SmartServe: Integrated Resident Information and Public Service Management System

In partial fulfillment of the requirements for the course Database Management System

Submitted by:

Oli, John Clarence M.

Dominguez, Alfrendo B.

Garcia, Alejandro Jr. A.

Orias, John Gilbert R.

Vargas, Angle Mia L.

BSIT-2101

December 14, 2025

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

1.1 Project Context

In the current landscape of local governance, many communities still rely on traditional, paper-based systems to manage resident records and process service requests. This manual approach often leads to data redundancy, inefficient transaction processing, and the risk of document loss or damage. As the population grows, the administrative burden on barangay officials increases, necessitating a more organized and secure data management system.

The SmartServe is a web-based Resident Information and Public Service Management System developed to address these operational inefficiencies. By leveraging modern web technologies and a structured relational database, the system transitions manual logbooks into a centralized digital platform. This project serves as a practical application of Modular Programming and Database Management, demonstrating how information technology can streamline public service delivery, ensure data integrity, and improve the overall experience for both residents and administrators.

1.2 Project Objectives

The general objective of this study is to develop a functional web-based information system that digitizes the document request process and centralizes resident record management.

Specifically, the project aims to:

- Design a secure resident module that allows users to register accounts, manage profiles, and submit document requests (e.g., Barangay Clearance, Certificate of Indigency) remotely.
- Develop a normalized relational database (compliant with 3rd Normal Form) to store user credentials and transaction history, minimizing data redundancy and anomalies.
- Implement an administrator dashboard for real-time monitoring, approval, and rejection of resident requests.
- Apply modular programming principles using Python and Flask to ensure code maintainability and system scalability.

1.3 Significance of the Study

This project aligns with the United Nations' Sustainable Development Goals (SDGs), highlighting the role of technology in community development.

- **SDG 9 (Industry, Innovation, and Infrastructure):** The system introduces digital innovation to local governance infrastructure, moving away from outdated manual processes towards efficient, automated systems.
- **SDG 11 (Sustainable Cities and Communities):** By providing an accessible online platform for essential public services, the system promotes inclusive and sustainable community management, reducing administrative barriers for residents.

1.4 Scope and Limitations

Scope of the Study

The project focuses on developing a comprehensive digital platform that manages the lifecycle of document requests, from user registration to administrative processing. The scope is divided into three core functional modules:

Resident Module (User Management & Request)

The system facilitates secure User Authentication, allowing residents to create accounts by providing necessary personal details (name, address, contact information) and securely logging in to access services. It includes Profile Management, permitting users to view and update their records. The core feature, Document Submission, enables authenticated users to formally request documents (e.g., Barangay Clearance, Certificate of Indigency) through dynamic forms. Additionally, a Status Tracking System is integrated, allowing residents to monitor the real-time progress of their requests (Pending, Approved, Rejected) directly via their dashboard.

Admin Module (System Management)

The system provides administrators with a centralized Dashboard for request oversight, allowing them to view, sort, and manage the queue of submitted applications. It features Request Processing, which grants admins the authority to validate requests, update their status, and provide remarks or notes for the resident. The module also includes User Management, giving administrators the ability to view the list of registered community members. Furthermore, basic Reporting and Analytics are implemented to visualize the total number of residents and the distribution of requests based on their status.

Limitations of the Study

While the system provides a functional framework for community service management, it has specific limitations. The Notification System is limited to on-site dashboard updates; the current iteration does not support external alerts via SMS or email. The system also does not include an Online Payment Gateway; thus, any fees associated with document processing must be settled physically at the barangay hall. Lastly, the system is designed for a single-tier administrative role and does not currently support complex role-based task assignments for multiple staff members.

CHAPTER 2: METHODOLOGY

2.1 System Architecture

The SmartServe system was developed using a Modular Architecture to ensure code maintainability, scalability, and separation of concerns. This approach aligns with the principles of advanced computer programming by organizing the codebase into distinct functional components rather than a monolithic structure.

The backend logic is powered by Python using the Flask web framework. Flask handles the HTTP requests, URL routing, and session management, acting as the controller that bridges the user interface and the database. The frontend utilizes HTML5, CSS3, and Jinja2 templating engine to render dynamic content responsive to user interactions. For data persistence, the system integrates SQLite, a serverless relational database engine, which is ideal for the system's current local deployment scope.

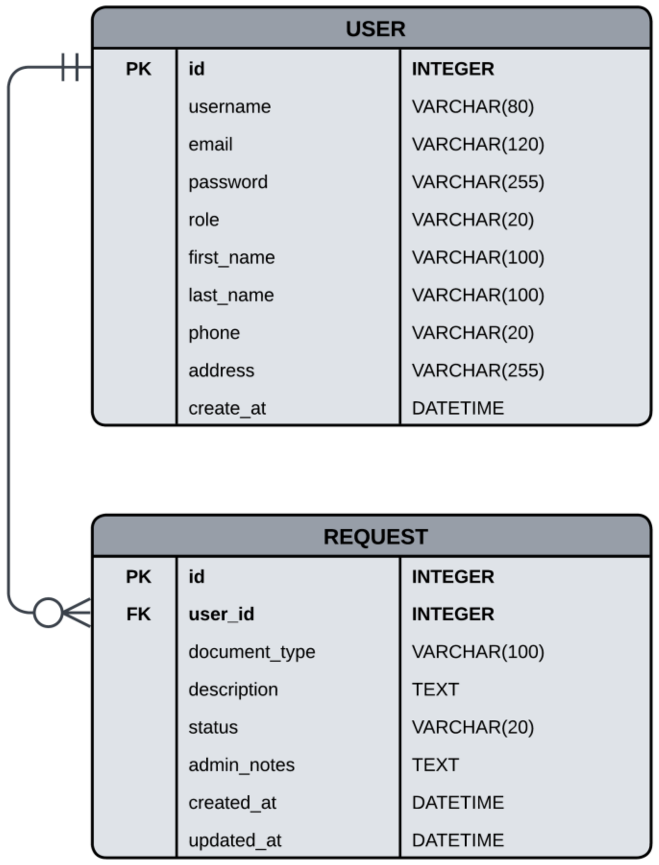
2.2 Database Design

The database structure is designed to uphold data integrity and support efficient query performance. An Entity-Relationship Diagram (ERD) was created to visualize the logical structure and relationships between the system's core entities.

The system utilizes two primary entities:

- **User Entity:** Stores the personal information and login credentials of residents and administrators.
- **Request Entity:** Stores the details of document transactions.

The relationship between these entities is defined as One-to-Many, where a single registered user can submit multiple document requests, but each specific request is linked to only one user. This relationship is enforced using a Foreign Key (user_id) in the Request table that references the Primary Key (id) of the User table.



USER		
PK	id	INTEGER
	username	VARCHAR(80)
	email	VARCHAR(120)
	password	VARCHAR(255)
	role	VARCHAR(20)
	first_name	VARCHAR(100)
	last_name	VARCHAR(100)
	phone	VARCHAR(20)
	address	VARCHAR(255)
	create_at	DATETIME

REQUEST		
PK	id	INTEGER
FK	user_id	INTEGER
	document_type	VARCHAR(100)
	description	TEXT
	status	VARCHAR(20)
	admin_notes	TEXT
	created_at	DATETIME
	updated_at	DATETIME

Figure 2.0: Entity-Relationship Diagram of SmartServe System

2.3 Data Normalization Process

To strictly adhere to relational database standards, the schema underwent a normalization process to organize data, minimize redundancy, and prevent data anomalies.

2.3.1 Unnormalized Form (UNF)

Initially, the data was conceptualized as a flat-file structure (similar to a spreadsheet) where resident profiles and their transaction history were combined in a single table. This resulted in significant data redundancy, as a resident's personal information was repeated for every new document request they submitted.

Table 2.1: Unnormalized Data Representation

Resident Name	Address	Document Type	Status
Juan Dela Cruz	Purok 1, Batangas City	Barangay Clearance	Pending
Juan Dela Cruz	Purok 1, Batangas City	Certificate of Indigency	Approved
Maria Clara	Purok 2, Batangas City	Certificate of Residency	Pending

As observed in Table 2.1, the name "Juan Dela Cruz" and address "Purok 1, Batangas City" are duplicated for every transaction, leading to data inconsistency risks.

2.3.2 First Normal Form (1NF)

To satisfy the First Normal Form (1NF), we enforced the rule of Atomicity. We eliminated repeating groups and ensured that each column contained only one specific, indivisible value. Specifically, the "Name" field was decomposed into first_name and last_name to allow for better sorting and querying.

Table 2.2: First Normal Form (1NF)

first_name	last_name	address	document_type	status
Juan	Dela Cruz	Purok 1, Batangas City	Barangay Clearance	Pending
Juan	Dela Cruz	Purok 1, Batangas City	Certificate of Indigency	Approved
Maria	Clara	Batangas City	Certificate of Residency	Pending

In Table 2.2, standard column naming conventions were applied, and compound values were broken down. However, redundancy in the address field still exists, requiring further normalization.

2.3.3 Second Normal Form (2NF)

The schema complies with 2NF by removing partial dependencies. We identified that transaction details (document_type, status) depend on the specific request, not just the resident. Thus, data was separated into two distinct tables: User (for profiles) and Request (for transactions).

Table 2.3: Resident Profile Group (Partial Dependency Removed)

first_name	last_name	address
Juan	Dela Cruz	Purok 1, Batangas City
Maria	Clara	Purok 2, Batangas City

As shown in Table 2.3, the resident's personal details are now isolated from the transaction data. This ensures that updates to a resident's address need only be made in one place.

2.3.4 Third Normal Form (3NF)

To achieve 3NF, we removed transitive dependencies. Attributes like role depend solely on the user_id, while status depends solely on the request_id. This final structure allows for efficient data management without anomalies.

Table 2.4: Normalized 3NF Schema (User Entity)

id (PK)	**username**	**first_name**	**last_name**	**address**
1	juan_delacruz	Juan	Dela Cruz	Purok 1, Batangas City
2	maria_clara	Maria	Clara	Purok 2, Batangas City

Table 2.5: Normalized 3NF Schema (Request Entity)

id (PK)	**user_id (FK)**	**document_type**	**status**
101	**1**	Barangay Clearance	Pending
102	**1**	Certificate of Residency	Approved
103	**2**	Certificate of Residency	Pending

As demonstrated in Tables 2.4 and 2.5, data redundancy is completely eliminated. The Request table now simply references the User table via the Foreign Key (user_id: 1), ensuring data integrity and optimizing storage.

CHAPTER 3: RESULTS

This chapter presents the outcome of the system development, including the data dictionary, user interface design, and the implementation of database queries.

3.1 Data Dictionary

The following tables describe the structure of the SmartServe database, detailing the field names, data types, and constraints used to ensure data integrity.

Table 3.1: User Entity (User)

Stores the personal information and credentials of residents and administrators.

Field Name	Data Type	Constraint / Key	Description
id	INTEGER	PRIMARY KEY, AUTOINCREMENT	Unique identifier for each user
username	VARCHAR(80)	NOT NULL, UNIQUE	User's login handle
email	VARCHAR(120)	NOT NULL, UNIQUE	Official email address for contact
password	VARCHAR(255)	NOT NULL	Hashed security credential
role	VARCHAR(20)	DEFAULT 'resident'	Access level ('admin' or 'resident')
first_name	VARCHAR(100)	NULLABLE	Resident's given name
last_name	VARCHAR(100)	NULLABLE	Resident's surname
phone	VARCHAR(20)	NULLABLE	Contact number
address	VARCHAR(255)	NULLABLE	Physical residency address
created_at	DATETIME	DEFAULT CURRENT_TIMESTAMP	Timestamp of account creation

Table 3.2: Request Entity (Request)

Stores the details of document transactions submitted by residents.

Field Name	Data Type	Constraint / Key	Description
id	INTEGER	PRIMARY KEY, AUTOINCREMENT	Unique identifier for the request
user_id	INTEGER	FOREIGN KEY (References user.id)	Links request to the requester
document_type	VARCHAR(100)	NOT NULL	Type of document (e.g. Barangay Clearance)
description	TEXT	NULLABLE	Purpose of request (e.g. for Scholarship)
status	VARCHAR(20)	DEFAULT 'pending'	Current state (Pending, Approved, etc.)
admin_notes	TEXT	NULLABLE	Remarks from the administrator
created_at	DATETIME	DEFAULT CURRENT_TIMESTAMP	Date of submission
updated_at	DATETIME	ON UPDATE CURRENT_TIMESTAMP	Date of last status change

3.2 System User Interface

This section provides a comprehensive visualization of the SmartServe System's Graphical User Interface (GUI), highlighting the application's frontend architecture. The following figures illustrate the sequential user interaction flow, beginning with the introductory Home Page and proceeding through the secure Registration and Authentication gateways. Furthermore, it showcases the system's role-based access control by distinguishing between the Resident and Administrator Dashboards, each tailored with specific functionalities to ensure efficient service delivery.

3.2.1 Public Access

Upon accessing the web application, users are greeted with the home page which provides an overview of the system's services and core features.

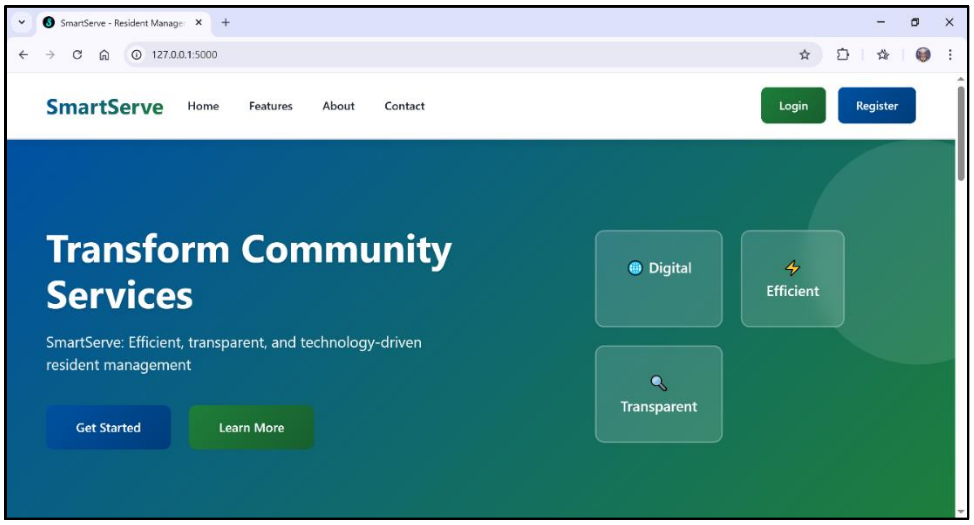
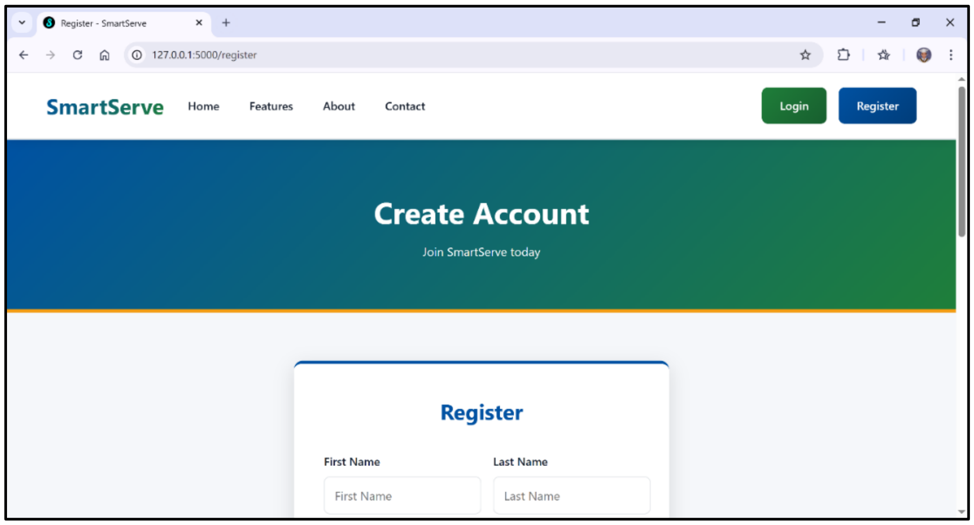

SmartServe - Resident Manage
127.0.0.1:5000
SmartServe
Home
Features
About
Contact
Login
Register
Transform Community Services
SmartServe: Efficient, transparent, and technology-driven resident management
Get Started
Learn More
Digital
Efficient
Transparent

Figure 3.1: SmartServe Home Page

3.2.2 Registration Module

New residents can create an account via the registration form. The system captures essential resident data and validates inputs before submission.


Register - SmartServe

127.0.0.1:5000/register

Create Account

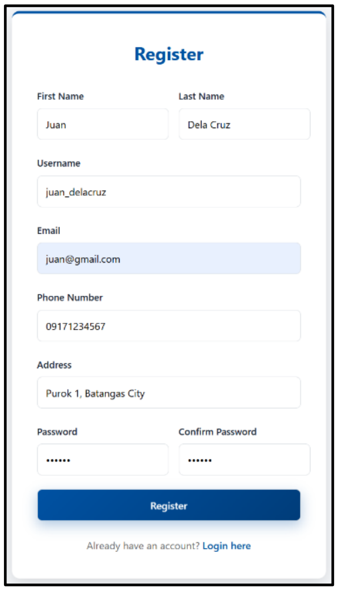
Join SmartServe today

Register

First Name: First Name

Last Name: Last Name

Figure 3.2: Registration Form


Register

First Name: Juan

Last Name: Dela Cruz

Username: juan_delacruz

Email: juan@gmail.com

Phone Number: 09171234567

Address: Purok 1, Batangas City

Password: ••••••

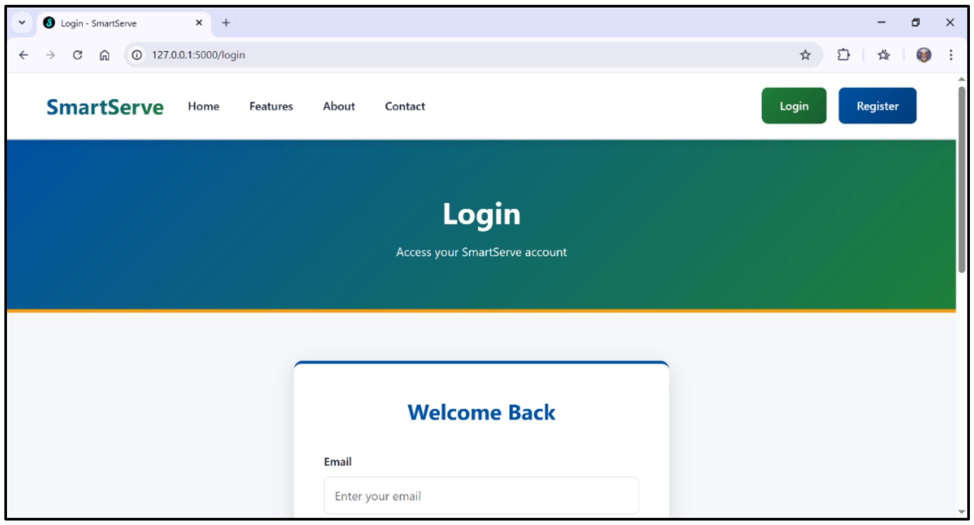
Confirm Password: ••••••

Already have an account? Login here

Figure 3.3: Registration Form with Sample Resident Data

3.2.3 Authentication Module

The system provides a secure login gateway. Users must input their registered credentials, and the system directs them to the appropriate interface based on their role (Resident or Administrator).



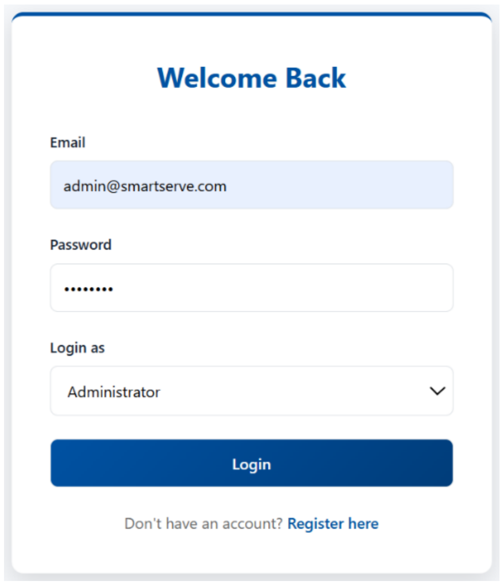
Login

Access your SmartServe account

Welcome Back

Email: Enter your email

Figure 3.4: Main Login Interface

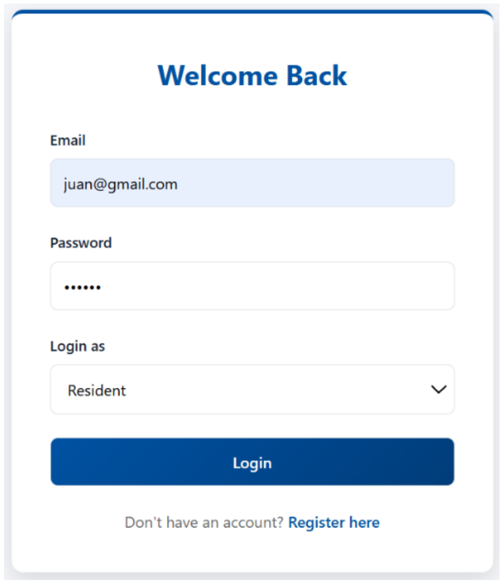

Welcome Back

Email: admin@smartserve.com

Password: ••••••••

Login as: Administrator

Figure 3.5: Login Input for Administrator Account



Welcome Back

Email

juan@gmail.com

Password

••••••

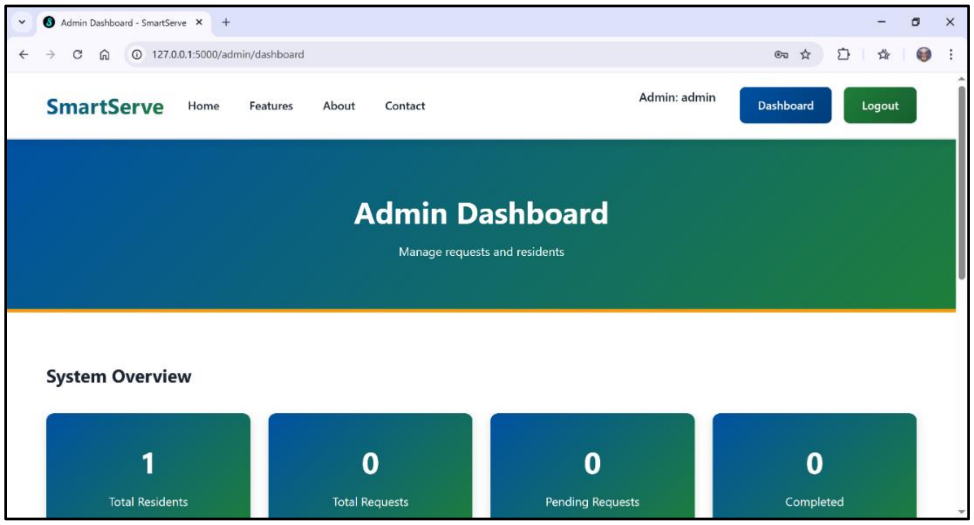
Login as

Resident

Figure 3.6: Login Input for Resident Account

3.2.4 System Dashboards

Upon successful authentication, users are redirected to their respective dashboards. This routing logic segregates resident services from administrative functions based on the user's role.



Admin Dashboard

Manage requests and residents

System Overview

1	0	0	0
Total Residents	Total Requests	Pending Requests	Completed

Figure 3.7: Admin Dashboard for Request Management and Resident Oversight

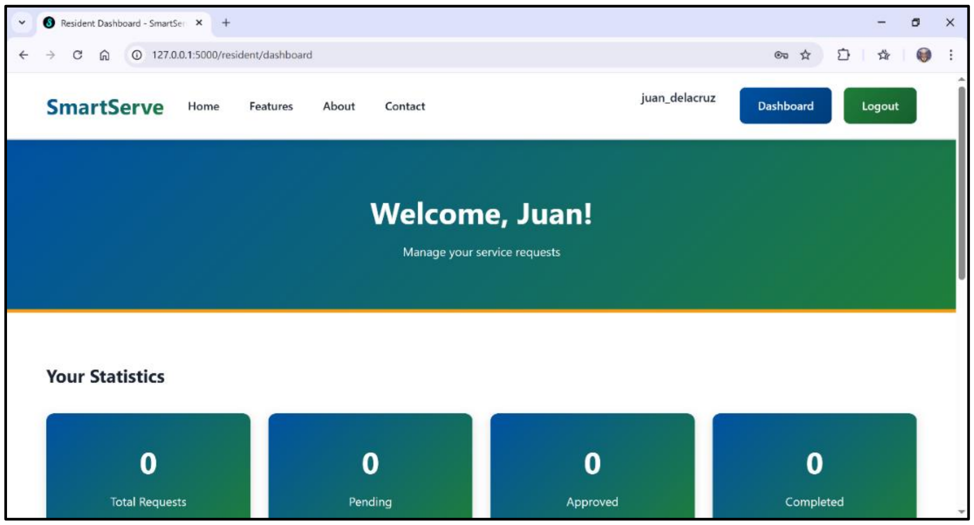


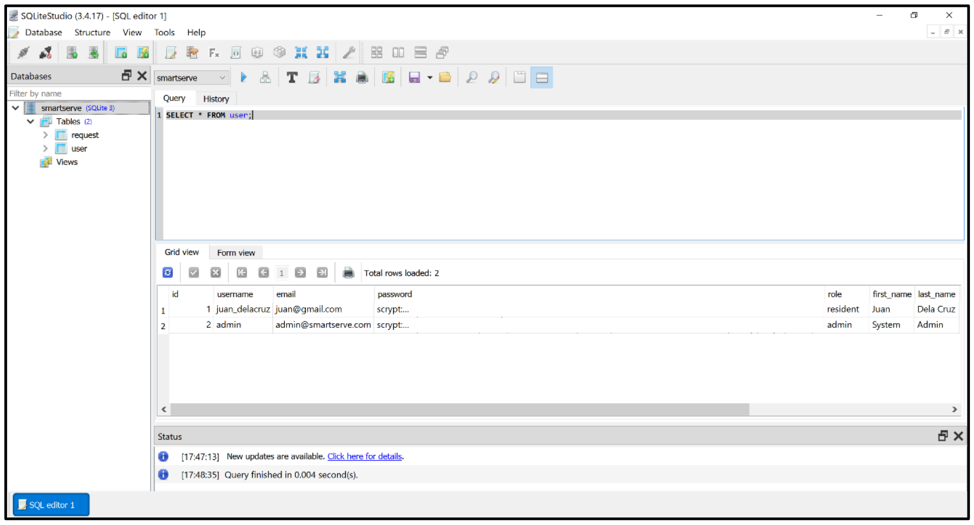
Figure 3.8: Resident Dashboard showing Status Tracking

3.3 SQL Query Implementation

This section provides a technical demonstration of the Structured Query Language (SQL) commands integrated into the system's backend, showcasing the precise execution of queries designed to manipulate, retrieve, and manage resident data while adhering to standard database management practices. To ensure the readability of table attributes and formatting, the figures presented in this section display a representative subset of the data; the complete dataset utilized for comprehensive system testing is provided in Appendix C.

3.3.1 Data Retrieval (SELECT)

This query retrieves the list of all registered users. In the application, this logic is applied in the Admin Dashboard to display the master list of residents.



```
SELECT * FROM user;
```

Total rows loaded: 2

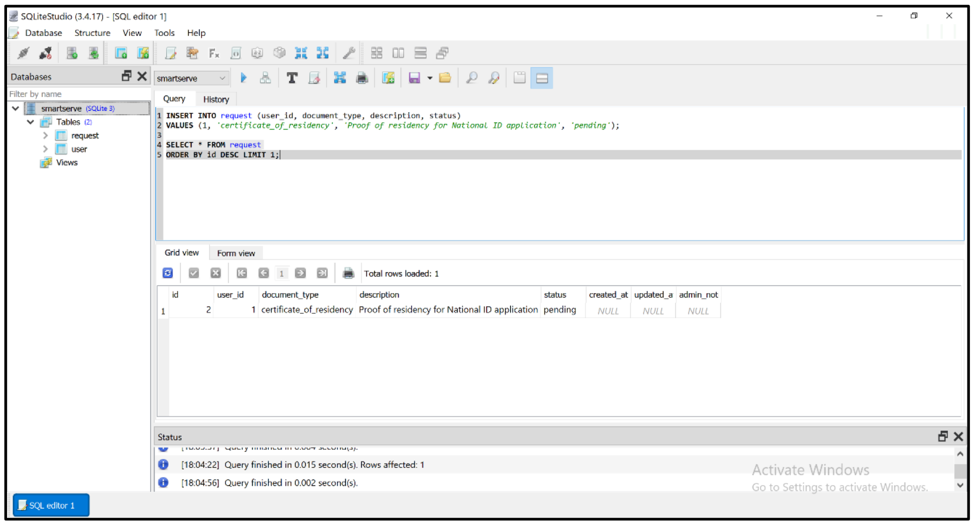
	id	username	email	password	role	first_name	last_name
1	1	juan_delacruz	juan@gmail.com	scrypt:...	resident	Juan	Dela Cruz
2	2	admin	admin@smartserve.com	scrypt:...	admin	System	Admin

```
[17:47:13] New updates are available. Click here for details.
[17:48:35] Query finished in 0.004 second(s).
```

Figure 3.5: SQL Execution displaying all registered users

3.3.2 Data Insertion (INSERT)

This command demonstrates adding a new record to the database. This logic is used in the Request Form, where a resident's submission is saved into the request table.



```sql
INSERT INTO request (user_id, document_type, description, status)
VALUES (1, 'certificate_of_residency', 'Proof of residency for National ID application', 'pending');

SELECT * FROM request
ORDER BY id DESC LIMIT 1;
```

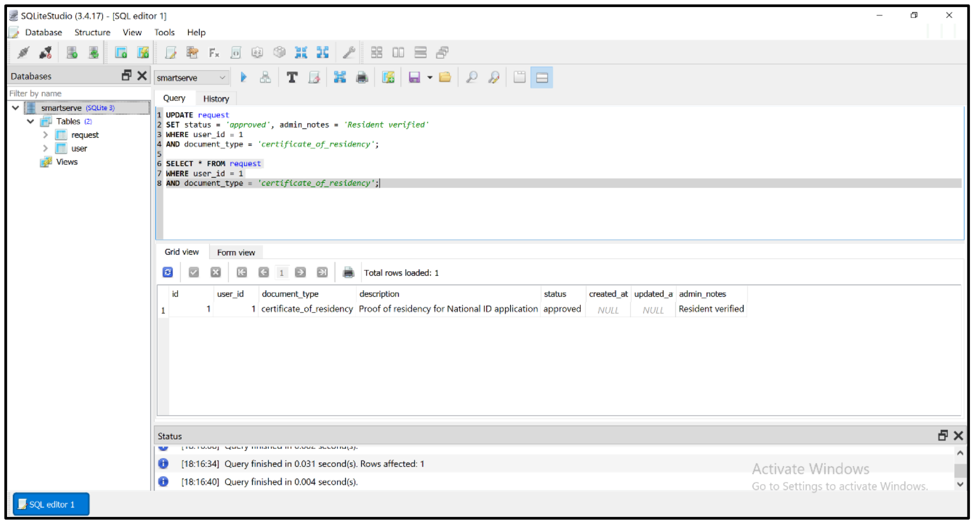
```
Total rows loaded: 1

   id  user_id  document_type             description                                      status   created_at  updated_a  admin_not
1   2        1  certificate_of_residency  Proof of residency for National ID application  pending  NULL        NULL       NULL
```

```
[18:04:22] Query finished in 0.015 second(s). Rows affected: 1
[18:04:56] Query finished in 0.002 second(s).
```

Figure 3.6: SQL Execution adding a new document request

3.3.3 Data Modification (UPDATE)

This command updates an existing record. This is applied in the Admin Module when an official changes a request status from 'Pending' to 'Approved' or 'Rejected'.



```sql
UPDATE request
SET status = 'approved', admin_notes = 'Resident verified'
WHERE user_id = 1
AND document_type = 'certificate_of_residency';

SELECT * FROM request
WHERE user_id = 1
AND document_type = 'certificate_of_residency';
```

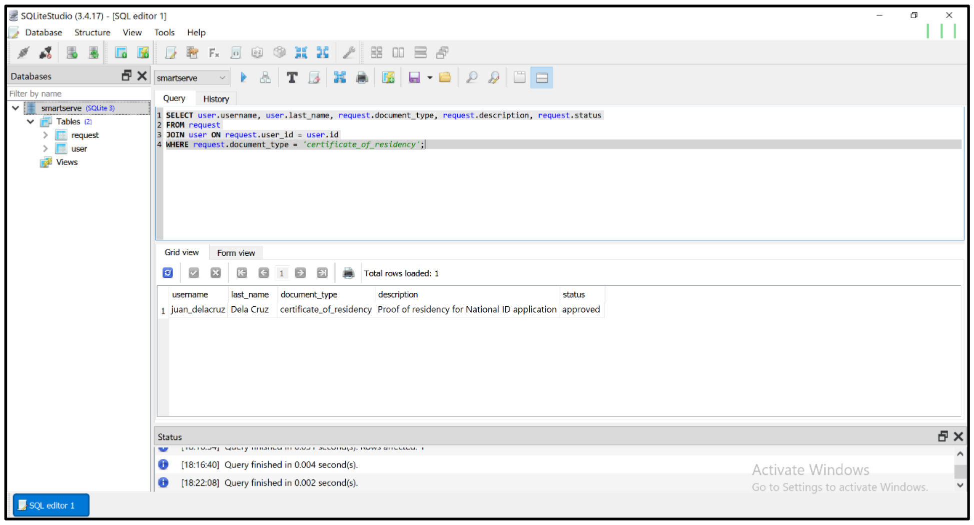
```
Total rows loaded: 1

   id  user_id  document_type             description                                      status    created_at  updated_a  admin_notes
1   1        1  certificate_of_residency  Proof of residency for National ID application  approved  NULL        NULL       Resident verified
```

```
[18:16:34] Query finished in 0.031 second(s). Rows affected: 1
[18:16:40] Query finished in 0.004 second(s).
```

Figure 3.7: SQL Execution updating request status

3.3.4 Relational Query (JOIN)

This query combines data from the user and request tables. It is essential for the Admin View to identify which resident submitted a specific request by linking the user_id Foreign Key.



```sql
SELECT user.username, user.last_name, request.document_type, request.description, request.status
FROM request
JOIN user ON request.user_id = user.id
WHERE request.document_type = 'certificate_of_residency';
```

Total rows loaded: 1

	username	last_name	document_type	description	status
1	juan_delacruz	Dela Cruz	certificate_of_residency	Proof of residency for National ID application	approved

```
[18:16:40] Query finished in 0.004 second(s).
[18:22:08] Query finished in 0.002 second(s).
```

Activate Windows
Go to Settings to activate Windows.

Figure 3.8: SQL JOIN operation linking Residents to their Requests

CHAPTER 4: DISCUSSION

4.1 Summary of Findings

The development and testing phase confirmed that the SmartServe System functions as intended. All core modules—specifically User Registration, Authentication, and Document Request Processing—performed reliably during the simulation. The system successfully delivered a streamlined workflow, allowing residents to submit requests remotely and enabling barangay staff to manage these transactions efficiently. The database also functioned correctly, validating the "One-to-Many" relationship between residents and their requests without data anomalies. Overall, the system met the functional requirements set in the objectives, demonstrating readiness for potential real-world application.

4.2 Challenges Encountered

During the development process, the team encountered specific technical hurdles:

- **Database Connectivity:** One of the primary challenges was establishing a stable connection between the Flask application and the SQLite database. Initial issues involved configuration errors in the Object-Relational Mapping (ORM) and inconsistencies in data handling. These were resolved through careful debugging of the database schema and adjustment of the connection settings.
- **Interface Integration:** Minor hurdles arose during the integration of the backend logic with the frontend user interface. Ensuring that dynamic data (such as status updates) reflected accurately on the user dashboard required precise coding in the HTML templates. These issues were addressed through iterative testing and code refinement.

CHAPTER 5: CONCLUSION AND RECOMMENDATIONS

5.1 Conclusion

Based on the results of the development and testing phases, the proponents conclude that the SmartServe System was successfully developed and achieved its intended objectives. The project provides a functional, user-friendly, and secure platform that streamlines the request and processing of barangay documents. The implementation of a normalized relational database (3NF) ensured data integrity, while the modular programming approach facilitated a stable system architecture. All core features operate effectively, demonstrating that the system is ready for deployment and actual use by residents and barangay staff to improve local governance efficiency.

5.2 Recommendations

While the current system satisfies the core requirements of the study, the proponents recommend the following enhancements for future iterations:

- **Integration of SMS Notification Feature:** As suggested by the limitations of the current study, future developers should integrate an automated SMS system. This would provide faster and more accessible updates to residents regarding the status of their documents, especially for users without regular internet access.
- **Development of a Mobile Application Version:** Transitioning the web-based system into a dedicated mobile application (Android/iOS) is recommended. This would offer a more convenient and portable way for residents to submit requests, track documents, and receive push notifications directly on their smartphones.
- **Implementation of Online Payment Gateway:** To fully digitize the transaction process, integrating e-wallet solutions (such as GCash or Maya) is recommended. This would allow residents to settle document fees remotely, further reducing the need for physical presence at the barangay hall.
- **Cloud Hosting Deployment:** To expand accessibility beyond the local network, deploying the system to a cloud hosting provider is recommended to ensure 24/7 availability for the community.

REFERENCES

Connolly, T., & Begg, C. (2014). *Database Systems: A Practical Approach to Design, Implementation, and Management* (6th ed.). Pearson Education.

Grinberg, M. (2018). *Flask Web Development: Developing Web Applications with Python*. O'Reilly Media.

Pallets Projects. (2023). *Flask Documentation (3.0.x)*. Retrieved December 10, 2025, from https://flask.palletsprojects.com/

Python Software Foundation. (2023). *Python 3.12.1 Documentation*. Retrieved December 10, 2025, from https://docs.python.org/3/

SQLite Consortium. (2023). *SQLite Database Engine: Architecture and Design*. Retrieved December 10, 2025, from https://www.sqlite.org/

United Nations. (2015). *The 17 Goals | Sustainable Development*. United Nations Department of Economic and Social Affairs. Retrieved December 10, 2025, from https://sdgs.un.org/goals

W3Schools. (2023). *SQL Tutorial and Database Normalization*. Retrieved December 10, 2025, from https://www.w3schools.com/sql/

APPENDICES

Appendix A: User Guide

A.1 How to Register an Account

1. **Access the System:** Open the SmartServe website on your browser.
2. **Navigate to Registration:** On the login page, click the link labeled **"Register here"**.
3. **Fill out Information:** Enter the required details in the registration form:

   - First Name and Last Name
   - Username and Email Address
   - Phone Number and Home Address
   - Password (and Confirm Password)

4. **Submit:** Click the **"Register"** button. The system will redirect you to the login page upon success.

A.2 How to Log In

1. **Enter Credentials:** On the homepage, type your registered **Email Address** and **Password**.
2. **Access Dashboard:** Click the **"Login"** button. You will be directed to the Resident Dashboard.

A.3 How to Request a Document

1. **Select a Document:** On your dashboard, navigate to the **"Request Document"** section. Choose the type of document you need (e.g., Barangay Clearance, Certificate of Indigency) from the dropdown menu.
2. **Provide Details:** In the purpose field, explain the reason for your request (e.g., "For Employment," "For Scholarship").
3. **Submit Request:** Click the **"Submit Request"** button. Your transaction will be added to the queue.

A.4 How to Check Status

1. **Monitor Dashboard:** After submitting, you will be redirected to your dashboard. Look at the **"My Requests"** table.
2. **Check Status Column:**

- **Pending:** The admin is still reviewing your request.
- **Approved:** Your document is ready. Please proceed to the barangay hall for pickup.
- **Rejected:** Your request was declined. Check the "Admin Notes" column for the reason.

Appendix B: Database Schema Scripts

This SQL script defines the relational structure of the SmartServe database, establishing the entities for users and requests, along with their relationships and integrity constraints.

File Name: database/schema_v1.sql

```sql
-- schema_v1.sql
-- Project: SmartServe
-- Description: Initial database schema (3NF Compliant)

-- Enable Foreign Key support for SQLite
PRAGMA foreign_keys = ON;

-- 1. USERS TABLE (Stores Residents and Admins)
-- Stores login credentials and profile information
CREATE TABLE user (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    username VARCHAR(80) NOT NULL UNIQUE,
    email VARCHAR(120) NOT NULL UNIQUE,
    password VARCHAR(255) NOT NULL,
    role VARCHAR(20) DEFAULT 'resident', -- Values: 'resident' or 'admin'
    first_name VARCHAR(100),
    last_name VARCHAR(100),
    phone VARCHAR(20),
    address VARCHAR(255),
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

```sql
-- 2. REQUESTS TABLE (Transactions)
-- Tracks document requests linked to a specific user
CREATE TABLE request (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    user_id INTEGER NOT NULL,
    document_type VARCHAR(100) NOT NULL,
    description TEXT,
    status VARCHAR(20) DEFAULT 'pending', -- Values: 'pending', 'approved',
'rejected', 'completed'
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    updated_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    admin_notes TEXT,
    FOREIGN KEY (user_id) REFERENCES user(id) ON DELETE CASCADE
);
```

Appendix C: Seed Data Script

This SQL script provides the initial dataset for the SmartServe database, utilizing INSERT statements to populate the system with twenty (20) sample user records and corresponding service requests. This data ensures a populated environment to facilitate comprehensive system testing, functionality validation, and performance assessment.

File Name: database/seed.sql

```sql
-- Project: SmartServe (Sample Data)
-- Description: Seed Data for Barangay Document Management System (Testing
Purposes)

-- ==========================================
-- 1. SAMPLE RESIDENTS
-- Passwords are placeholders for testing
-- ==========================================
INSERT INTO user (username, email, password, role, first_name, last_name, phone,
address) VALUES
('juan_delacruz', 'juan@gmail.com', 'scrypt:32768:8:1$DummyHash...', 'resident',
'Juan', 'Dela Cruz', '09171234567', 'Purok 1, Batangas City'),
('maria_clara', 'maria@gmail.com', 'scrypt:32768:8:1$DummyHash...', 'resident',
'Maria', 'Clara', '09171234568', 'Purok 2, Batangas City'),
('jose_rizal', 'jose@gmail.com', 'scrypt:32768:8:1$DummyHash...', 'resident',
'Jose', 'Rizal', '09171234569', 'Purok 3, Batangas City'),
('andres_boni', 'andres@gmail.com', 'scrypt:32768:8:1$DummyHash...', 'resident',
'Andres', 'Bonifacio', '09171234570', 'Purok 4, Batangas City'),
('emilio_aguinaldo', 'emilio@gmail.com', 'scrypt:32768:8:1$DummyHash...',
'resident', 'Emilio', 'Aguinaldo', '09171234571', 'Purok 5, Batangas City'),
('apolinario_mabini', 'pol@gmail.com', 'scrypt:32768:8:1$DummyHash...',
'resident', 'Apolinario', 'Mabini', '09171234572', 'Purok 6, Batangas City'),
('antonio_luna', 'luna@gmail.com', 'scrypt:32768:8:1$DummyHash...', 'resident',
'Antonio', 'Luna', '09171234573', 'Purok 1, Batangas City'),
('gregorio_delpilar', 'goyo@gmail.com', 'scrypt:32768:8:1$DummyHash...',
'resident', 'Gregorio', 'Del Pilar', '09171234574', 'Purok 2, Batangas City'),
('gabriela_silang', 'gabriela@gmail.com', 'scrypt:32768:8:1$DummyHash...',
'resident', 'Gabriela', 'Silang', '09171234575', 'Purok 3, Batangas City'),
('melchora_aquino', 'tandang_sora@gmail.com', 'scrypt:32768:8:1$DummyHash...',
'resident', 'Melchora', 'Aquino', '09171234576', 'Purok 4, Batangas City'),
('lapu_lapu', 'lapu@gmail.com', 'scrypt:32768:8:1$DummyHash...', 'resident',
'Lapu', 'Lapu', '09171234577', 'Purok 5, Batangas City'),
('francisco_balagtas', 'kiko@gmail.com', 'scrypt:32768:8:1$DummyHash...',
'resident', 'Francisco', 'Balagtas', '09171234578', 'Purok 6, Batangas City'),
('marcelo_delpilar', 'marcelo@gmail.com', 'scrypt:32768:8:1$DummyHash...',
'resident', 'Marcelo', 'Del Pilar', '09171234579', 'Purok 1, Batangas City'),
('graciano_lopez', 'graciano@gmail.com', 'scrypt:32768:8:1$DummyHash...',
'resident', 'Graciano', 'Lopez Jaena', '09171234580', 'Purok 2, Batangas City'),
('juan_luna', 'juanluna@gmail.com', 'scrypt:32768:8:1$DummyHash...', 'resident',
'Juan', 'Luna', '09171234581', 'Purok 3, Batangas City'),
('mariano_gomez', 'gomez@gmail.com', 'scrypt:32768:8:1$DummyHash...', 'resident',
'Mariano', 'Gomez', '09171234582', 'Purok 4, Batangas City'),
('jose_burgos', 'burgos@gmail.com', 'scrypt:32768:8:1$DummyHash...', 'resident',
'Jose', 'Burgos', '09171234583', 'Purok 5, Batangas City'),
('jacinto_zamora', 'zamora@gmail.com', 'scrypt:32768:8:1$DummyHash...',
'resident', 'Jacinto', 'Zamora', '09171234584', 'Purok 6, Batangas City'),
('emilio_jacinto', 'dimasalang@gmail.com', 'scrypt:32768:8:1$DummyHash...',
'resident', 'Emilio', 'Jacinto', '09171234585', 'Purok 1, Batangas City'),
('miguel_malvar', 'malvar@gmail.com', 'scrypt:32768:8:1$DummyHash...',
'resident', 'Miguel', 'Malvar', '09171234586', 'Purok 2, Batangas City');

-- ==========================================
-- 2. SAMPLE REQUESTS
-- Linked to User IDs above
-- ==========================================
INSERT INTO request (user_id, document_type, description, status) VALUES
(1, 'barangay_clearance', 'For employment purposes', 'pending'),
(2, 'residency_cert', 'Proof of residency for bank', 'approved'),
(3, 'indigency_cert', 'Scholarship application', 'pending'),
(4, 'barangay_clearance', 'Business permit renewal', 'completed'),
(5, 'business_permit', 'Sari-sari store application', 'rejected'),
(1, 'residency_cert', 'Voters registration', 'approved'),
(2, 'barangay_clearance', 'Job application requirement', 'pending'),
(3, 'indigency_cert', 'Medical assistance', 'completed'),
(6, 'barangay_clearance', 'New work requirement', 'pending'),
(7, 'residency_cert', 'Postal ID application', 'approved'),
(8, 'indigency_cert', 'School requirement', 'pending'),
(9, 'barangay_clearance', 'Travel requirement', 'completed'),
(10, 'business_permit', 'Small eatery permit', 'rejected'),
(11, 'barangay_clearance', 'Local employment', 'approved'),
(12, 'residency_cert', 'Bank account opening', 'pending'),
(13, 'indigency_cert', 'Financial aid request', 'completed'),
(14, 'barangay_clearance', 'Work permit', 'pending'),
(15, 'residency_cert', 'ID application', 'approved'),
(16, 'indigency_cert', 'School scholarship', 'pending'),
(17, 'barangay_clearance', 'Driver license requirement', 'completed');
```