APARTMENT MANAGEMENT SYSTEM

A MINI PROJECT REPORT

Submitted by

MOHAMMED IMRAN J 231801107

PADMAVATHY R 231801122

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

RAJALAKSHMI ENGINEERING COLLEGE,

ANNA UNIVERSITY: CHENNAI 600025

MAY 2024

ANNA UNIVERSITY: CHENNAI-600025

BONAFIDE CERTIFICATE

Certified that this project report “APARTMENT MANAGEMENT SYSTEM” is the bonafide work of “MOHAMMED IMRAN J (231801107), PADMAVATHY R (231801122)” who carried out the project work under my supervision.

SIGNATURE

Mrs. M. THAMIZHARASI,

SUPERVISOR AND

ASSISTANT PROFESSOR

Department of Artificial Intelligence and Data Science,

Rajalakshmi Engineering College

Thandalam, Chennai - 602 105

Submitted for Project Viva-Voce Examination held on\_\_\_\_\_\_\_\_\_\_\_.

INTERNAL EXAMINEREXTERNAL EXAMINER

**TABLE OF CONTENTS**

CHAPTER TITLE PAGE NO

ACKNOWLEDGEMENT ii

ABSTRACT iii

LIST OF FIGURES iv

LIST OF ABBREVIATIONS v

1. INTRODUCTION
   1. PROJECTION DEFINITION
      1. PROJECT OBJECTIVES
      2. KEY FEATURES
   2. NEED FOR PROPOSED SYSTEM
   3. APPLICATION OF THE PROPOSED SYSTEM
2. LITERATURE REVIEW
   1. EXISTING SYSTEMS AND THEIR LIMITATIONS
   2. ADVANCEMENTS IN WEB-BASED PROPERTY MANAGEMENT SYYSTEMS
   3. RELEVANT RESEARCH AND CASE STUDIES
   4. LIMITATIONS OF CURRENT LITERATURE
   5. POSITIONING OF AMS
3. PROJECT FORMULATION
   1. PLATFORM
      1. FRONTEND TECHNOLOGIES
      2. BACKEND TECHNOLOGIES
      3. DATABASE
      4. DEVELOPMENT TOOLS
      5. SERVER
   2. SPECIFIC OBJECTIVES
   3. METHODOLOGY
      1. REQUIREMENT ANALYSIS
      2. SYSTEM DESIGN
      3. IMPLEMENTATION
      4. TESTING
      5. DEPLOYMENT
      6. MAINTENANCE
4. SYSTEM ANALYSIS AND DESIGN
   1. SYSTEM ANALYSIS
      1. EXISTING SYSTEM
      2. PROPOSED SYSTEM
   2. DATABASE DESIGN
      1. RESIDENTS TABLE
      2. APARTMENTS TABLE
   3. SYSTEM ARCHITECTURE
   4. UML DIAGRAMS
      1. USE CASE DIAGRAM
      2. ENTITY-RELATIONSHIP(ER) DIAGRAM
      3. SEQUENCE DIAGRAM
      4. CLASS DIAGRAM
5. FUNCTIONAL DESCRIPTION
   1. ADD RESIDENT
      1. PURPOSE
      2. IMPLEMENTATION
      3. WORKFLOW
      4. ERROR HANDLING
   2. VIEW RESIDENTS
      1. PURPOSE
      2. IMPLEMENTATION
      3. WORKFLOW
      4. FEATURES
      5. ERROR HANDLING
   3. ADD APARTMENT
      1. PURPOSE
      2. IMPLEMENTATION
      3. WORKFLOW
      4. ERROR HANDLING
   4. VIEW APARTMENTS
      1. PURPOSE
      2. IMPLEMENTATION
      3. WORKFLOW
      4. FEATURES
      5. ERROR HANDLING
   5. SUMMARY OF FUNCTIONS
6. IMPLEMENTATION AND TESTING
   1. IMPLEMENTATION
      1. FRONTEND IMPLEMENTATION
      2. BACKEND IMPLEMENTATION
      3. DATABASE IMPLEMENTATION
      4. DEPLOYMENT
   2. TESTING
      1. UNIT TESTING
      2. INTEGRATION TESTING
      3. USER TESTING
      4. TEST RESULTS
7. CONCLUSION AND FUTURE SCOPES
   1. CONCLUSION
      1. KEY ACHIEVVEMENTS
   2. FUTURE SCOPES
8. REFERENCES

**ACKNOWLEDGEMENT**

Initially, we thank the Almighty for being with us through every walk of our life and showering his blessings through the endeavor to put forth this report. Our sincere

thanks to our Chairman Mr. S. MEGANATHAN, B.E, F.I.E., our Vice Chairman Mr. ABHAY SHANKAR MEGANATHAN, B.E., M.S., and our respected Chairperson Dr. (Mrs.) THANGAM MEGANATHAN, Ph.D., for providing us with the requisite infrastructure and sincere endeavoring in educating us in their premier institution.

Our sincere thanks to Dr. S.N. MURUGESAN, M.E., Ph.D., our beloved Principal for his kind support and facilities provided to complete our work in time. We express our sincere thanks to Dr. K. SEKAR, M.E, Ph.D., Head of the Department of Artificial Intelligence and Machine Learning for his guidance and encouragement throughout the project work. We convey our sincere and deepest gratitude to our

internal guide, Mrs. M. THAMIZHARASI, M.E., Assistant Professor

Department of Artificial Intelligence and Machine Learning, Rajalakshmi Engineering College for her valuable guidance throughout the course of the project. We are very glad to thank our Project Coordinator, Mr. K. GOPINATH, M.E., Assistant Professor Department of Artificial Intelligence and Machine Learning for his useful tips during our review to build our project.

MOHAMMED IMRAN J (231801107)

PADMAVATHY R (231801122)

**ABSTRACT**

The **Apartment Management System** is a comprehensive solution designed to streamline administrative tasks and enhance residential community management. This system offers robust features to manage and organize critical information related to the residents and their apartments. Key features include a centralized database for storing and accessing detailed residential information, such as resident profiles, apartment details, contact information. The system enables efficient communication between residents and management. By automating routine operations, the Apartment Management System fosters an organized, secure, and user-friendly environment, improving the overall quality of life for residents and simplifying the tasks of property managers.

**LIST OF FIGURES**

STRUCTURE OF THE PROJECT

DATABASE USED FOR PROJECT

ADDITION OF DATAS

VIEWING OF DATAS

**LIST OF ABBREVIATIONS**

PHP Hypertext Preprocessor

SQL Sequential Query Language

JS JavaScript

HTML Hypertext Markup Language

CSS Cascading Style Sheets

DBMS Database Management System

CHAPTER 1

**INTRODUCTION**

* 1. **PROJECT DEFINITION**

The Apartment Management System (AMS) is a comprehensive, web-based solution developed to manage and organize data related to apartment buildings, residents, and their respective management activities. The AMS aims to streamline administrative operations such as tenant management, maintenance scheduling, billing, communication, and security, ultimately providing a more organized and efficient living environment for residents and simplifying property management tasks for managers.

1.1.1 PROJECT OBJECTIVES

* **Centralized Data Management:** Create a centralized database for storing resident information, apartment details, and contact records to ensure easy and quick access.
* **Efficient Communication:** Facilitate seamless communication between residents and management to address queries.
* **Automation of Routine Tasks:** Automate day-to-day operations like rent collection and notifications to save time and reduce errors.
* **User-Friendly Interface:** Develop an intuitive and easy-to-use interface for both residents and management, ensuring accessibility and efficiency.
* **Security and Data Privacy:** Implement a secure login and data encryption system to ensure residents' personal details remains protected.
  + 1. KEY FEATURES
* **Resident Profiles Management:** Allow the creation, modification, and deletion of resident profiles containing personal details, contact information, and apartment assignments.
* **Apartment Management:** Organize apartment data including apartment numbers, floor plans, rent details, and availability.
* **Security Features:** Implement role-based access control, ensuring that only authorized users (management, residents) can access specific data and functionalities.
  1. **NEED FOR PROPOSED SYSTEM**

The need for the **Apartment Management System (AMS)** arises from the necessity to streamline property management operations, improve communication, enhance data security, and provide residents with a better living experience. Below are the key reasons for the proposed system:

* **Centralized Database for Efficient Management:** A centralized database would allow property managers to store and access all relevant information in one place, including resident profiles, apartment details, payment histories, and maintenance requests. This centralized approach would not only reduce data redundancy but also enable quick and easy access to critical information, reducing administrative overhead.
* **Real-Time Information Access:** Residents would be able to access real-time information on their apartment details, rent payment status, maintenance requests, and community announcements. This level of transparency would reduce confusion, increase satisfaction, and create a more organized living environment. Property managers would also benefit from having access to up-to-date data on all apartment units, reducing the risk of errors.
* **Enhanced Security and Data Privacy:** The AMS would include robust security features, such as encrypted data storage and role-based access controls, ensuring that sensitive resident information remains protected. The system would limit access to personal data only to authorized personnel, enhancing privacy and security compared to traditional systems where data might be more vulnerable to unauthorized access.
* **Cost and Time Savings:** By automating administrative tasks and streamlining communication, the AMS would save time and reduce labor costs. Property managers can spend less time managing paperwork, resolving disputes and more time on strategic tasks, such as improving property amenities or enhancing resident experiences.
* **Improved Resident Satisfaction:** A key benefit of the AMS is its ability to enhance resident satisfaction. With efficient communication, transparent management, and an organized approach to maintenance and billing, residents will experience fewer issues and feel more confident in the management of their living space. This leads to better tenant retention rates and a higher overall quality of life.

**1.3 APPLICATIONS OF THE PROPOSED SYSTEM**

The **Apartment Management System (AMS)** is a versatile tool with a wide range of applications. It caters to various stakeholders in property management, including property managers, residents, and real estate firms. The following are key application areas:

* **Residential Societies**
* Simplifies the management of residential apartments in societies by digitizing resident profiles, maintenance requests, and billing records.
* Enhances communication between residents and society management through real-time notifications and announcements.
* **Property Management Firms**
* Enables firms to handle multiple properties from a single system, streamlining operations such as rent details, vacancy tracking and scheduling.
* Facilitates secure access to property information, improving transparency and trust with clients.
* **Real Estate Developers**
* Assists developers in tracking apartment sales, rentals, and tenant details during the property leasing phase.
* Allows for easy documentation and reporting, which aids in compliance with local regulations.
* **Rental Management Services**
* Supports rental agencies by providing tools to manage tenant records, rental agreements.
* **Individual Landlords**
* Provides an easy-to-use interface for small landlords to manage tenants and communication without requiring extensive technical knowledge.

CHAPTER 2

**LITERATURE REVIEW**

The **Apartment Management System (AMS)** is built on the principles of modern property management and web application development. This section reviews existing literature on property management systems, highlighting their evolution, limitations, and advancements.

**2.1 EXISTING SYSTEMS AND THEIR LIMITATIONS**

* **Traditional Systems**: Property management traditionally relied on paper-based systems and manual record-keeping. These methods are prone to errors, time-consuming, and lack scalability.
* **Excel and Simple Software Solutions**: Early digitization efforts included the use of spreadsheets and basic software for data organization. While an improvement, these systems often lack integration, real-time updates, and security.
* **Standalone Applications**: Standalone property management software offered significant improvements but faced challenges in user-friendliness, high costs, and limited customization.

**2.2 ADVANCEMENTS IN WEB-BASED PROPERTY MANAGEMENT SYSTEMS**

* Web-based systems emerged as a solution to the challenges of traditional and standalone systems, providing:
  + Centralized databases for easy access and management of information.
  + Real-time data updates and multi-user collaboration.
* **Key Technologies**:
  + **Flask Framework**: Flask's lightweight design and flexibility make it a popular choice for web applications like AMS.
  + **MySQL Database**: Its robust performance and scalability ensure efficient data storage and retrieval.
  + **Role-Based Access Control**: Widely used in property management applications to secure sensitive data and restrict unauthorized access.

**2.3 RELEVANT RESEARCH AND CASE STUDIES**

* Studies have shown that automated systems improve property management efficiency by up to 40%, reducing errors and improving tenant satisfaction.
* Case studies from companies using property management software indicate significant time and cost savings, as well as better compliance with regulations.
* The adoption of web-based platforms in property management is increasing due to their scalability and ability to handle complex operations.

**2.4 LIMITATIONS OF CURRENT LITERATURE**

* Existing research often overlooks the importance of user experience in property management software.
* Many studies focus on large-scale commercial applications, leaving gaps in understanding the needs of smaller residential or individual landlords.

**2.5 POSITIONING OF AMS**

The AMS addresses these limitations by combining scalability and user-friendliness, making it suitable for both large-scale property managers and individual landlords. It integrates modern technologies such as Flask and MySQL while prioritizing data security and an intuitive interface.

CHAPTER 3

**PROJECT FORMULATION**

**3.1 PLATFORM**

The **Apartment Management System** is built using a combination of technologies to ensure robustness, scalability, and ease of use. Below are the core platforms and tools used:

3.1.1 FRONTEND TECHNOLOGIES

* **HTML, CSS, and Bootstrap**: Used for creating user-friendly and responsive interfaces.

3.1.2 BACKEND TECHNOLOGIES

* **Python with Flask**: Flask is a lightweight web framework ideal for developing scalable and modular web applications. Python’s versatility ensures smooth backend operations.

3.1.3 DATABASE

* **MySQL**: A relational database system used for securely storing and retrieving data like resident details, apartment records, and other information.

3.1.4 DEVELOPMENT TOOLS

* **Text Editor/IDE**: Visual Studio Code or PyCharm for efficient coding.
* **Version Control**: Git and GitHub for version control and collaboration.

3.1.5 SERVER

* **Localhost**: During development, the system runs on a local server (localhost) using **Xampp** development server.
* **Deployment Options**: AWS, Heroku, or other cloud platforms for future scalability.

**3.2 SPECIFIC OBJECTIVES**

The specific objectives of the AMS are aligned with the project's mission to streamline apartment and resident management. These include:

1. **Centralized Data Storage**:  
   Ensure all apartment and resident information is stored in a unified, secure database for quick access and management.
2. **Dynamic Data Display**:  
   Implement real-time rendering of apartment and resident data through interactive web interfaces.
3. **User Roles and Security**:  
   Establish role-based access controls to protect sensitive information and restrict unauthorized access.
4. **Ease of Use**:  
   Design an intuitive interface that caters to both residents and property managers, minimizing the learning curve.
5. **Automated Processes**:  
   Automate tasks such as notifications, rent collection, and data management to reduce manual workload.
6. **Scalability**:  
   Develop a system that can accommodate more users and features without compromising performance.
7. **Real-Time Communication**:  
   Facilitate seamless communication between management and residents for faster query resolution and announcements.

**3.3 METHODOLOGY**

The **System Development Life Cycle (SDLC)** methodology is adopted for the development of the AMS. The project phases are outlined below:

3.3.1 REQUIREMENT ANALYSIS

* **Objective**: Understand the needs of the stakeholders, including property managers and residents.
* **Outcome**: Define system requirements such as functionalities, database schemas, and security protocols.

3.3.2 SYSTEM DESIGN

* **Database Design**: Develop an Entity-Relationship (ER) diagram to define the structure of tables for apartments and residents.
* **Architecture Design**: Define the system’s client-server architecture with Flask as the backend and MySQL as the database.

3.3.3 IMPLEMENTATION

* **Frontend Development**: Build interactive templates using HTML, CSS and JS.
* **Backend Development**: Implement Flask routes for managing CRUD operations for residents and apartments.
* **Database Integration**: Connect the Flask application to the MySQL database for data storage and retrieval.

3.3.4 TESTING

* **Unit Testing**: Test individual components, such as form validation and database queries.
* **Integration Testing**: Verify the seamless interaction between the frontend, backend, and database.
* **User Testing**: Conduct usability testing with sample users to ensure the interface meets their needs.

3.3.5 DEPLOYMENT

* Deploy the application on a local server (localhost) during development.
* Prepare for deployment on a cloud platform to support real-world usage.

3.3.6 MAINTENANCE

* Monitor the system for bugs, updates, and improvements.
* Collect feedback from users to identify areas for enhancement.

CHAPTER 4

**SYSTEM ANALYSIS AND DESIGN**

**4.1 SYSTEM ANALYSIS**

4.1.1 EXISTING SYSTEM

* In the absence of a centralized management system, property managers rely on manual record-keeping or standalone tools like Excel sheets.
* This approach is prone to data redundancy, inconsistencies, and inefficiencies in communication and management.

4.1.2 PROPOSED SYSTEM

The AMS aims to address these challenges by:

* Centralizing data storage for easy access and management.
* Automating routine tasks like rent collection and notifications.
* Providing secure and role-based access control to ensure data privacy.

**4.2 DATABASE DESIGN**

4.2.1 RESIDENTS TABLE

Stores detailed information about residents.

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Data Type** | **Description** |
| ResidentID | INT (Primary Key) | Unique identifier for each resident. |
| Name | VARCHAR(100) | Full name of the resident. |
| ContactNo | VARCHAR(15) | Phone number of the resident. |
| Email | VARCHAR(100) | Email address for communication. |
| ApartmentID | INT (Foreign Key) | Links the resident to an apartment. |
| Role | VARCHAR(50) | Role of the resident (e.g., tenant, owner). |

4.2.2 APARTMENTS TABLE

Stores detailed information about apartments.

|  |  |  |
| --- | --- | --- |
| **Field Name** | **Data Type** | **Description** |
| ApartmentID | INT (Primary Key) | Unique identifier for each apartment. |
| ApartmentNo | VARCHAR(50) | Apartment number or identifier. |
| Type | VARCHAR(50) | Type of apartment (e.g., 1BHK, 2BHK). |
| Rent | DECIMAL(10,2) | Monthly rent for the apartment. |

**4.3 SYSTEM ARCHITECTURE**

The AMS is based on a **3-tier architecture**:

1. **Presentation Layer**:
   * User interacts through the web interface.
   * Built using HTML, CSS, Bootstrap, and Jinja2 templates.
2. **Application Layer**:
   * Flask-based backend handles logic, routing, and communication with the database.
3. **Data Layer**:
   * MySQL database securely stores information about residents and apartments.

**4.4 UML DIAGRAMS**

4.4.1 USE CASE DIAGRAM

* **Actors**:
  + Property Manager: Manages apartments and residents.
  + Resident: Views personal and apartment information.
* **Use Cases**:
  + Add/Update/Delete/View Apartments.
  + Add/Update/Delete/View Residents.
  + Communicate with property management.

4.4.2 ENTITY-RELATIONSHIP (ER) DIAGRAM

* **Entities**:
  + Residents: Fields include ResidentID, Name, ContactNo, Email, ApartmentID, Role.
  + Apartments: Fields include ApartmentID, ApartmentNo, Type, Rent, Size.
* **Relationships**:
  + One-to-Many: Each apartment can have multiple residents, but each resident belongs to one apartment.

4.4.3 SEQUENCE DIAGRAM

Illustrates the process flow when a user (e.g., property manager) interacts with the system:

1. User sends a request to view residents or apartments.
2. Flask backend fetches data from the MySQL database.
3. Data is rendered dynamically on the web page.

4.4.4 CLASS DIAGRAM

* **Classes**:
  + Resident Class: Attributes include *ResidentID, Name, ContactNo, Email, ApartmentID, Role.*
  + Apartment Class: Attributes include *ApartmentID, ApartmentNo,Type,Rent*.

CHAPTER 5

**FUNCTIONAL DESCRIPTION**

**5.1 ADD RESIDENT**

5.1.1 PURPOSE

* Allows the property manager to add a new resident's information to the system.

5.1.2 IMPLEMENTATION

* **Route**: /add\_resident
  + **HTTP Methods**: GET and POST
* **Template**: add\_resident.html
* **Database Query**:

INSERT INTO Residents (Name, ContactNo, Email, ApartmentID, Role)

VALUES (%s, %s, %s, %s, %s);

5.1.3 WORKFLOW

1. User navigates to the *Add Resident* page.
2. The system displays a form to input resident details (name, contact number, email, apartment ID, role).
3. On form submission (POST request):
   * Data is validated and stored in the Residents table in the database.
   * User is redirected to the *View Residents* page.

5.1.4 ERROR HANDLING

* Checks for missing or invalid inputs (e.g., empty fields, incorrect email format).
* Ensures that the referenced ApartmentID exists in the Apartments table.

**5.2 VIEW RESIDENTS**

5.2.1 PURPOSE

* Displays a list of all residents with their details.

5.2.2 IMPLEMENTATION

* Route: /view\_residents
  + HTTP Methods: GET
* Template: view\_residents.html
* Database Query:

SELECT \* FROM Residents;

5.2.3 WORKFLOW

1. User navigates to the *View Residents* page.
2. Flask fetches all records from the Residents table.
3. The data is passed to the view\_residents.html template and displayed in a tabular format.

5.2.4 FEATURES

* Columns: ID, Name, Contact Number, Email, Apartment ID, and Role.
* Data is rendered dynamically, allowing real-time updates.

5.2.5 ERROR HANDLING

* Displays a friendly message if no residents are found in the database.

**5.3 ADD APARTMENT**

5.3.1 PURPOSE

* Allows the property manager to add a new apartment's details to the system.

5.3.2 IMPLEMENTATION

* **Route**: /add\_apartment
  + **HTTP Methods**: GET and POST
* **Template**: add\_apartment.html
* **Database Query**:

INSERT INTO Apartments (ApartmentID, ApartmentNo, Type, Rent, Size)

VALUES (%s, %s, %s, %s, %s);

5.3.3 WORKFLOW

1. User navigates to the *Add Apartment* page.
2. The system displays a form to input apartment details (ID, number, type, rent, size).
3. On form submission (POST request):
   * Data is validated and stored in the Apartments table in the database.
   * User is redirected to the *View Apartments* page.

5.3.4 ERROR HANDLING

* Validates inputs (e.g., ensures apartment ID is unique, rent is a valid number).
* Checks for missing or invalid fields.

**5.4 VIEW APARTMENTS**

5.4.1 PURPOSE

* Displays a list of all apartments with their details.

5.4.2 IMPLEMENTATION

* **Route**: /view\_apartments
  + **HTTP Methods**: GET
* **Template**: view\_apartments.html
* **Database Query**:

SELECT \* FROM Apartments;

5.4.3 WORKFLOW

1. User navigates to the *View Apartments* page.
2. Flask fetches all records from the Apartments table.
3. The data is passed to the view\_apartments.html template and displayed in a tabular format.

5.4.4 FEATURES

* Columns: Apartment ID, Apartment Number, Type, Rent, and Size.
* Data is rendered dynamically, providing real-time updates.

5.4.5 ERROR HANDLING

* Displays a friendly message if no apartments are found in the database.

**5.5 SUMMARY OF FUNCTIONS**

|  |  |  |  |
| --- | --- | --- | --- |
| **Function** | **Route** | **Template** | **Key Features** |
| Add Resident | /add\_resident | add\_resident.html | Form to add new residents; stores data in DB. |
| View Residents | /view\_residents | view\_residents.html | Displays all residents in a table. |
| Add Apartment | /add\_apartment | add\_apartment.html | Form to add new apartments; stores data in DB. |
| View Apartments | /view\_apartments | view\_apartments.html | Displays all apartments in a table. |

CHAPTER 6

**IMPLEMENTATION AND TESTING**

**6.1 IMPLEMENTATION**

The implementation of AMS involves integrating the frontend, backend, and database. Below is a breakdown of the implementation process:

6.1.1 FRONTEND IMPLEMENTATION

* **HTML, CSS, and Bootstrap**:
  + Used to create interactive forms and tables.
  + Pages like add\_resident.html and view\_residents.html are structured for ease of use.

6.1.2 BACKEND IMPLEMENTATION

* **Flask Routes**:
  + Routes such as /add\_resident, /view\_residents, /add\_apartment, and /view\_apartments handle HTTP requests.
* **Form Handling**:
  + Data submitted through forms is validated and processed using POST requests.
* **Database Connectivity**:
  + MySQL queries are executed using mysql.connector to fetch, insert, and manage records.

6.1.3 DATABASE IMPLEMENTATION

* MySQL tables (Residents and Apartments) were created based on the defined schema.
* Data integrity constraints (e.g., primary and foreign keys) were implemented to ensure consistency.

6.1.4 DEPLOYMENT

* The system was tested locally on a Flask development server (http://127.0.0.1:5000).
* Prepared for deployment on cloud platforms such as AWS or Heroku for scalability.

**6.2 TESTING**

The AMS was rigorously tested to ensure it meets functional and non-functional requirements.

6.2.1 UNIT TESTING

* Each Flask route (e.g., /add\_resident) was tested independently for proper form submission and database insertion.
* Edge cases such as empty fields or invalid data formats were handled appropriately.

6.2.2 INTEGRATION TESTING

* Verified that all components (frontend, backend, and database) work together seamlessly.
* Tested scenarios like adding a resident and immediately viewing the updated list of residents.

6.2.3 USER TESTING

* Conducted with sample users to ensure the user interface is intuitive and the application meets their needs.
* Feedback was incorporated to improve navigation and data visibility.

6.2.4 TEST RESULTS

* All critical functionalities were validated, including:
  + Adding residents and apartments.
  + Viewing and managing data in real-time.
  + Handling invalid inputs gracefully.

CHAPTER 7

**CONCLUSION AND FUTURE SCOPES**

**7.1 CONCLUSION**

The **Apartment Management System (AMS)** successfully achieves its primary goal of streamlining apartment and resident management. By automating administrative tasks, centralizing data, and providing real-time access to information, the AMS enhances the efficiency of property management and improves resident satisfaction.

7.1.1 KEY ACHIEVEMENTS

* Simplified the management of resident and apartment information.
* Enhanced security with role-based access and data encryption.
* Provided a user-friendly interface for seamless interaction.

**7.2 FUTURE SCOPES**

The AMS can be further enhanced by incorporating the following features:

* **Authentication and Authorization**:
  + Implement user login with role-based dashboards for property managers and residents.
* **Payment Integration**:
  + Enable online rent payment through integrated payment gateways.
* **Maintenance Management**:
  + Add modules to schedule and track maintenance requests.
* **Data Analytics**:
  + Generate reports and insights on rent collection, occupancy rates, and other metrics.
* **Mobile Application**:
  + Develop a mobile app for easier access by residents and property managers.

CHAPTER 8

**8.1 SCHEMA OF THE PROJECT**

Mini\_project/

---app.py

---db\_setup.sql

------templates/

---------add\_apartment.html

---------add\_resident.html

---------base.html

---------index.html

---------view\_residents.html

---------view\_apartments.html

------static/

---------css/

------------style.css

---------js/

------------java.js

**8.2 SOURCE CODE**

1. **app.py**

from flask import Flask, render\_template, request, redirect, url\_for

import mysql.connector

app = Flask(\_\_name\_\_)

def get\_db\_connection():

return mysql.connector.connect(

host='localhost',

user='root',

database='ApartmentManagement'

)

@app.route('/')

def index():

return render\_template('index.html', title="Home")

@app.route('/add\_resident', methods=['GET', 'POST'])

def add\_resident():

if request.method == 'POST':

name = request.form['name']

contact\_no = request.form['contact\_no']

email = request.form['email']

apartment\_id = request.form['apartment\_id']

role = request.form['role']

conn = get\_db\_connection()

cursor = conn.cursor()

cursor.execute(

"INSERT INTO Residents (Name, ContactNo, Email, ApartmentID, Role) VALUES (%s, %s, %s, %s, %s)",

(name, contact\_no, email, apartment\_id, role)

)

conn.commit()

cursor.close()

conn.close()

return redirect(url\_for('view\_residents'))

return render\_template('add\_resident.html', title="Add Resident")

@app.route('/view\_residents')

def view\_residents():

conn = get\_db\_connection()

cursor = conn.cursor()

cursor.execute("SELECT \* FROM Residents")

residents = cursor.fetchall()

cursor.close()

conn.close()

return render\_template('view\_residents.html', residents=residents, title="View Residents")

@app.route('/add\_apartment', methods=['GET', 'POST'])

def add\_apartment():

if request.method == 'POST':

apartment\_id = request.form['apartment\_id']

apartment\_no = request.form['apartment\_no']

type = request.form['type']

rent = request.form['rent']

size = request.form['size']

conn = get\_db\_connection()

cursor = conn.cursor()

cursor.execute(

"INSERT INTO `Apartments`(`ApartmentID`, `ApartmentNo`, `Type`, `Rent`, `Size`) VALUES (%s,%s,%s,%s,%s)",

(apartment\_id, apartment\_no, type, rent, size)

)

conn.commit()

cursor.close()

conn.close()

return redirect(url\_for('view\_apartments'))

return render\_template('add\_apartment.html', title="Add Apartment")

@app.route('/view\_apartments')

def view\_apartments():

conn = get\_db\_connection()

cursor = conn.cursor()

cursor.execute("SELECT \* FROM Apartments")

apartments = cursor.fetchall()

cursor.close()

conn.close()

return render\_template('view\_apartments.html', apartments=apartments, title="View Apartments")

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

1. **db\_setup.sql**

-- Create Database

CREATE DATABASE ApartmentManagement;

USE ApartmentManagement;

-- Create Tables

CREATE TABLE Residents (

    ResidentID INT AUTO\_INCREMENT PRIMARY KEY,

    Name VARCHAR(100) NOT NULL,

    ContactNo VARCHAR(15) NOT NULL,

    Email VARCHAR(100) UNIQUE,

    ApartmentID INT,

    Role ENUM('Owner', 'Tenant') NOT NULL,

    FOREIGN KEY (ApartmentID) REFERENCES Apartments(ApartmentID)

);

CREATE TABLE Apartments (

    ApartmentID INT AUTO\_INCREMENT PRIMARY KEY,

    ApartmentNo VARCHAR(10) UNIQUE NOT NULL,

    Type ENUM('1BHK', '2BHK', '3BHK') NOT NULL,

    Rent DECIMAL(10, 2),

    Size INT NOT NULL

);

CREATE TABLE Payments (

    PaymentID INT AUTO\_INCREMENT PRIMARY KEY,

    ResidentID INT,

    Amount DECIMAL(10, 2) NOT NULL,

    PaymentDate DATE NOT NULL,

    PaymentType ENUM('Rent', 'Maintenance') NOT NULL,

    FOREIGN KEY (ResidentID) REFERENCES Residents(ResidentID)

);

CREATE TABLE Complaints (

    ComplaintID INT AUTO\_INCREMENT PRIMARY KEY,

    ResidentID INT,

    Description TEXT NOT NULL,

    Status ENUM('Pending', 'Resolved') DEFAULT 'Pending',

    SubmissionDate DATE NOT NULL,

);

CREATE TABLE Staff (

    StaffID INT AUTO\_INCREMENT PRIMARY KEY,

    Name VARCHAR(100) NOT NULL,

    Role VARCHAR(50) NOT NULL,

    ContactNo VARCHAR(15) NOT NULL

);

1. **app\_apartment.html**

{% extends 'base.html' %}

{% block content %}

<h2>Add a New Apartment</h2>

<form method="POST">

<div class="mb-3">

<label for="apartment\_id" class="form-label">Apartment ID</label>

<input type="text" class="form-control" id="apartment\_id" name="apartment\_id" required>

</div>

<div class="mb-3">

<label for="apartment\_no" class="form-label">Apartment Number</label>

<input type="text" class="form-control" id="apartment\_no" name="apartment\_no" required>

</div>

<div class="mb-3">

<label for="type" class="form-label">Type</label>

<select class="form-select" id="type" name="type">

<option value="1BHK">1BHK</option>

<option value="2BHK">2BHK</option>

<option value="3BHK">3BHK</option>

</select>

</div>

<div class="mb-3">

<label for="rent" class="form-label">Rent</label>

<input type="number" class="form-control" id="rent" name="rent" required>

</div>

<div class="mb-3">

<label for="size" class="form-label">Size</label>

<input type="number" class="form-control" id="size" name="size" required>

</div>

</div>

<button type="submit" class="btn btn-primary">Add Apartment</button>

</form>

{% endblock %}

1. **add\_residents.html**

{% extends 'base.html' %}

{% block content %}

<h2>Add a New Resident</h2>

<form method="POST">

<div class="mb-3">

<label for="name" class="form-label">Name</label>

<input type="text" class="form-control" id="name" name="name" required>

</div>

<div class="mb-3">

<label for="contact\_no" class="form-label">Contact Number</label>

<input type="text" class="form-control" id="contact\_no" name="contact\_no" required>

</div>

<div class="mb-3">

<label for="email" class="form-label">Email</label>

<input type="email" class="form-control" id="email" name="email" required>

</div>

<div class="mb-3">

<label for="apartment\_id" class="form-label">Apartment ID</label>

<input type="number" class="form-control" id="apartment\_id" name="apartment\_id" required>

</div>

<div class="mb-3">

<label for="role" class="form-label">Role</label>

<select class="form-select" id="role" name="role">

<option value="Owner">Owner</option>

<option value="Tenant">Tenant</option>

</select>

</div>

<button type="submit" class="btn btn-primary">Add Resident</button>

</form>

{% endblock %}

1. **base.html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>{{ title }}</title>

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet">

<link rel="stylesheet" href="{{ url\_for('static', filename='css/style.css') }}">

</head>

<body>

<nav class="navbar navbar-expand-lg navbar-dark bg-dark">

<div class="container-fluid">

<a class="navbar-brand" href="/">Apartment Management</a>

<button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarNav">

<span class="navbar-toggler-icon"></span>

</button>

<div class="collapse navbar-collapse" id="navbarNav">

<ul class="navbar-nav">

<li class="nav-item"><a class="nav-link" href="/">Home</a></li>

<li class="nav-item"><a class="nav-link" href="/add\_resident">Add Resident</a></li>

<li class="nav-item"><a class="nav-link" href="/view\_residents">View Residents</a></li>

<li class="nav-item"><a class="nav-link" href="/add\_apartment">Add apartment</a></li>

<li class="nav-item"><a class="nav-link" href="/view\_apartments">View apartments</a></li>

</ul>

</div>

</div>

</nav>

<div class="container mt-4">

{% block content %}{% endblock %}

</div>

<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></script>

</body>

</html>

1. **index.html**

{% extends 'base.html' %}

{% block content %}

<h2>Residents List</h2>

<table class="table table-striped">

<thead>

<tr>

<th>ID</th>

<th>Name</th>

<th>Contact</th>

<th>Email</th>

<th>Apartment ID</th>

<th>Role</th>

</tr>

</thead>

<tbody>

{% for resident in residents %}

<tr>

<td>{{ resident[0] }}</td>

<td>{{ resident[1] }}</td>

<td>{{ resident[2] }}</td>

<td>{{ resident[3] }}</td>

<td>{{ resident[4] }}</td>

<td>{{ resident[5] }}</td>

</tr>

{% endfor %}

</tbody>

</table>

{% endblock %}

1. **view\_apartments.html**

{% extends 'base.html' %}

{% block content %}

<h2>List of Apartments</h2>

<table class="table table-striped">

<thead>

<tr>

<th>Apartment ID</th>

<th>Apartment Name</th>

<th>Address</th>

</tr>

</thead>

<tbody>

{% for apartment in apartments %}

<tr>

<td>{{ apartment[0] }}</td>

<td>{{ apartment[1] }}</td>

<td>{{ apartment[2] }}</td>

</tr>

{% endfor %}

</tbody>

</table>

{% endblock %}

1. **view\_residents.html**

{% extends 'base.html' %}

{% block content %}

<h2>Residents List</h2>

<table class="table table-striped">

<thead>

<tr>

<th>ID</th>

<th>Name</th>

<th>Contact</th>

<th>Email</th>

<th>Apartment ID</th>

<th>Role</th>

</tr>

</thead>

<tbody>

{% for resident in residents %}

<tr>

<td>{{ resident[0] }}</td>

<td>{{ resident[1] }}</td>

<td>{{ resident[2] }}</td>

<td>{{ resident[3] }}</td>

<td>{{ resident[4] }}</td>

<td>{{ resident[5] }}</td>

</tr>

{% endfor %}

</tbody>

</table>

{% endblock %}

CHAPTER 8

**REFERENCES**

1. **Flask Framework Documentation**  
   *Flask Official Documentation*. Retrieved from https://flask.palletsprojects.com
2. **MySQL Documentation**  
   *MySQL Database Reference*. Retrieved from <https://dev.mysql.com/doc/>
3. **Bootstrap Framework**  
   *Bootstrap Official Documentation*. Retrieved from <https://getbootstrap.com>
4. Related research on web-based property management systems from IEEE and ACM Digital Library.
5. Tutorials and guides on Flask-MySQL integration from platforms like GeeksforGeeks and RealPython.
6. Case studies on property management systems and their impact on administrative efficiency.

**5.1 Add R**