

DIGITAL WALLET SYSTEM



A PROJECT REPORT

Submitted by

SHAHUL HAMEED M (2303811724321101)

in partial fulfillment of requirements for the award of the course

CGB1221-DATABASE MANAGEMENT SYSTEMS

in

ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

K. RAMAKRISHNAN COLLEGE OF TECHNOLOGY

(An Autonomous Institution, affiliated to Anna University Chennai and Approved by AICTE, New Delhi)

SAMAYAPURAM – 621 112

JUNE- 2025

**K. RAMAKRISHNAN COLLEGE OF TECHNOLOGY
(AUTONOMOUS)**

SAMAYAPURAM – 621 112

BONAFIDE CERTIFICATE

Certified that this project report on “**DIGITAL WALLET SYSTEM**”
is the bonafide work of **SHAHUL HAMEED M (2303811724321101)**
who carried out the project work during the academic year 2024 - 2025
under my supervision.

SIGNATURE

Dr.T. AVUDAIAPPAN, M.E.,Ph.D.,

HEAD OF THE DEPARTMENT

ASSOCIATE PROFESSOR

Department of Artificial Intelligence

K.Ramakrishnan College of Technology
(Autonomous)

Samayapuram–621112.

SIGNATURE

Mrs.S. GEETHA, M.E.,

SUPERVISOR

ASSISTANT PROFESSOR

Department of Artificial Intelligence

K.Ramakrishnan College of Technology
(Autonomous)

Samayapuram–621112.

Submitted for the viva-voce examination held on 04.06.2025

INTERNAL EXAMINER

EXTERNAL EXAMINER

DECLARATION

I declare that the project report on “**DIGITAL WALLET SYSTEM**” is the result of original work done by me and best of my knowledge, similar work has not been submitted to “**ANNA UNIVERSITY CHENNAI**” for the requirement of Degree of **BACHELOR OF TECHNOLOGY**. This project report is submitted on the partial fulfilment of the requirement of the completion of the course **CGB1221 – DATABASE MANAGEMENT SYSTEMS**.

Signature

SHAHUL HAMEEDM

Place: Samayapuram

Date: 04.06.2025

ACKNOWLEDGEMENT

It is with great pride that I express our gratitude and in-debt to our institution “**K.Ramakrishnan College of Technology (Autonomous)**”, for providing us with the opportunity to do this project.

I glad to credit honourable chairman **Dr. K. RAMAKRISHNAN, B.E.**, for having provided for the facilities during the course of our study in college.

I would like to express our sincere thanks to our beloved Executive Director **Dr. S. KUPPUSAMY, MBA, Ph.D.**, for forwarding to our project and offering adequate duration in completing our project.

I would like to thank **Dr. N. VASUDEVAN, M.Tech., Ph.D.**, Principal, who gave opportunity to frame the project the full satisfaction.

I whole heartily thanks to **Dr. T. AVUDAIAPPAN, M.E.,Ph.D.**, Head of the department, **ARTIFICIAL INTELLIGENCE** for providing his encourage pursuing this project.

I express our deep expression and sincere gratitude to our project supervisor **Mrs.S.GEETHA, M.E.**, Department of **ARTIFICIAL INTELLIGENCE**, for her incalculable suggestions, creativity, assistance and patience which motivated us to carry out this project.

I render our sincere thanks to Course Coordinator and other staff members for providing valuable information during the course.

I wish to express our special thanks to the officials and Lab Technicians of our departments who rendered their help during the period of the work progress.

INSTITUTE

Vision:

- To serve the society by offering top-notch technical education on par with global standards.

Mission:

- Be a center of excellence for technical education in emerging technologies by exceeding the needs of industry and society.
- Be an institute with world class research facilities.
- Be an institute nurturing talent and enhancing competency of students to transform them as all – round personalities respecting moral and ethical values.

DEPARTMENT

Vision:

- To excel in education, innovation, and research in Artificial Intelligence and Data Science to fulfil industrial demands and societal expectations.

Mission

- To educate future engineers with solid fundamentals, continually improving teaching methods using modern tools.
- To collaborate with industry and offer top-notch facilities in a conducive learning environment.
- To foster skilled engineers and ethical innovation in AI and Data Science for global recognition and impactful research.
- To tackle the societal challenge of producing capable professionals by instilling employability skills and human values.

PROGRAM EDUCATIONAL OBJECTIVES (PEO)

- **PEO1:** Compete on a global scale for a professional career in Artificial Intelligence and Data Science.
- **PEO2:** Provide industry-specific solutions for the society with effective communication and ethics.
- **PEO3** Enhance their professional skills through research and lifelong learning initiatives.

PROGRAM SPECIFIC OUTCOMES (PSOs)

- **PSO1:** Capable of finding the important factors in large datasets, simplify the data, and improve predictive model accuracy.
- **PSO2:** Capable of analyzing and providing a solution to a given real-world problem by designing an effective program.

PROGRAM OUTCOMES (POs)

Engineering students will be able to:

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development

- 8. Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
- 9. Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
- 10. Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
- 11. Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
- 12. Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

ABSTRACT

The **Digital Wallet System** is a Flask-based web application developed using Python and SQLite to facilitate secure and efficient personal financial management. This system allows users to add funds, spend money, and perform simulated UPI-based transfers while maintaining a real-time transaction history. Each transaction is recorded with its amount, type (Credit, Debit, or Transfer), description, and timestamp, providing full transparency and traceability. To ensure security and accountability, the system incorporates a **digital ID verification** feature. Users must register at least one digital identity before performing any spend or transfer operations. This added layer of validation helps simulate real-world digital wallet systems where identity verification is essential. All data is persistently stored using **SQLite**, with structured tables for wallet balance, transaction logs, and digital ID records. The application's API endpoints support operations like balance inquiry, transaction retrieval, digital ID management, and fund handling. Designed with a modular approach, the system can be further extended to include authentication, notification systems, or integration with external payment gateways.

ABSTRACT WITH POs AND PSOs MAPPING

CO 5 : BUILD DATABASES FOR SOLVING REAL-TIME PROBLEMS.

ABSTRACT	POs MAPPED	PSOs MAPPED
<p>The Digital Wallet System is a Python-based desktop application using Tkinter for the user interface and MySQL for data storage. It allows users to create accounts, add funds, and perform transactions such as sending or receiving money. The system ensures secure authentication, maintains transaction history, and provides balance tracking. It offers a convenient and secure platform for managing digital payments and financial records.</p>	<p>PO1-3 PO2-3 PO3-3 PO4-3 PO5 -3 PO6-3 PO7-3 PO8-3 PO9-3 PO10-3 PO11-3 PO12-3</p>	<p>PSO1 - 3 PSO2 - 3</p>

Note: 1- Low, 2-Medium, 3- High

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	ABSTRACT	viii
1	INTRODUCTION	1
	1.1 OBJECTIVE	1
	1.2 OVERVIEW	1
	1.3 SQL AND DATABASE CONCEPT	1
2	PROJECT METHODOLOGY	4
	2.1 PROPOSED WORK	4
	2.2 BLOCK DIAGRAM	4
3	MODULE DESCRIPTION	5
	3.1 DATABASE MODULE	6
	3.2 BALANCE MANAGEMENT MODULE	6
	3.3 TRANSACTION MANAGEMENT MODULE	7
	3.4 DIGITAL ID MODULE	7
	3.5 API MODULE	7
4	CONCLUSION & FUTURE SCOPE	8
	APPENDIX A SOURCE CODE	9
	APPENDIX B SCREENSHOTS	16
	REFERENCES	18

CHAPTER 1

INTRODUCTION

1.1 OBJECTIVE

The Digital Wallet System is a lightweight Flask-based web application designed to facilitate the management of personal finances in a secure and user-friendly manner. It allows users to add, spend, and transfer funds while maintaining a transaction history and verifying the presence of digital identification before any financial operation. Built with Python and SQLite, the application is structured to serve both as a practical digital wallet and as an educational tool for understanding web-based CRUD operations.

1.2 OVERVIEW

The Digital Wallet System is implemented as a Flask-based web application that interacts with a SQLite database. It offers API endpoints to perform operations such as checking the balance, adding funds, spending money, sending money via UPI, and managing digital ID information. Upon application startup, the system initializes the database and ensures required tables are created (transactions, balance, digital_ids). All user actions are stored persistently, and transaction data is timestamped for historical recordkeeping.

Users interact with the wallet via HTTP requests, and responses are returned in JSON format, making it suitable for future integration with mobile apps or front-end frameworks. Furthermore, a simple home page (index.html) is rendered to serve as a starting point for UI development. To ensure safety and accountability, the system mandates that a user must have a registered digital ID before performing any financial transactions like spending or transferring money.

1.3 SQL AND DATABASE CONCEPTS

The Digital Wallet System uses SQLite to manage data related to user balance, transactions, and digital IDs. It initializes necessary tables at startup and maintains a single balance record that is updated with each transaction. All financial activities—such as adding, spending, or transferring money—are logged in the transactions table with timestamps and descriptions for transparency. Before allowing sensitive actions, the system checks for a valid digital ID in the digital_ids table, enhancing security. The system employs core SQL operations like INSERT, SELECT, and UPDATE to handle all database interactions, demonstrating effective use of CRUD operations in a real-world finance application.

CHAPTER 2

PROJECT METHODOLOGY

2.1 PROPOSED WORK

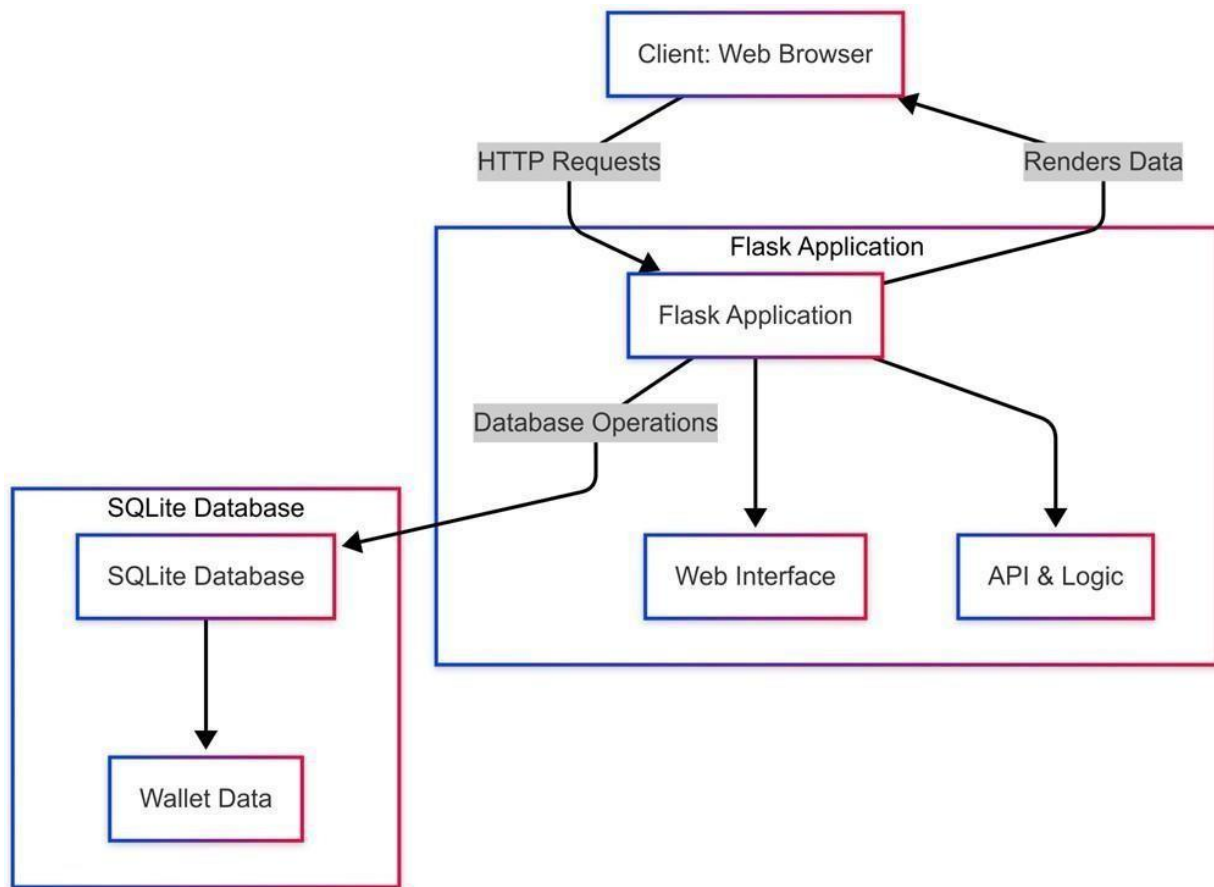
The proposed system is a lightweight and efficient Digital Wallet application built using the Flask web framework, aimed at providing users with a simple yet secure platform for managing their personal finances. The system leverages SQLite as the backend database to store critical data such as account balance, transaction history, and digital identification details, ensuring data consistency and integrity.

Users can easily add funds to their wallet, spend money, and transfer amounts to other users by specifying their UPI ID, all through clearly defined REST API endpoints. The system requires users to have at least one digital ID registered before permitting any spending or transfer operations, adding a layer of security and verification. Each financial transaction is recorded in the database with details including amount, description, type (credit, debit, transfer), and timestamp, which helps in maintaining an accurate transaction history.

The application ensures real-time balance updates and supports retrieval of transaction and digital ID data through API calls, facilitating smooth integration with front-end interfaces. Parameterized SQL queries are used throughout to prevent SQL injection attacks, while database commits guarantee reliable and atomic updates. The overall design offers users a seamless experience to manage digital funds, track transaction history, and maintain verified digital identities, making the system both practical and secure.

.

2.2 BLOCK DIAGRAM



CHAPTER 3

MODULE DESCRIPTION

3.1 DATABASE MANAGEMENT MODULE

The Database Module serves as the foundation of the Digital Wallet system. It uses SQLite to manage persistent storage through a local file named wallet.db. Upon initialization, it creates three essential tables: transactions, balance, and digital_ids. The transactions table logs all financial activities with attributes such as amount, description, type (Credit, Debit, or Transfer), and timestamp. The balance table maintains the current wallet balance and is initialized with a default amount of 0.0. The digital_ids table stores digital identity types and numbers (such as Aadhaar, PAN, or UPI ID). This module is invoked at the start of the application to ensure the database structure is properly set up.

3.2 BALANCE MANAGEMENT MODULE

This module is responsible for managing the financial balance within the wallet. It provides functionality to retrieve the current balance through the /balance endpoint. Users can add funds using the /add route, which updates the balance and logs the transaction as a credit. Spending funds is handled through the /spend endpoint, which deducts the specified amount only if a digital ID is present and sufficient funds exist. Additionally, the /send route allows users to transfer funds to a specified UPI ID, also updating the balance and recording the transaction. This module ensures that all balance updates are secure, validated, and properly recorded.

3.3 TRANSACTION MANAGEMENT MODULE

The Transaction Management Module handles the recording and retrieval of all wallet transactions. Whenever funds are added, spent, or transferred, a corresponding entry is inserted into the transactions table. Each entry includes the

transaction amount, a brief description, the type of transaction (Credit, Debit, or Transfer), and a timestamp. Users can retrieve the complete history of transactions via the `/transactions` endpoint, which returns the list in descending order of date. This module ensures transparency and traceability of every financial operation conducted within the wallet.

3.4 DIGITAL ID MODULE

This module manages the user's digital identity records linked to the wallet. Through the `/add_id` endpoint, users can add various digital IDs like Aadhaar, PAN, or UPI numbers. The system mandates the presence of at least one digital ID to authorize spending or transferring funds, adding a layer of security and compliance. Users can retrieve their stored digital IDs using the `/digital_ids` endpoint. The presence of this module ensures that the wallet adheres to identification standards and prevents unauthorized financial activity.

3.5 API MODULE

The API Module encapsulates all the RESTful endpoints of the application. It defines how users and external applications interact with the system. Each operation, from checking the balance to adding digital IDs, is accessible through clearly defined routes such as `/balance`, `/add`, `/spend`, `/send`, and others. JSON is used for both requests and responses, ensuring that the API is lightweight and easy to integrate with frontend or third-party applications. This module is crucial for enabling seamless communication between the user interface and the backend services.

CHAPTER 4

CONCLUSION & FUTURE SCOPE

CONCLUSION

The Digital Wallet System successfully demonstrates a secure, lightweight, and responsive financial management application using Flask as the backend framework and SQLite as the database engine. The system includes essential modules such as balance tracking, transaction history, fund transfers, and digital ID management, all accessible through RESTful API endpoints. Each module is integrated with robust error checking, validation, and transactional integrity to ensure user trust and operational reliability. The use of JSON-based communication enables smooth interaction between clients and the server, while the initialization logic ensures the database is ready upon deployment. Overall, the project showcases the effective use of Python's Flask framework to build a modular, extensible, and secure financial application suited for personal or prototype-level digital wallet operations.

FUTURE SCOPE

The current implementation of the Digital Wallet System offers a strong foundational structure, but there are several opportunities for enhancement. In future iterations, user authentication features such as secure login and registration can be introduced to allow personalized wallet management. The web interface can be expanded into a full-fledged responsive dashboard using modern front-end frameworks like React or Vue.js. Integration with third-party APIs, such as real-time currency conversion or UPI verification, would provide extended functionality. Adding analytics features such as monthly spending reports, transaction filtering, and visual charts can improve usability. Support for QR code scanning, mobile app deployment (using Flutter or React Native), and integration with cloud databases (e.g., Firebase or PostgreSQL on AWS) could significantly increase the scalability and reach of the system.

CHAPTER 5

APPENDIX A – SOURCE CODE

```
from flask import Flask, jsonify, request
from datetime import datetime
from flask import render_template
import sqlite3

app = Flask(__name__)

def init_db():
    conn = sqlite3.connect('wallet.db')
    c = conn.cursor()
    c.execute("""CREATE TABLE IF NOT EXISTS
transactions
(id INTEGER PRIMARY KEY
AUTOINCREMENT,
amount REAL,
description TEXT,
type TEXT,
date TEXT)""")
    c.execute("""CREATE TABLE IF NOT EXISTS
balance
(id INTEGER PRIMARY KEY,
amount REAL)""")
    c.execute("""CREATE TABLE IF NOT EXISTS
digital_ids
(id INTEGER PRIMARY KEY
AUTOINCREMENT,
```

```

        type TEXT,
        number TEXT)'''

    c.execute('INSERT OR IGNORE INTO balance (id,
amount) VALUES (1, 0.0)')

    conn.commit()

    conn.close()


@app.route('/')
def index():

    return render_template('index.html')


@app.route('/balance', methods=['GET'])
def get_balance():

    conn = sqlite3.connect('wallet.db')

    c = conn.cursor()

    c.execute('SELECT amount FROM balance WHERE
id = 1')

    balance = c.fetchone()[0]

    conn.close()

    return jsonify({'balance': balance})


@app.route('/transactions', methods=['GET'])
def get_transactions():

    conn = sqlite3.connect('wallet.db')

    c = conn.cursor()

    c.execute('SELECT amount, description, type, date
FROM transactions ORDER BY date DESC')

    transactions = [{'amount': row[0], 'description':

```

```

row[1], 'type': row[2], 'date': row[3]} for row in c.fetchall()]

    conn.close()

    return jsonify(transactions)

```

```

@app.route('/digital_ids', methods=['GET'])
def get_digital_ids():
    conn = sqlite3.connect('wallet.db')
    c = conn.cursor()
    c.execute('SELECT type, number FROM digital_ids')
    ids = [{ 'type': row[0], 'number': row[1]} for row in
c.fetchall()]
    conn.close()
    return jsonify(ids)

```

```

@app.route('/add', methods=['POST'])
def add_funds():
    data = request.get_json()
    amount = data['amount']
    description = data['description']

    conn = sqlite3.connect('wallet.db')
    c = conn.cursor()
    c.execute('UPDATE balance SET amount = amount +
? WHERE id = 1', (amount,))

    c.execute('INSERT INTO transactions (amount,
description, type, date) VALUES (?, ?, ?, ?)',
            (amount, description, 'Credit',
datetime.now().strftime('%Y-%m-%d %H:%M:%S')))

```

```

conn.commit()

conn.close()

return jsonify({'status': 'success'})

@app.route('/spend', methods=['POST'])
def spend_funds():
    data = request.get_json()
    amount = data['amount']
    description = data['description']

    conn = sqlite3.connect('wallet.db')
    c = conn.cursor()
    # Check for digital ID
    c.execute('SELECT COUNT(*) FROM digital_ids')
    id_count = c.fetchone()[0]
    if id_count == 0:
        conn.close()
        return jsonify({'error': 'No digital ID found. Please
add a digital ID first.'}), 400

    # Check balance
    c.execute('SELECT amount FROM balance WHERE
id = 1')
    balance = c.fetchone()[0]

    if balance >= amount:
        c.execute('UPDATE balance SET amount =
amount - ? WHERE id = 1', (amount,))

```

```

        c.execute('INSERT INTO transactions (amount,
description, type, date) VALUES (?, ?, ?, ?)',
        (amount, description, 'Debit',
datetime.now().strftime('%Y-%m-%d %H:%M:%S')))

```

```

        conn.commit()

```

```

        conn.close()

```

```

        return jsonify({'status': 'success'})

```

```

    else:

```

```

        conn.close()

```

```

        return jsonify({'error': 'Insufficient funds'}), 400

```

```

@app.route('/send', methods=['POST'])

```

```

def send_funds():

```

```

    data = request.get_json()

```

```

    upi_id = data['upi_id']

```

```

    amount = data['amount']

```

```

    conn = sqlite3.connect('wallet.db')

```

```

    c = conn.cursor()

```

```

    # Check for digital ID

```

```

    c.execute('SELECT COUNT(*) FROM digital_ids')

```

```

    id_count = c.fetchone()[0]

```

```

    if id_count == 0:

```

```

        conn.close()

```

```

        return jsonify({'error': 'No digital ID found. Please
add a digital ID first.'}), 400

```

```

    # Check balance

```

```
c.execute('SELECT amount FROM balance WHERE  
id = 1')
```

```
balance = c.fetchone()[0]
```

```
if balance >= amount:
```

```
c.execute('UPDATE balance SET amount =  
amount - ? WHERE id = 1', (amount,))
```

```
c.execute('INSERT INTO transactions (amount,  
description, type, date) VALUES (?, ?, ?, ?)',
```

```
(amount, f'Sent to {upi_id}', 'Transfer',  
datetime.now().strftime('%Y-%m-%d %H:%M:%S'))
```

```
conn.commit()
```

```
conn.close()
```

```
return jsonify({'status': 'success'})
```

```
else:
```

```
conn.close()
```

```
return jsonify({'error': 'Insufficient funds'}), 400
```

```
@app.route('/add_id', methods=['POST'])
```

```
def add_digital_id():
```

```
data = request.get_json()
```

```
type = data['type']
```

```
number = data['number']
```

```
conn = sqlite3.connect('wallet.db')
```

```
c = conn.cursor()
```

```
c.execute('INSERT INTO digital_ids (type, number)  
VALUES (?, ?)', (type, number))
```

```
conn.commit()
conn.close()
return jsonify({'status': 'success'})
```

```
if __name__ == '__main__':
    init_db()
    app.run(debug=True)
```


APPENDIX B – SCREENSHOTS

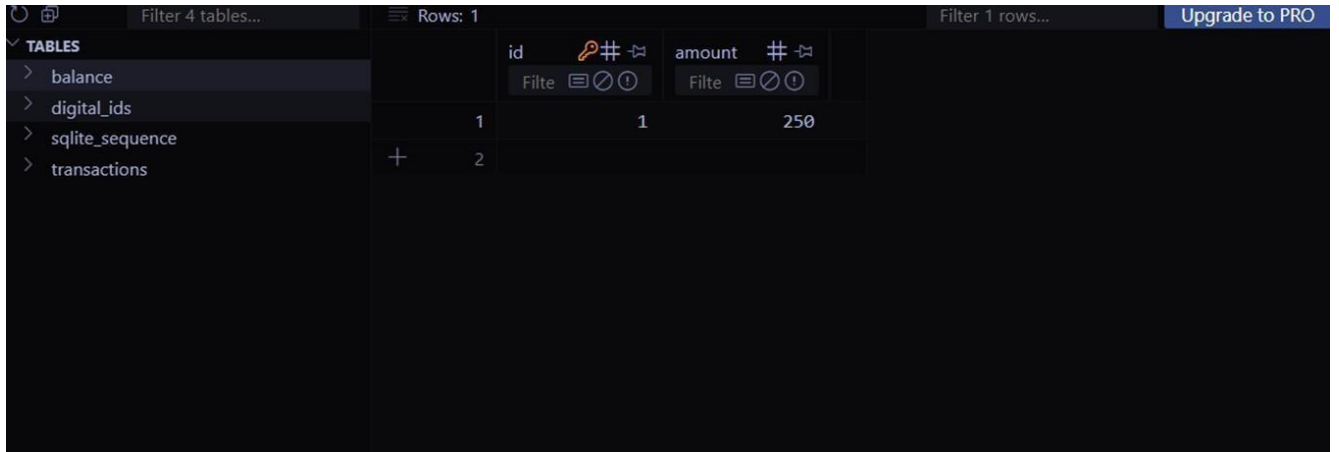
1. DIGITAL ID MODULE

The screenshot displays a 'Digital Wallet' interface. At the top, it shows the 'Current Balance' as '\$0.00'. Below this, the 'Digital IDs' section contains two input fields: 'ID Type' with a placeholder 'E.g., UPI' and 'ID Number' with a placeholder 'Enter ID number'. A blue 'Add Digital ID' button is positioned below these fields. Underneath the button, a grey box displays the UPI ID 'UPI: 1234123412341234'. The 'Manage Funds' section follows, featuring an 'Amount' input field with the placeholder 'Enter amount' and a 'Description' input field with the placeholder 'E.g., Coffee'.

2. TRANSACTION MANAGEMENT MODULE

The screenshot shows the 'Transaction Management' module. At the top, there are two buttons: 'Add Funds' (blue) and 'Spend' (red). Below these is the 'Send to Others' section, which includes a 'UPI ID' input field with the placeholder 'E.g., 1234 1234 1234 1234' and an 'Amount to Send' input field with the placeholder 'Enter amount'. A green 'Send Funds' button is located below the input fields. The 'Transaction History' section at the bottom lists two transactions: '2025-05-28 17:34:11 - Sent to 1234123412341234: \$100.00 (Transfer)' and '2025-05-28 17:33:54 - pocket money: \$100.00 (Credit)'.

3. DATABASE MODULE



The screenshot shows a database application interface. On the left, a sidebar lists tables: balance, digital_ids, sqlite_sequence, and transactions. The main area displays a table with two columns: 'id' and 'amount'. The 'id' column has a value of 1, and the 'amount' column has a value of 250. The interface includes a top bar with a search filter, a 'Rows: 1' indicator, and an 'Upgrade to PRO' button.

TABLES	id	amount
> balance	1	250
> digital_ids		
> sqlite_sequence		
> transactions		

REFERENCE

1. Flask Documentation – *Flask: Web Development One Drop at a Time*, Pallets Projects
<https://flask.palletsprojects.com/>
2. SQLite Documentation – *SQLite: Lightweight Relational Database Engine*
<https://www.sqlite.org/docs.html>
3. Jinja2 Templating – *Official Jinja2 Template Engine for Python*
<https://jinja.palletsprojects.com/>
4. Python Official Documentation – *Python 3 Standard Library Reference*
<https://docs.python.org/3/>