("Password", type="password")
                  confirm_password = st.text_input("Confirm Password",
type="password")

        submit = st.form_submit_button("Register")

        if submit:
            # Form validation
                   if not name or not email or not password or not
confirm_password:
                st.error("Please fill in all fields")
```

```
            elif not is_valid_email(email):
                st.error("Please enter a valid email address")
            elif password != confirm_password:
                st.error("Passwords do not match")
            elif len(password) < 8:
                st.error("Password must be at least 8 characters long")
            else:
                try:
                    hashed_pw = hash_password(password)
                        cursor.execute("INSERT INTO Users (name, email,
password) VALUES (%s, %s, %s)",
                                       (name, email, hashed_pw))
                    conn.commit()

                    # Get the new user's ID
                        cursor.execute("SELECT user_id FROM Users WHERE
email = %s", (email,))
                    user_id = cursor.fetchone()['user_id']

                    # Create a wallet for the new user
                           cursor.execute("INSERT INTO Wallets (user_id,
balance) VALUES (%s, %s)",
                                          (user_id, 100.0))  # Starting balance
of 100
                    conn.commit()

                       st.success("Account created successfully! You can
now log in.")
                    time.sleep(1)
                    st.rerun()
                except Exception as e:
                    st.error(f"Registration failed: {e}")

# DASHBOARD PAGE
elif choice == "Dashboard" and st.session_state.user_id:
   st.title("Your Dashboard")

   # Get user info
      cursor.execute("SELECT  name  FROM  Users  WHERE  user_id  =  %s",
(st.session_state.user_id,))
   user = cursor.fetchone()
```

```
    # Get user's wallet balance - this is correct, it's just summing up
the actual balances
    cursor.execute("SELECT SUM(balance) as total_balance FROM Wallets
WHERE user_id = %s",
                   (st.session_state.user_id,))
    balance = cursor.fetchone()['total_balance'] or 0

    # Get user's transaction count - but we need to make sure we're not
double counting
    # Modified query to prevent double counting when both sender and
receiver are the same user
    cursor.execute("""
        SELECT COUNT(DISTINCT t.transaction_id) as tx_count
        FROM Transactions t
             JOIN Wallets w ON t.sender_wallet_id = w.wallet_id OR
t.receiver_wallet_id = w.wallet_id
        WHERE w.user_id = %s
    """, (st.session_state.user_id,))
    tx_count = cursor.fetchone()['tx_count']

    # Display key metrics
    col1, col2, col3, col4 = st.columns(4)

    with col1:
        st.markdown("""
        <div class="card">
            <div class="metric-label">Total Balance</div>
            <div class="metric-value">$%.2f</div>
        </div>
        """ % balance, unsafe_allow_html=True)

    with col2:
        st.markdown("""
        <div class="card">
            <div class="metric-label">Transactions</div>
            <div class="metric-value">%d</div>
        </div>
        """ % tx_count, unsafe_allow_html=True)

    with col3:
        # Fix the total sent calculation to handle internal transfers
        cursor.execute("""
            SELECT SUM(amount) as sent
```

```
            FROM Transactions t
            JOIN Wallets ws ON t.sender_wallet_id = ws.wallet_id
            LEFT JOIN Wallets wr ON t.receiver_wallet_id = wr.wallet_id
             WHERE ws.user_id = %s AND (wr.user_id IS NULL OR wr.user_id
!= %s)
        """, (st.session_state.user_id, st.session_state.user_id))
        sent_external = cursor.fetchone()['sent'] or 0

          # Add internal transfers as a separate calculation to avoid
double counting
        cursor.execute("""
            SELECT SUM(amount) as sent_internal
            FROM Transactions t
            JOIN Wallets ws ON t.sender_wallet_id = ws.wallet_id
            JOIN Wallets wr ON t.receiver_wallet_id = wr.wallet_id
            WHERE ws.user_id = %s AND wr.user_id = %s
        """, (st.session_state.user_id, st.session_state.user_id))
        sent_internal = cursor.fetchone()['sent_internal'] or 0

        # Total sent is external transfers plus internal transfers
        sent = sent_external + sent_internal

        st.markdown("""
        <div class="card">
            <div class="metric-label">Total Sent</div>
            <div class="metric-value">$%.2f</div>
        </div>
        """ % sent, unsafe_allow_html=True)

    with col4:
        # Fix the total received calculation similarly
        cursor.execute("""
            SELECT SUM(amount) as received
            FROM Transactions t
            JOIN Wallets wr ON t.receiver_wallet_id = wr.wallet_id
            LEFT JOIN Wallets ws ON t.sender_wallet_id = ws.wallet_id
             WHERE wr.user_id = %s AND (ws.user_id IS NULL OR ws.user_id
!= %s)
        """, (st.session_state.user_id, st.session_state.user_id))
        received_external = cursor.fetchone()['received'] or 0

        # Use the same internal transfer amount calculated above
        # Total received is external transfers plus internal transfers
```

```
            received = received_external + sent_internal    # We use
sent_internal since it's the same amount

        st.markdown("""
        <div class="card">
            <div class="metric-label">Total Received</div>
            <div class="metric-value">$%.2f</div>
        </div>
        """ % received, unsafe_allow_html=True)

    cursor.execute("""
        SELECT DATE(t.timestamp) as date,
               SUM(
                   CASE
                               -- When user is sender but not receiver
(outgoing)
                       WHEN sw.user_id = %s AND rw.user_id != %s THEN
-amount
                               -- When user is receiver but not sender
(incoming)
                       WHEN rw.user_id = %s AND sw.user_id != %s THEN
amount
                               -- When user is both sender and receiver
(internal), don't count
                       ELSE 0
                   END
               ) as net_flow
        FROM Transactions t
        JOIN Wallets sw ON t.sender_wallet_id = sw.wallet_id
        JOIN Wallets rw ON t.receiver_wallet_id = rw.wallet_id
        WHERE sw.user_id = %s OR rw.user_id = %s
        GROUP BY DATE(t.timestamp)
        ORDER BY DATE(t.timestamp)
           """,    (st.session_state.user_id,    st.session_state.user_id,
st.session_state.user_id,                       st.session_state.user_id,
st.session_state.user_id, st.session_state.user_id))

    history_data = cursor.fetchall()

    if history_data:
        history_df = pd.DataFrame(history_data)

        # Create a transaction flow chart
```

```
        fig = px.line(history_df, x='date', y='net_flow',
                      title='Daily Transaction Flow',
                      labels={'date': 'Date', 'net_flow': 'Net Flow'})

        # Add a horizontal line at y=0
                fig.add_shape(type='line',  x0=history_df['date'].min(),
x1=history_df['date'].max(),
                              y0=0, y1=0, line=dict(color='gray', width=1,
dash='dash'))

        # Color code positive vs negative flows
        fig.update_traces(line=dict(color='green'))

        st.plotly_chart(fig, use_container_width=True)
    else:
        st.info("No transaction history available")

    # Recent activity
    col1, col2 = st.columns(2)

    with col1:
        st.markdown("""
        <div class="card">
            <h3>Recent Transactions</h3>
        </div>
        """, unsafe_allow_html=True)

        cursor.execute("""
            SELECT t.transaction_hash, t.amount,
                   s.name as sender, r.name as receiver,
                   t.timestamp,
                    CASE WHEN ws.user_id = %s THEN 'Sent' ELSE 'Received'
END as type
            FROM Transactions t
            JOIN Wallets ws ON t.sender_wallet_id = ws.wallet_id
            JOIN Users s ON ws.user_id = s.user_id
            JOIN Wallets wr ON t.receiver_wallet_id = wr.wallet_id
            JOIN Users r ON wr.user_id = r.user_id
            WHERE ws.user_id = %s OR wr.user_id = %s
            ORDER BY t.timestamp DESC LIMIT 5
              """,  (st.session_state.user_id,  st.session_state.user_id,
st.session_state.user_id))
```

```
        user_txs = cursor.fetchall()

        if user_txs:
            user_tx_df = pd.DataFrame(user_txs)
                                        user_tx_df['transaction_hash']    =
user_tx_df['transaction_hash'].apply(lambda x: x[:8] + '...' + x[-8:])
            st.dataframe(user_tx_df, hide_index=True)
        else:
            st.info("No transactions yet")

    with col2:
        st.markdown("""
        <div class="card">
            <h3>Your Wallets</h3>
        </div>
        """, unsafe_allow_html=True)

        cursor.execute("""
            SELECT wallet_id, balance, created_at
            FROM Wallets
            WHERE user_id = %s
        """, (st.session_state.user_id,))

        wallets = cursor.fetchall()

        if wallets:
            wallet_df = pd.DataFrame(wallets)
            st.dataframe(wallet_df, hide_index=True)
        else:
            st.info("No wallets found")

# MY TRANSACTIONS PAGE
elif choice == "My Transactions" and st.session_state.user_id:
    st.title("My Transactions")

    # Transaction filters
    col1, col2, col3 = st.columns(3)

    with col1:
            tx_type  =  st.selectbox("Transaction  Type",  ["All",  "Sent",
"Received"])

    with col2:
```

```
            sort_by = st.selectbox("Sort By", ["Newest First", "Oldest
First", "Amount (High to Low)", "Amount (Low to High)"])

    with col3:
        date_range = st.date_input("Date Range", value=[datetime.now() -
timedelta(days=30), datetime.now()])

    # Apply filters to query
    query = """
        SELECT t.transaction_id, t.transaction_hash, t.amount,
               s.name as sender, r.name as receiver,
               t.timestamp,
                CASE WHEN ws.user_id = %s THEN 'Sent' ELSE 'Received' END
as type
        FROM Transactions t
        JOIN Wallets ws ON t.sender_wallet_id = ws.wallet_id
        JOIN Users s ON ws.user_id = s.user_id
        JOIN Wallets wr ON t.receiver_wallet_id = wr.wallet_id
        JOIN Users r ON wr.user_id = r.user_id
        WHERE (ws.user_id = %s OR wr.user_id = %s)
    """

        params = [st.session_state.user_id, st.session_state.user_id,
st.session_state.user_id]

    # Add transaction type filter
    if tx_type == "Sent":
        query += " AND ws.user_id = %s"
        params.append(st.session_state.user_id)
    elif tx_type == "Received":
        query += " AND wr.user_id = %s"
        params.append(st.session_state.user_id)

    # Add date range filter
    if len(date_range) == 2:
        start_date, end_date = date_range
        end_date = end_date + timedelta(days=1)  # Include the end date
        query += " AND t.timestamp BETWEEN %s AND %s"
        params.extend([start_date, end_date])

    # Add sorting
    if sort_by == "Newest First":
        query += " ORDER BY t.timestamp DESC"
```

```
    elif sort_by == "Oldest First":
        query += " ORDER BY t.timestamp ASC"
    elif sort_by == "Amount (High to Low)":
        query += " ORDER BY t.amount DESC"
    elif sort_by == "Amount (Low to High)":
        query += " ORDER BY t.amount ASC"

    cursor.execute(query, tuple(params))
    transactions = cursor.fetchall()

    if transactions:
        tx_df = pd.DataFrame(transactions)

        # Display the transactions
        st.dataframe(tx_df, hide_index=True)

        # Add export button
         st.markdown(get_download_link(tx_df, "my_transactions.csv", "📥
Download Transactions as CSV"), unsafe_allow_html=True)

        # Create a pie chart of sent vs received
            sent_amount = sum([tx['amount'] for tx in transactions if
tx['type'] == 'Sent'])
          received_amount = sum([tx['amount'] for tx in transactions if
tx['type'] == 'Received'])

        if sent_amount > 0 or received_amount > 0:
            fig = go.Figure(data=[go.Pie(
                labels=['Sent', 'Received'],
                values=[sent_amount, received_amount],
                hole=.3,
                marker_colors=['#FF6B6B', '#4CAF50']
            )])

            fig.update_layout(title_text="Transaction Balance")
            st.plotly_chart(fig, use_container_width=True)
    else:
        st.info("No transactions found matching your criteria")

# MAKE TRANSACTION PAGE
elif choice == "Make Transaction" and st.session_state.user_id:
    st.title("Send Transaction")
```

```
    # Initialize session states for wallet selection if not exist
    if "receiver_username" not in st.session_state:
        st.session_state.receiver_username = ""
    if "send_to_self" not in st.session_state:
        st.session_state.send_to_self = False
    if "transaction_submitted" not in st.session_state:
        st.session_state.transaction_submitted = False

    # Get sender's wallets
    cursor.execute("""
        SELECT w.wallet_id, w.balance
        FROM Wallets w
        WHERE w.user_id = %s
    """, (st.session_state.user_id,))

    sender_wallets = cursor.fetchall()

    if not sender_wallets:
        st.error("You don't have any wallets. Please contact support.")
    else:
        # Get user's name
           cursor.execute("SELECT name FROM Users WHERE user_id = %s",
(st.session_state.user_id,))
        user = cursor.fetchone()
        sender_name = user['name']

         # Create two sections - first for recipient selection, then for
transaction details
        st.markdown("### Step 1: Select Source and Destination")

        col1, col2 = st.columns(2)

        with col1:
            st.markdown(f"**Sender:** {sender_name}")

            # Select wallet if multiple
            if len(sender_wallets) > 1:
                  wallet_options = {f"Wallet #{w['wallet_id']} (Balance:
${w['balance']:.2f})": w['wallet_id'] for w in sender_wallets}
                      selected_wallet_key = st.selectbox("Select Source
Wallet", list(wallet_options.keys()))
                wallet_id = wallet_options[selected_wallet_key]
```

```
                        current_balance = next(w['balance'] for w in
sender_wallets if w['wallet_id'] == wallet_id)
            else:
                wallet_id = sender_wallets[0]['wallet_id']
                current_balance = sender_wallets[0]['balance']
                    st.markdown(f"**Source Wallet:** Wallet #{wallet_id}
(Balance: ${current_balance:.2f})")

        with col2:
            # Add option to send to own wallet
             st.session_state.send_to_self = st.checkbox("Send to my own
wallet", value=st.session_state.send_to_self)

            if st.session_state.send_to_self:
                 # Get user's other wallets (exclude the selected source
wallet)
                        other_wallets = [w for w in sender_wallets if
w['wallet_id'] != wallet_id]

                if not other_wallets:
                      st.error("You don't have any other wallets to send
to. Please create another wallet first.")
                    receiver_wallet_id = None
                else:
                                    receiver_wallet_options = {f"Wallet
#{w['wallet_id']} (Balance: ${w['balance']:.2f})": w['wallet_id'] for w
in other_wallets}
                    selected_receiver_wallet_key = st.selectbox("Select
Destination Wallet", list(receiver_wallet_options.keys()))
                                                        receiver_wallet_id =
receiver_wallet_options[selected_receiver_wallet_key]
                    receiver_username = sender_name  # Same as sender
            else:
                # Receiver details
                 receiver_username = st.text_input("Recipient Username",
value=st.session_state.receiver_username)
                st.session_state.receiver_username = receiver_username

                receiver_wallet_id = None  # Initialize as None
                if receiver_username:
                    # Verify receiver exists
                     cursor.execute("SELECT user_id FROM Users WHERE name
= %s", (receiver_username,))
```

```
                    receiver = cursor.fetchone()

                    if receiver:
                                                if  receiver['user_id']  ==
st.session_state.user_id:
                                    st.warning("This is your own username.
Consider using the 'Send to my own wallet' option instead.")

                        # Get receiver's wallets
                        cursor.execute("""
                            SELECT wallet_id, balance
                            FROM Wallets
                            WHERE user_id = %s
                        """, (receiver['user_id'],))

                        receiver_wallets = cursor.fetchall()

                        if receiver_wallets:
                            # Let user select which wallet to send to
                                        receiver_wallet_options = {f"Wallet
#{w['wallet_id']} (Balance: ${w['balance']:.2f})": w['wallet_id'] for w
in receiver_wallets}
                                             selected_receiver_wallet_key  =
st.selectbox("Select                 Recipient's                 Wallet",
list(receiver_wallet_options.keys()))
                                                        receiver_wallet_id  =
receiver_wallet_options[selected_receiver_wallet_key]
                        else:
                                    st.error("Recipient does not have any
wallets")
                    else:
                        st.error("Recipient not found")

        st.markdown("---")
        st.markdown("### Step 2: Transaction Details")

        # Now create the form for the actual transaction
        with st.form("transaction_form"):
            # Start with 0 instead of 0.01
                     amount  =  st.number_input("Amount",  min_value=0.00,
max_value=float(current_balance), value=0.00, format="%.2f")
            memo = st.text_input("Memo (Optional)", max_chars=100)
```

```
            # Hidden fields to store the selected wallet IDs
                st.markdown(f"From Wallet #{wallet_id} to {'your own ' if
st.session_state.send_to_self  else  ''}Wallet  #{receiver_wallet_id  if
receiver_wallet_id else 'unknown'}")

            # Add a confirmation checkbox
                    confirm_transaction  =  st.checkbox("I  confirm  this
transaction is correct")

            submit = st.form_submit_button("Send Transaction")

            if submit:
                if not receiver_wallet_id:
                    st.error("Please select a valid recipient wallet")
                elif amount <= 0:
                    st.error("Amount must be greater than 0")
                elif amount > current_balance:
                    st.error("Insufficient balance")
                elif not confirm_transaction:
                        st.error("Please confirm your transaction before
sending")
                else:
                    st.session_state.transaction_submitted = True
                    try:
                                # Use individual statements with proper
transaction management
                        try:
                            # Start transaction
                            conn.autocommit = False

                            # Generate transaction hash
                                                                tx_hash  =
hashlib.sha256(f"{wallet_id}{receiver_wallet_id}{amount}{time.time()}".
encode()).hexdigest()

                            # Create transaction record
                            cursor.execute("""
                                INSERT INTO Transactions
                                    (transaction_hash, sender_wallet_id,
receiver_wallet_id, amount, memo)
                                VALUES (%s, %s, %s, %s, %s)
                                            """, (tx_hash, wallet_id,
receiver_wallet_id, amount, memo))
```

```
                            # Commit transaction
                            conn.commit()

                                # In Make Transaction section, modify the
confirmation message:
                                        recipient_display = sender_name if
st.session_state.send_to_self else receiver_username
                            if st.session_state.send_to_self:
                                    st.success(f"Successfully transferred
${amount:.2f} between your wallets (Wallet #{wallet_id} → Wallet
#{receiver_wallet_id})")
                            else:
                                                st.success(f"Successfully sent
${amount:.2f} to {recipient_display}'s Wallet #{receiver_wallet_id}")

                            # Show animation of successful transaction
                            st.balloons()

                        except Exception as e:
                            # Rollback on error
                            conn.rollback()
                            raise e
                        finally:
                            # Reset autocommit
                            conn.autocommit = True

                    except Exception as e:
                        st.error(f"Transaction failed: {e}")

        # Add a separate transaction confirmation area
        if st.session_state.transaction_submitted:
                st.session_state.transaction_submitted = False   # Reset
after displaying

# MY WALLETS PAGE
elif choice == "My Wallets" and st.session_state.user_id:
    st.title("My Wallets")

    # Get user's wallets
    cursor.execute("""
        SELECT wallet_id, balance, created_at
        FROM Wallets
```

```
        WHERE user_id = %s
    """, (st.session_state.user_id,))

    wallets = cursor.fetchall()

    # Display wallets
    if wallets:
        for wallet in wallets:
            with st.container():
                st.markdown(f"""
                <div class="card">
                    <h3>Wallet #{wallet['wallet_id']}</h3>
                                                                    <div
class="metric-value">${wallet['balance']:.2f}</div>
                                    <div  class="metric-label">Created:
{wallet['created_at']}</div>
                </div>
                """, unsafe_allow_html=True)

                # Get wallet transaction history
                cursor.execute("""
                    SELECT transaction_id, amount, timestamp,
                           CASE
                                        WHEN sender_wallet_id = %s THEN
'Outgoing'
                               ELSE 'Incoming'
                           END as direction
                    FROM Transactions
                     WHERE sender_wallet_id = %s OR receiver_wallet_id =
%s
                    ORDER BY timestamp DESC
                    LIMIT 5
                        """, (wallet['wallet_id'], wallet['wallet_id'],
wallet['wallet_id']))

                wallet_txs = cursor.fetchall()

                if wallet_txs:
                    wallet_tx_df = pd.DataFrame(wallet_txs)
                    st.dataframe(wallet_tx_df, hide_index=True)
                else:
                    st.info("No transactions for this wallet")
    else:
```

```
        st.error("You don't have any wallets")

    # Option to create a new wallet
    st.markdown("---")
    if st.button("Create New Wallet"):
        try:
                    cursor.execute("INSERT INTO Wallets (user_id, balance)
VALUES (%s, %s)",
                                   (st.session_state.user_id, 0.0))
            conn.commit()
            st.success("New wallet created successfully!")
            time.sleep(1)
            st.rerun()
        except Exception as e:
            st.error(f"Failed to create wallet: {e}")

# BLOCK EXPLORER PAGE
elif choice == "Block Explorer" and st.session_state.user_id:
    st.title("Block ")

    # Block navigation
    col1, col2 = st.columns([1, 3])

    with col1:
             search_type = st.selectbox("Search By", ["Block Number",
"Transaction Hash"])

        if search_type == "Block Number":
            # Get max block id
            cursor.execute("SELECT MAX(block_id) as max_id FROM Blocks")
            max_id = cursor.fetchone()['max_id'] or 0

                block_id = st.number_input("Block Number", min_value=1,
max_value=max_id, value=max_id)
            search = st.button("Search Block")

            if search:
                block_to_show = block_id
            else:
                block_to_show = max_id
        else:
            tx_hash = st.text_input("Transaction Hash")
            search = st.button("Search Transaction")
```

```
            block_to_show = None
            if search and tx_hash:
                  cursor.execute("SELECT block_id FROM Transactions WHERE
transaction_hash = %s", (tx_hash,))
                result = cursor.fetchone()
                if result:
                    block_to_show = result['block_id']
                    if not block_to_show:
                         st.info("This transaction is not yet included in
a block")
                else:
                    st.error("Transaction not found")

    # Visual blockchain representation
    st.markdown("### Blockchain")

    # Get last 10 blocks for visualization
    cursor.execute("""
          SELECT block_id, LEFT(block_hash, 8) as short_hash, timestamp,
nonce
        FROM Blocks
        ORDER BY block_id DESC
        LIMIT 10
    """)
    recent_blocks = cursor.fetchall()

    if recent_blocks:
        # Display blocks as connected cards
        cols = st.columns(min(5, len(recent_blocks)))
        for i, block in enumerate(recent_blocks[:5]):
            with cols[i]:
                st.markdown(f"""
                    <div class="card" style="text-align: center; cursor:
pointer;" onclick="alert('Block #{block['block_id']}')">
                                                    <div  style="font-size:
1.5rem;">#{block['block_id']}</div>
                                                                    <div
class="hash-text">{block['short_hash']}...</div>
                                                                    <div
class="metric-label">{block['timestamp'].strftime('%m/%d %H:%M')}</div>
                </div>
                """, unsafe_allow_html=True)
```

```
    # Block details
    if block_to_show:
        st.markdown("---")
        st.markdown("### Block Details")

        # Get block info
        cursor.execute("""
                    SELECT  b.block_id,  b.block_hash,  b.previous_block_id,
b.timestamp, b.nonce,
                     pb.block_hash as prev_hash
            FROM Blocks b
            LEFT JOIN Blocks pb ON b.previous_block_id = pb.block_id
            WHERE b.block_id = %s
        """, (block_to_show,))

        block = cursor.fetchone()

        if block:
            col1, col2 = st.columns(2)

            with col1:
                st.markdown(f"""
                <div class="card">
                    <h3>Block #{block['block_id']}</h3>
                                        <p><strong>Timestamp:</strong>
{block['timestamp']}</p>
                    <p><strong>Nonce:</strong> {block['nonce']}</p>
                                    <p><strong>Previous  Block:</strong>
#{block['previous_block_id'] or 'Genesis'}</p>
                </div>
                """, unsafe_allow_html=True)

            with col2:
                st.markdown(f"""
                <div class="card">
                    <h3>Block Hash</h3>
                    <p class="hash-text">{block['block_hash']}</p>
                    <h3>Previous Hash</h3>
                       <p class="hash-text">{block['prev_hash'] or '0' *
64}</p>
                </div>
                """, unsafe_allow_html=True)
```

```
        # Get transactions in this block
        cursor.execute("""
            SELECT t.transaction_hash, t.amount,
                   s.name as sender, r.name as receiver,
                   t.timestamp
            FROM Transactions t
            JOIN Wallets ws ON t.sender_wallet_id = ws.wallet_id
            JOIN Users s ON ws.user_id = s.user_id
            JOIN Wallets wr ON t.receiver_wallet_id = wr.wallet_id
            JOIN Users r ON wr.user_id = r.user_id
            WHERE t.block_id = %s
            ORDER BY t.timestamp
        """, (block_to_show,))

        block_txs = cursor.fetchall()

        st.markdown("### Block Transactions")

        if block_txs:
            tx_df = pd.DataFrame(block_txs)
            st.dataframe(tx_df, hide_index=True)
        else:
            st.info("No transactions in this block")
    else:
        st.error("Block not found")

# Optional: Blockchain integrity verification
with st.expander("Verify Blockchain Integrity"):
    if st.button("Verify Blockchain"):
        cursor.execute("""
            SELECT block_id, block_hash, previous_block_id, nonce
            FROM Blocks
            ORDER BY block_id
        """)
        all_blocks = cursor.fetchall()

        valid = True
        prev_hash = None

        for i, block in enumerate(all_blocks):
            # For genesis block
            if i == 0 and not block['previous_block_id']:
```

```
                    prev_hash = block['block_hash']
                    continue

                # Check if previous hash matches
                if block['previous_block_id']:
                    prev_block_index = block['previous_block_id'] - 1  # Adjust for 0-indexing
                    if prev_block_index < len(all_blocks):
                        expected_prev_hash = all_blocks[prev_block_index]['block_hash']
                        if expected_prev_hash != prev_hash:
                            valid = False
                            break

                prev_hash = block['block_hash']

            if valid:
                st.success("Blockchain integrity verified! All blocks are correctly linked.")
            else:
                st.error("Blockchain integrity check failed. Chain may be compromised.")

# PROFILE SETTINGS PAGE
elif choice == "Profile Settings" and st.session_state.user_id:
    st.title("Profile Settings")

    # Get user info
    cursor.execute("SELECT name, email FROM Users WHERE user_id = %s", (st.session_state.user_id,))
    user = cursor.fetchone()

    col1, col2 = st.columns(2)

    with col1:
        st.markdown("""
        <div class="card">
            <h3>Profile Information</h3>
        </div>
        """, unsafe_allow_html=True)

        with st.form("profile_form"):
            name = st.text_input("Username", value=user['name'])
```

```
            email = st.text_input("Email", value=user['email'])

            update_profile = st.form_submit_button("Update Profile")

            if update_profile:
                try:
                     cursor.execute("UPDATE Users SET name = %s, email =
%s WHERE user_id = %s",
                                                              (name,  email,
st.session_state.user_id))
                    conn.commit()
                    st.success("Profile updated successfully!")
                except Exception as e:
                    st.error(f"Failed to update profile: {e}")

    with col2:
        st.markdown("""
        <div class="card">
            <h3>Change Password</h3>
        </div>
        """, unsafe_allow_html=True)

        with st.form("password_form"):
                    current_password  =  st.text_input("Current  Password",
type="password")
                         new_password   =   st.text_input("New   Password",
type="password")
             confirm_new_password = st.text_input("Confirm New Password",
type="password")

            update_password = st.form_submit_button("Change Password")

            if update_password:
                     if not current_password or not new_password or not
confirm_new_password:
                    st.error("Please fill in all password fields")
                elif new_password != confirm_new_password:
                    st.error("New passwords do not match")
                elif len(new_password) < 8:
                        st.error("Password must be at least 8 characters
long")
                else:
                    # Verify current password
```

```
                    hashed_current = hash_password(current_password)
                        cursor.execute("SELECT user_id FROM Users WHERE
user_id = %s AND password = %s",
                                                (st.session_state.user_id,
hashed_current))

                    if cursor.fetchone():
                        try:
                            hashed_new = hash_password(new_password)
                             cursor.execute("UPDATE Users SET password =
%s WHERE user_id = %s",
                                                                (hashed_new,
st.session_state.user_id))
                            conn.commit()
                            st.success("Password changed successfully!")
                        except Exception as e:
                            st.error(f"Failed to update password: {e}")
                    else:
                        st.error("Current password is incorrect")

    # Account security settings
    st.markdown("---")
    st.markdown("""
    <div class="card">
        <h3>Security Settings</h3>
    </div>
    """, unsafe_allow_html=True)

    col1, col2 = st.columns(2)

    with col1:
        st.markdown("#### Session Timeout")
          timeout = st.slider("Session Timeout (minutes)", min_value=5,
max_value=60, value=15, step=5)

        if st.button("Update Timeout"):
            st.success(f"Session timeout updated to {timeout} minutes")

    with col2:
        st.markdown("#### Two-Factor Authentication")
        enable_2fa = st.checkbox("Enable 2FA (Preview Only)")

        if enable_2fa:
```

```
            st.info("Two-factor authentication feature coming soon")

# LOGOUT PAGE
elif choice == "Logout":
    st.session_state.user_id = None
    st.success("You have been logged out.")
    time.sleep(1)
    st.rerun()

# Handle Footer
st.markdown("---")
st.markdown("<div     class='footer'>©     Decentralized     Transaction
Verification System | DBS Team A15 </div>", unsafe_allow_html=True)

# Close DB connection when app is done
conn.close()
```

Chapter 6

6. Result

Homepage

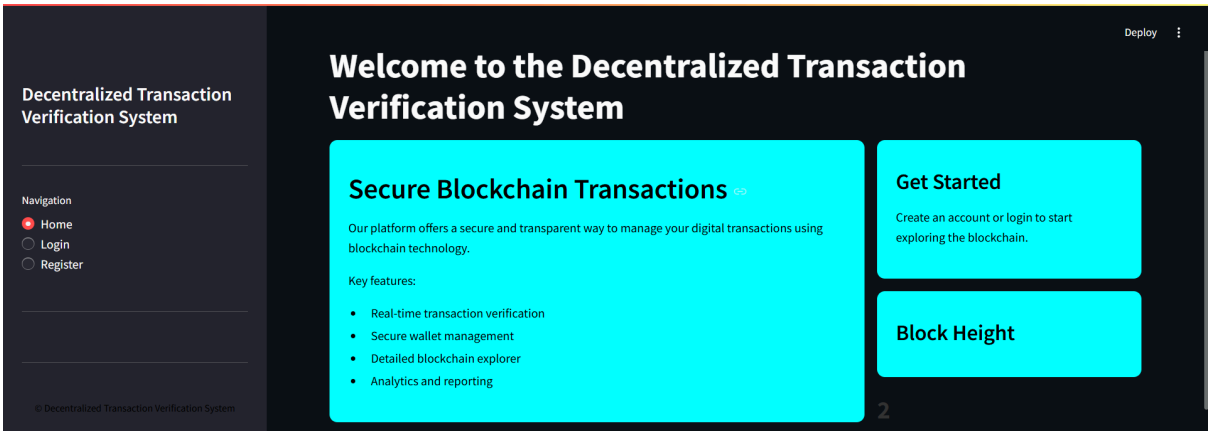

Figure 6.1 Homepage

Recent Transactions

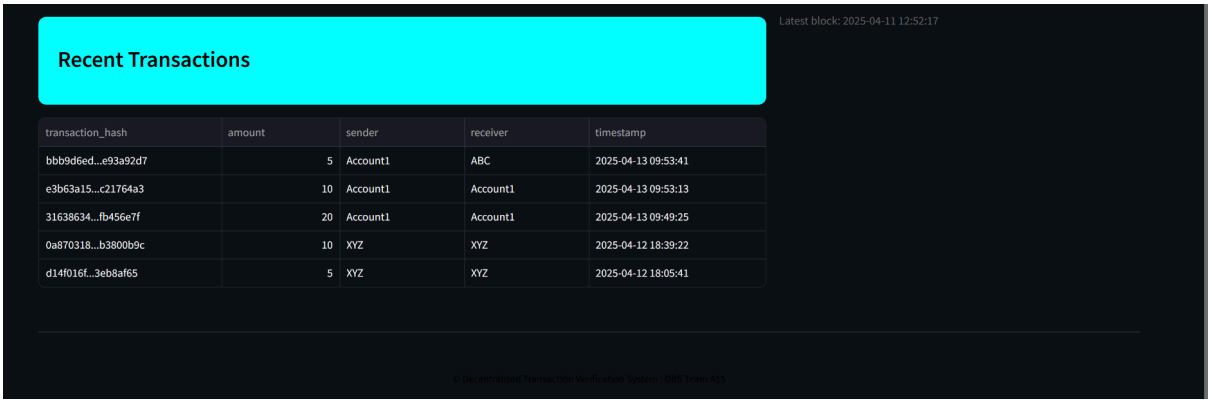

Figure 6.2 Recent Transactions

New Account Creation

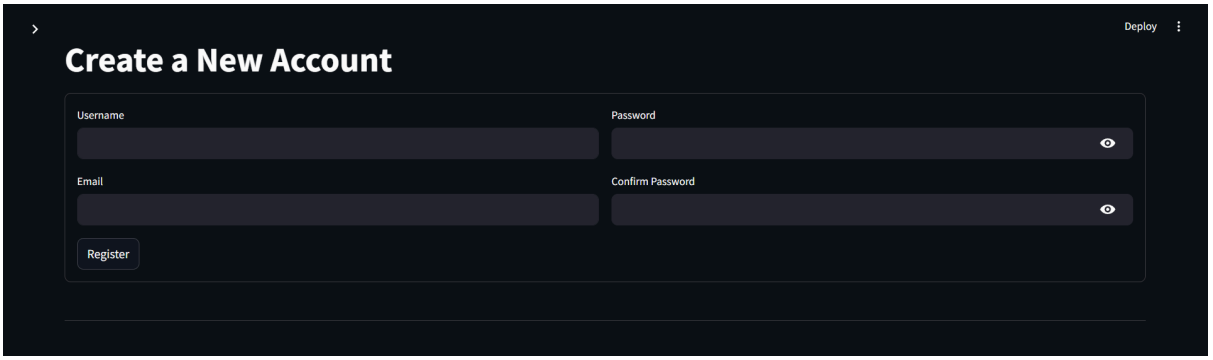

Figure 6.3 New Account

User Dashboard

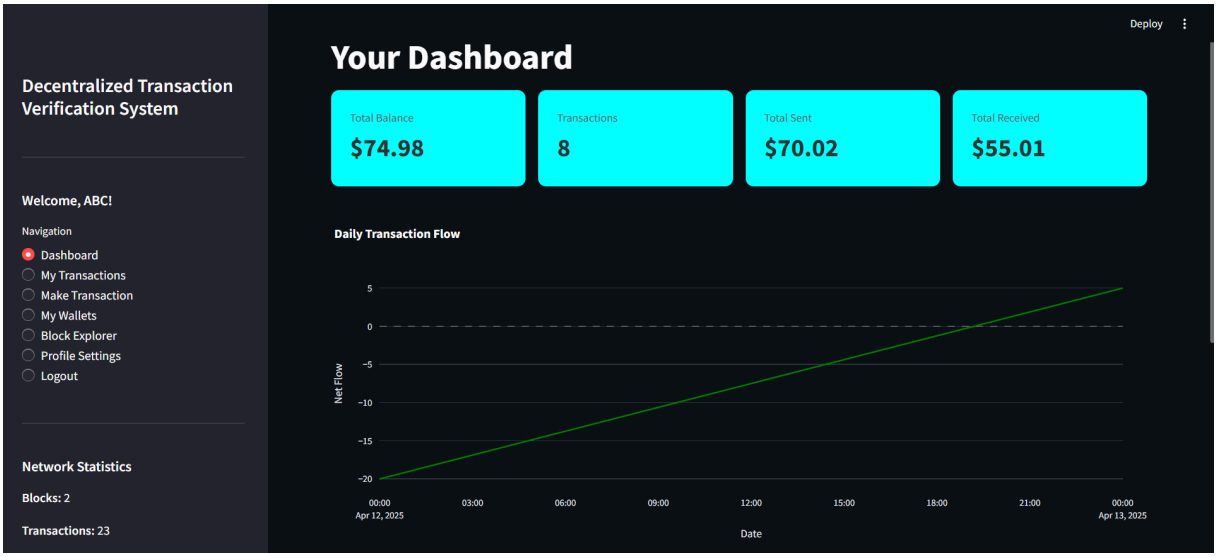

Figure 6.4 Dashboard

User Transaction List

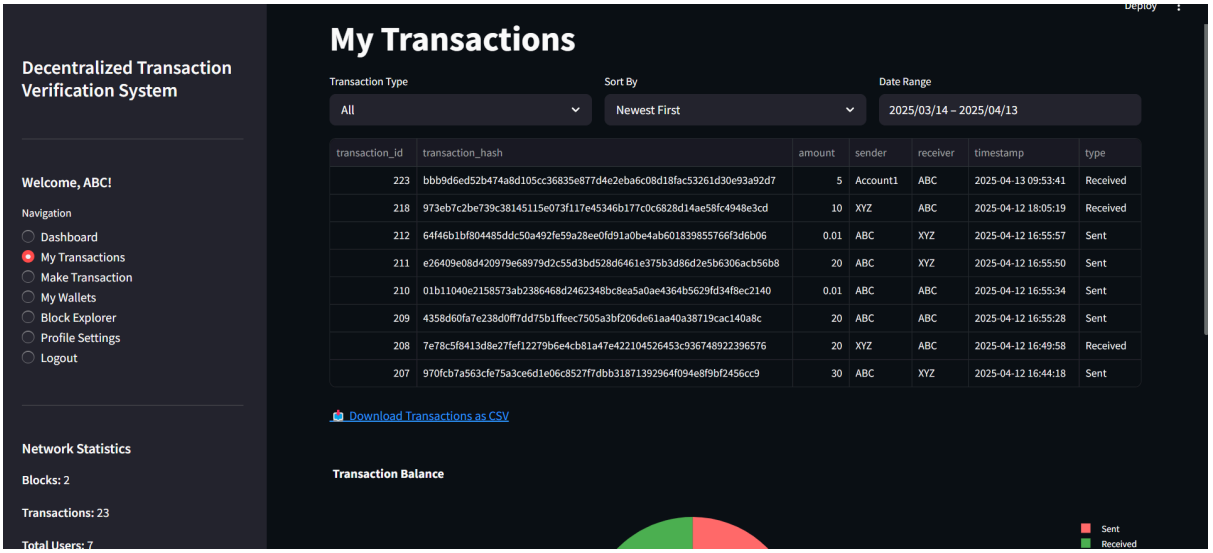

Figure 6.5 Transactions

User Send/Receive ratio represented graphically

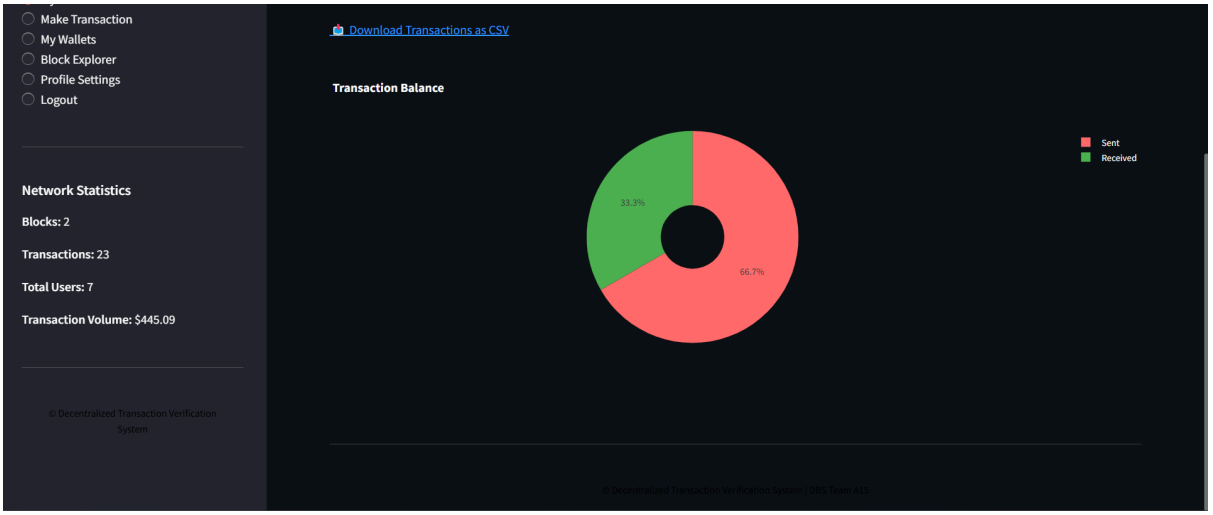

Figure 6.6 Send/Recieve Ratio

User wallets summary

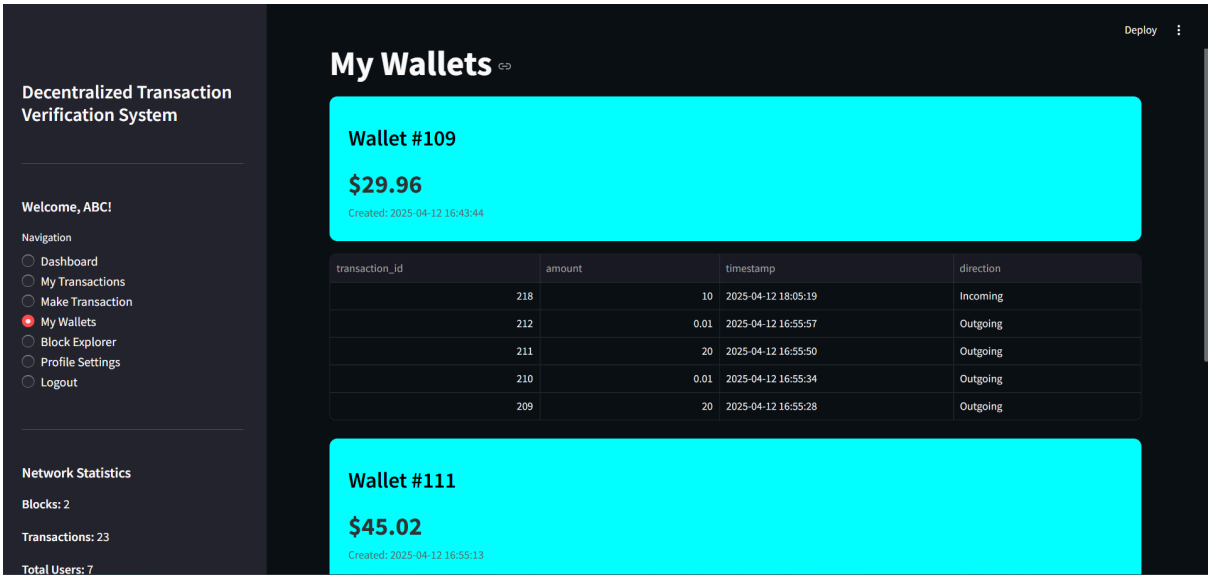

Figure 6.7 Wallet Summary

Block Information

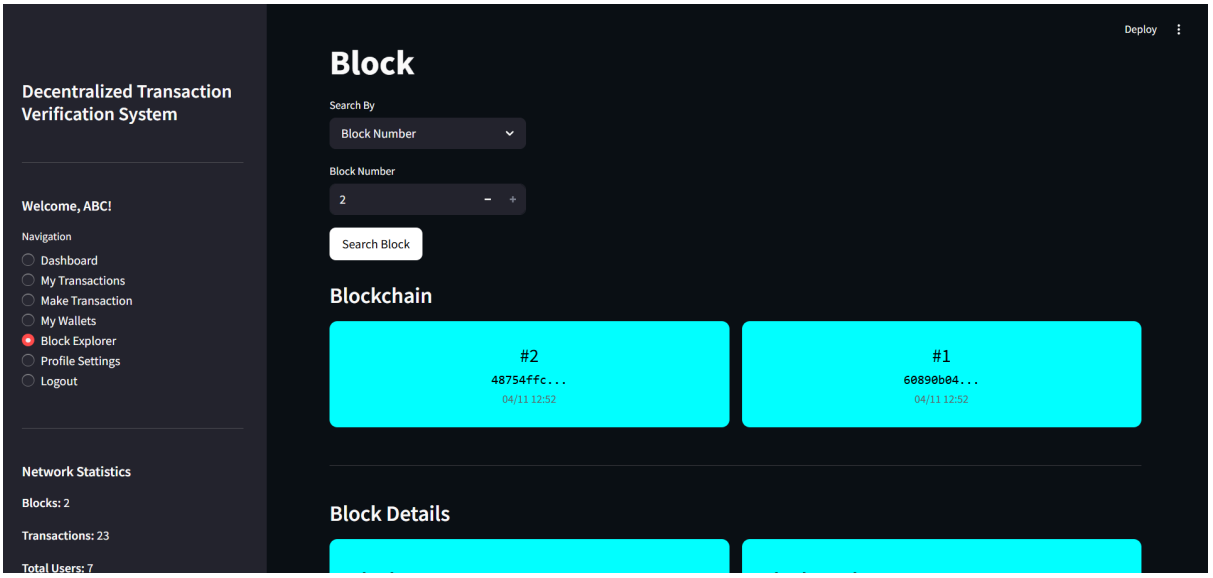

Figure 6.8 Block

Block Transactions and security verification

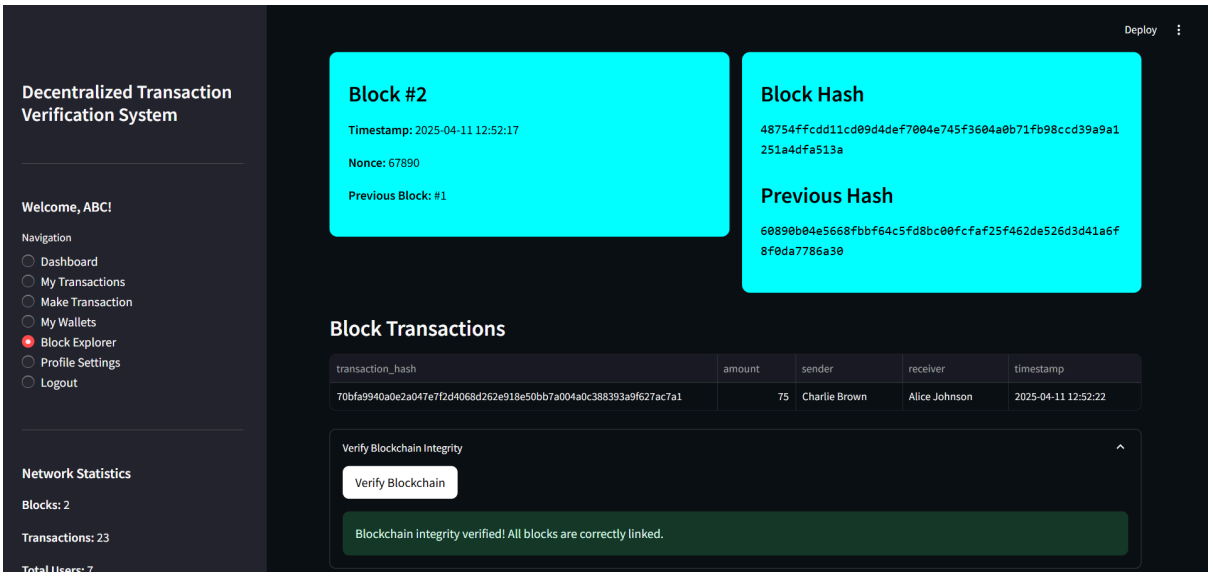

Figure 6.9 Block Transactions

Profile Settings and option to reset password

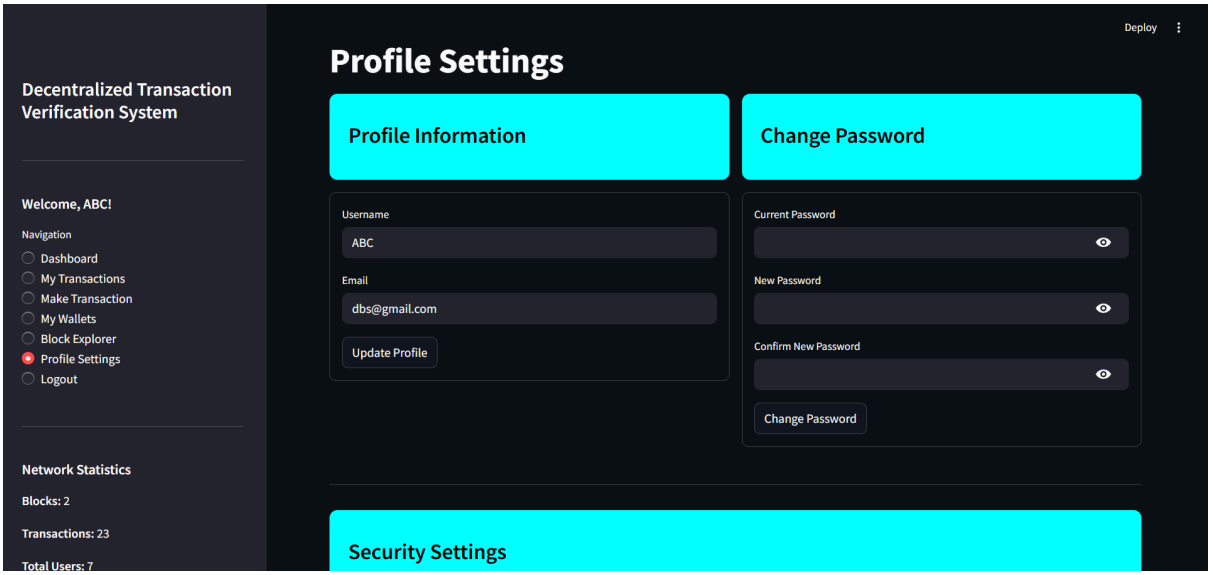

Figure 6.10 Profile Settings

Transaction page (sending to same user's different wallet)

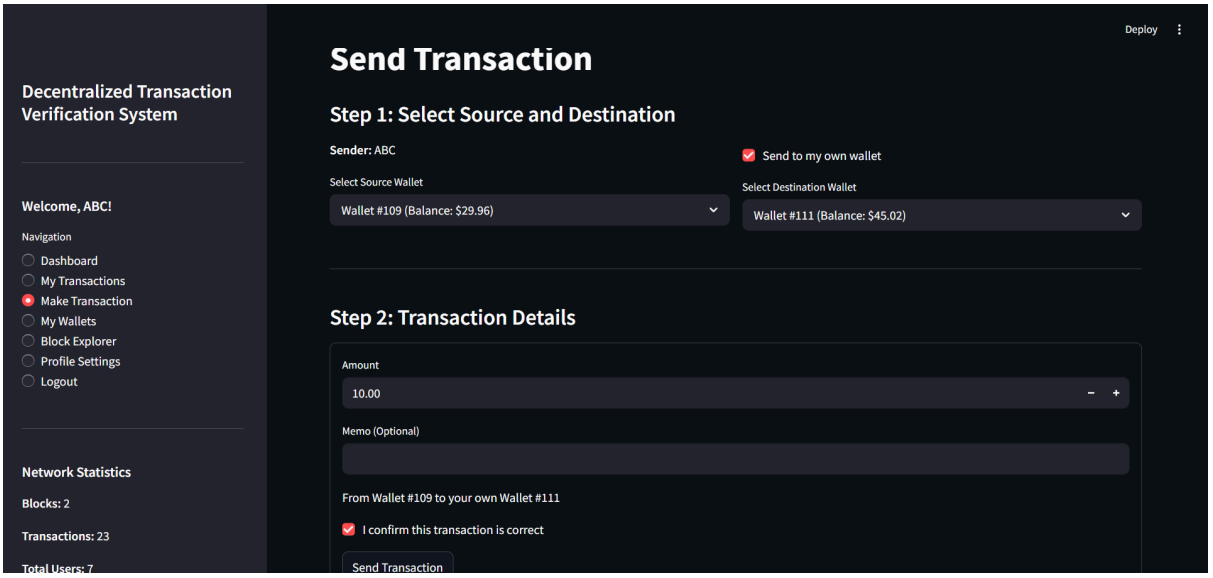

Figure 6.11 Transaction page (sending to same user's different wallet)

Transaction page (sending to different user)

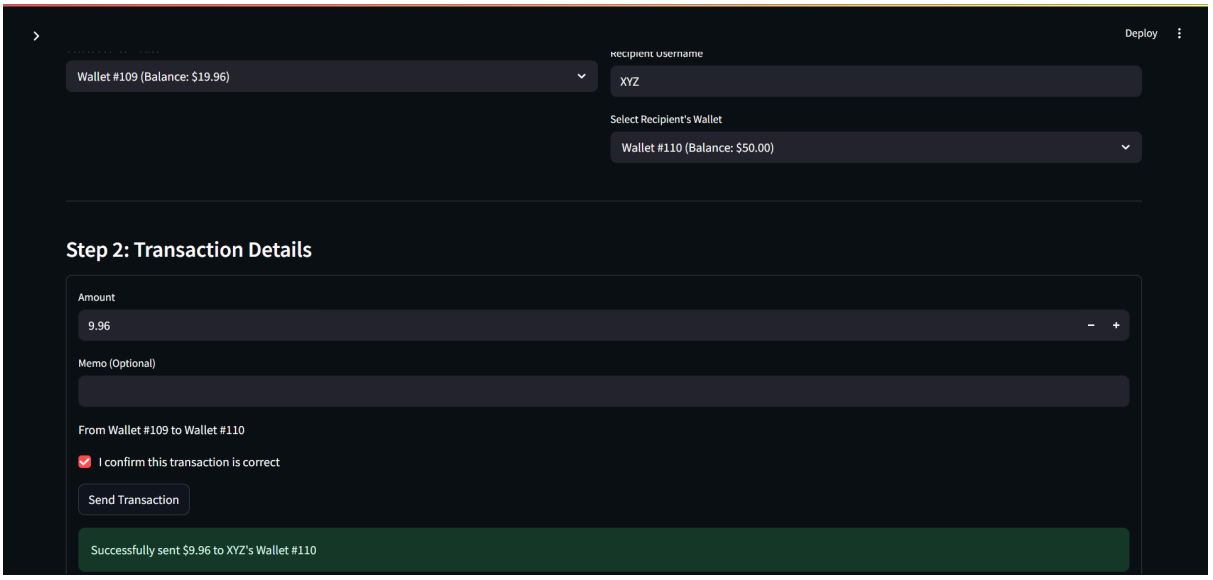

Figure 6.12 Transaction page (sending to different user)

Creating Account with Existing Email

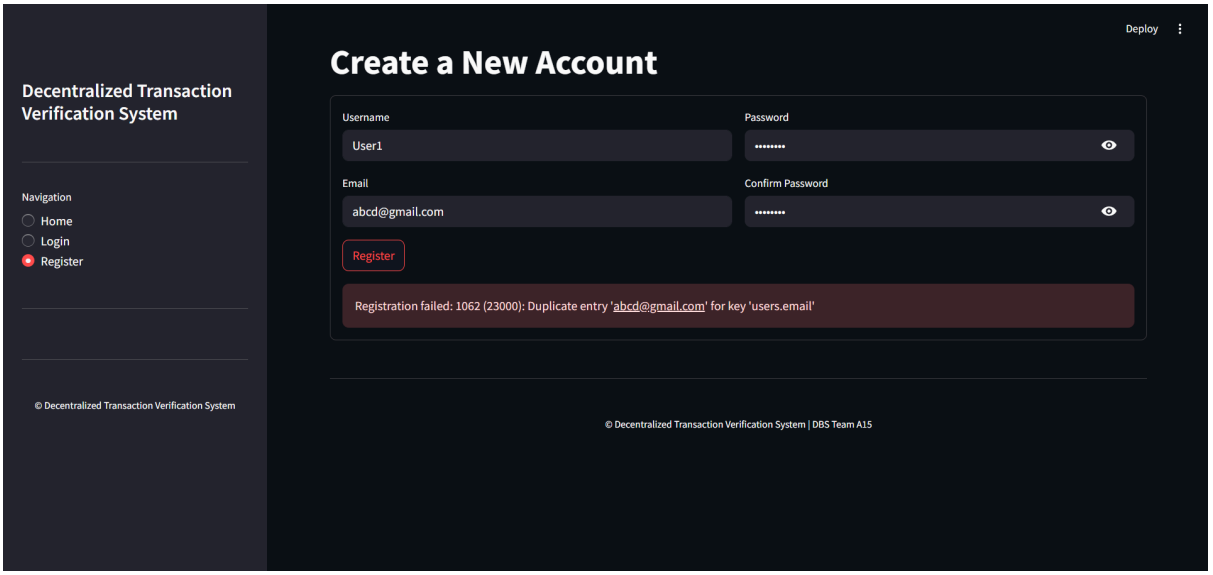

Figure 6.13 Creating Account with Existing Email

```
mysql> select * from transactions;
+----------------+----------+--------------+------------------+--------------------+---------------------+------------------------------------------------------------------
----------------+------------+------+---------+
| transaction_id | block_id | amount       | sender_wallet_id | receiver_wallet_id | timestamp           | transaction_hash
               | fee        | memo | status  |
+----------------+----------+--------------+------------------+--------------------+---------------------+------------------------------------------------------------------
----------------+------------+------+---------+
|            201 |        1 |  50.00000000 |              101 |                103 | 2025-04-11 12:52:22 | 5235dde3806c7f929ad562925c85905afaa495c2c3de6ccff
50a340ee2af6b1c | 0.00000000 | NULL | pending |
|            202 |        1 | 100.00000000 |              103 |                104 | 2025-04-11 12:52:22 | f8bda198c44c47ae6ee074ed38aa1044c5b72231748a4637a
03989ee26bed23f | 0.00000000 | NULL | pending |
|            203 |        2 |  75.00000000 |              104 |                102 | 2025-04-11 12:52:22 | 70bfa9940a0e2a047e7f2d4068d262e918e50bb7a004a0c38
8393a9f627ac7a1 | 0.00000000 | NULL | pending |
|            204 |     NULL |   0.01000000 |              103 |                101 | 2025-04-11 13:20:06 | b85a53c667c1141024928bf9fbfe7df248c5975bf5c9abb74
8df89c313a451e8 | 0.00000000 | NULL | pending |
|            205 |     NULL |   0.04000000 |              101 |                103 | 2025-04-12 16:14:20 | 7be88487ae65439a64745b7f9665d26271149b99509070f9d
2324f3588512420 | 0.00000000 | NULL | pending |
|            206 |     NULL |  20.00000000 |              108 |                101 | 2025-04-12 16:42:48 | 6527117f4d905444dab09e6ccac023553fc7f809fc57ba821
b25d45866c8a5f4 | 0.00000000 | NULL | pending |
|            207 |     NULL |  30.00000000 |              109 |                108 | 2025-04-12 16:44:18 | 970fcb7a563cfe75a3ce6d1e06c8527f7dbb31871392964f0
94e8f9bf2456cc9 | 0.00000000 | NULL | pending |
|            208 |     NULL |  20.00000000 |              108 |                109 | 2025-04-12 16:49:58 | 7e78c5f8413d8e27fef12279b6e4cb81a47e422104526453c
936748922396576 | 0.00000000 | NULL | pending |
|            209 |     NULL |  20.00000000 |              109 |                111 | 2025-04-12 16:55:28 | 4358d60fa7e238d0ff7dd75b1ffeec7505a3bf206de61aa40
a38719cac140a8c | 0.00000000 |      | pending |
|            210 |     NULL |   0.01000000 |              109 |                111 | 2025-04-12 16:55:34 | 01b11040e2158573ab2386468d2462348bc8ea5a0ae4364b5
629fd34f8ec2140 | 0.00000000 |      | pending |
|            211 |     NULL |  20.00000000 |              109 |                108 | 2025-04-12 16:55:50 | e26409e08d420979e68979d2c55d3bd528d6461e375b3d86d
2e5b6306acb56b8 | 0.00000000 |      | pending |
|            212 |     NULL |   0.01000000 |              109 |                108 | 2025-04-12 16:55:57 | 64f46b1bf804485ddc50a492fe59a28ee0fd91a0be4ab6018
39855766f3d6b06 | 0.00000000 |      | pending |
|            213 |     NULL |  20.00000000 |              108 |                110 | 2025-04-12 16:58:04 | 313c72f6da81874f3d81da80c6435647944ed5ab42e6c77fb
5ba413798aafd1c | 0.00000000 |      | pending |
|            214 |     NULL |   0.01000000 |              108 |                110 | 2025-04-12 16:58:10 | e01d3716a0098e4e94cdcfbf9d4d4ed83af09162c790e2c5b
5dd78449a06a7b9 | 0.00000000 |      | pending |
|            215 |     NULL |   0.01000000 |              110 |                101 | 2025-04-12 16:58:51 | 32f8c158f61d09a6c222af0c1cefea2b2d0ee54391bcb56a8
45f29cdc4d64692 | 0.00000000 |      | pending |
|            216 |     NULL |  20.00000000 |              110 |                108 | 2025-04-12 17:54:56 | 11fe97274071e67c9f58fd48a5b675f7d8fff9994379a8d16
b3e55e44fb28a1d | 0.00000000 |      | pending |
|            217 |     NULL |  10.00000000 |              108 |                110 | 2025-04-12 17:59:36 | 486b589a9cb0817f3210c94e6d8cc6d23757bade077a81386
```

Figure 6.14 Transactions Data Table

Trigger action when Amount is more than Balance

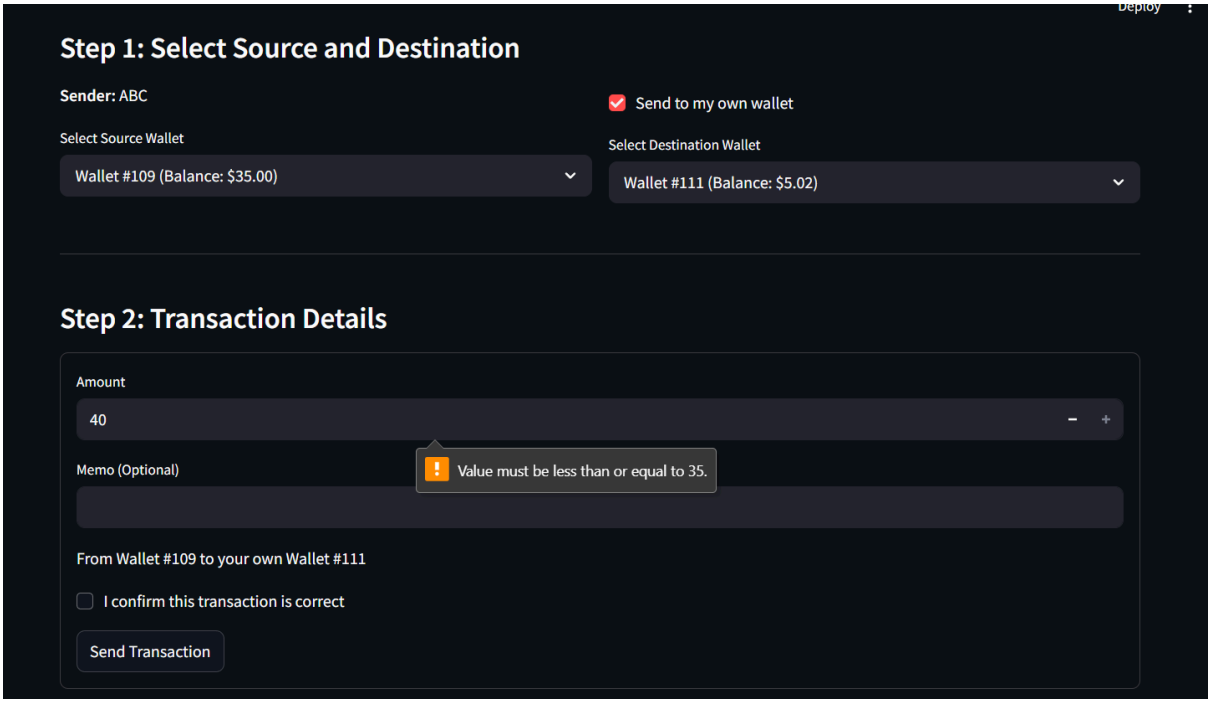

Figure 6.15 Trigger action when Amount is more than Balance

Chapter 7

7. Conclusion and Future Work

7.1 Conclusion

This project successfully demonstrates the core principles of blockchain technology using MySQL. By simulating key features such as block mining, transaction handling with fees and memos, reward distribution, and integrity enforcement through triggers and stored procedures, it provides a foundational understanding of how decentralized ledger systems operate. The use of SQL-based logic not only makes the concept accessible for database enthusiasts but also highlights the flexibility of relational databases in mimicking real-world blockchain behavior. This simulation lays the groundwork for more advanced implementations, serving as a valuable educational and experimental tool in the growing field of blockchain systems.

7.2 Scope for Future Work

7.2.1 Smart Contract Simulation: Extend the system to support smart contracts by allowing users to define programmable conditions for automated transactions.

7.2.2 Consensus Mechanism Integration: Implement consensus algorithms like Proof-of-Work (PoW), Proof-of-Stake (PoS), or Delegated Proof-of-Stake (DPoS) to make block validation more realistic and decentralized.

7.2.3 Wallet and Encryption System: Incorporate digital wallet management and basic public-private key encryption for secure identity and transaction handling.

7.2.4 Transaction Explorer UI: Develop a web-based dashboard to visually track blocks, transactions, miner rewards, and user balances for better user interaction.

7.2.5 Scalability Testing: Conduct performance analysis and optimize SQL procedures and triggers to handle larger volumes of transactions and users efficiently.

7.2.6 Fraud Detection and Anomaly Monitoring: Integrate triggers or stored procedures for identifying suspicious transaction patterns or attempts to exploit the system.

7.2.7 Integration with External APIs: Allow the system to simulate real-world usage by integrating with APIs for currency conversion, external data triggers, or other financial services.

7.2.8 Backup and Recovery Mechanisms: Add procedures to ensure secure backup of blockchain data and implement a recovery system in case of data corruption.

7.2.9 User Authentication and Access Control: Add login functionality and role-based access to differentiate between miners, regular users, and administrators. Also scope to add two-factor authentication.

7.2.10 Blockchain Fork Handling: Simulate forks in the blockchain and design conflict resolution strategies to handle competing chains.

Each Team Member Contribution:

Team Member	Reg No	Contribution
Piyush Verma	230968114	Worked on ER/Schema diagram, Frontend/integration, Backend and the Report
Kewal Thacker	230968126	Worked on Abstract, ER/Schema diagram, Frontend/integration, Backend and the Report
Anij Mehta	230968116	Worked on ER/Schema diagram, Frontend/integration and the Report
Purav Goyal	230968118	Worked on ER/Schema diagram and the Report
Kilaru Pardha Sai	230968134	Worked on Abstract and Report
Aditya Jain	230968132	Worked on Schema Diagram and Report