PATIENT REGISTRATION FORM

A MINI PROJECT REPORT

SUBMITTED BY

Mervin J 220701163

Manick Vishal C 220701158

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

RAJALAKSHMI ENGINEERING COLLEGE

THANDALAM

BONAFIDE CERTIFICATE

Certified that this project report "PATIENT REGISTRATION FORM" is the bonafide work of "MANICK VISHAL C (220701158), MERVIN J(220701163)," who carried out the project work under my supervision.

Submitted for the Practical Examination held on _____

SIGNATURE

DR R.SABITHA

Professor and II Year Academic Head,

Computer Science and Engineering,

Rajalakshmi Engineering College

(Autonomous),

Thandalam, Chennai – 602105

SIGNATURE

Ms D.KALPANA

Assistant Professor

Computer Science and Engineering,

Rajalakshmi Engineering College

(Autonomous),

Thandalam, Chennai – 602105

ABSTRACT

The Patient Registration Form project aims to streamline the process of gathering essential patient information in healthcare settings. This project involves developing an intuitive and user-friendly digital form that captures a wide range of patient data, including personal details, contact information, medical history, insurance details, and emergency contacts. The primary objectives are to enhance data accuracy, improve patient intake efficiency, and ensure secure handling of sensitive information. By leveraging modern web technologies and secure database management systems, the project seeks to replace traditional paper-based forms, reduce administrative burdens, and provide a seamless experience for both patients and healthcare providers. The implementation includes features such as real-time data validation, secure data encryption, and integration with existing hospital management systems. Ultimately, the Patient Registration Form project contributes to improved patient care, reduced wait times, and optimized workflow in medical facilities.

TABLE OF CONTENTS

1.1 INTRODUCTION

The digital Patient Registration Form will serve as a comprehensive tool to capture essential patient details, including personal information, medical history, insurance data, and emergency contacts. By leveraging modern web technologies, the form will provide an intuitive and user-friendly interface, ensuring that patients can easily complete the registration process with minimal assistance.

Key features of the project include real-time data validation to minimize errors, secure encryption to protect sensitive information, and integration capabilities with existing hospital management systems to streamline workflows. This digital transformation not only enhances data accuracy but also reduces the administrative burden on healthcare staff, allowing them to focus more on patient care.

The implementation of the Patient Registration Form project is expected to bring significant benefits, such as reduced wait times for patients, improved data integrity, and enhanced operational efficiency within healthcare facilities. By modernizing the patient intake process, the project contributes to the overall goal of delivering better healthcare services and improving patient satisfaction

1.2 OBJECTIVES

1. **Enhance Data Accuracy:**
    - Minimize errors in patient information by implementing real-time data validation and automated error-checking mechanisms.
2. **Improve Efficiency:**
    - Streamline the patient registration process to reduce wait times and administrative workload, allowing healthcare staff to focus more on patient care.
3. **Ensure Data Security:**

- Protect sensitive patient information through secure data encryption and compliance with relevant data protection regulations and standards.

4. **Facilitate Easy Access and Retrieval:**
    - Enable quick and easy access to patient information for authorized personnel, enhancing the ability to provide timely and effective medical care.
5. **Support Integration with Existing Systems:**
    - Ensure seamless integration with existing hospital management systems and electronic health records (EHR) for comprehensive patient data management.
6. **Enhance User Experience:**
    - Develop an intuitive and user-friendly interface that simplifies the registration process for patients, including features that accommodate various patient demographics and needs.
7. **Reduce Paperwork:**
    - Decrease reliance on paper-based forms, contributing to environmental sustainability and reducing physical storage requirements.
8. **Optimize Workflow:**
    - Automate and optimize administrative workflows associated with patient intake, including scheduling, billing, and insurance verification processes.

1.3 MODULES

User Authentication and Access Control Module:

Provides secure login and registration functionalities for patients and healthcare staff.

Manages user roles, permissions, and ensures data privacy through multi-factor authentication and encrypted sessions.

Patient Information and Medical History Module:

Collects and manages essential patient data, including personal details, contact information, and comprehensive medical history.

Supports uploading and attaching medical documents, ensuring all relevant health information is accessible in one place.

Insurance and Emergency Contact Module:

Records insurance details such as provider information, policy number, and coverage specifics.

Captures emergency contact information, allowing for multiple contacts and their relationship to the patient.

Appointment Scheduling and Management Module:

Facilitates scheduling, rescheduling, and cancellation of appointments.

Integrates with the hospital’s calendar system to check availability, send reminders, and manage patient appointments efficiently.

Data Security and Integration Module:

Ensures secure data encryption for storage and transmission of sensitive patient information.

Provides seamless integration with existing hospital management systems and electronic health records (EHR) for synchronized data management and interoperability

2. SURVEY OF TECHNOLOGIES

2.1 SOFTWARE DESCRIPTION:

- **Frontend:**
  - HTML5, CSS3, JavaScript or Python for responsive and interactive user interfaces.
  - Frameworks such as React.js or Angular for dynamic and scalable front-end development.
- **Backend:**
  - Server-side languages such as Node.js, Python (Django), or Java (Spring Boot) for handling business logic and server operations.
  - RESTful API design for efficient communication between the frontend and backend.
- **Database:**
  - Relational databases like MySQL or PostgreSQL for structured data storage.
  - NoSQL databases like MongoDB for flexible and scalable data management.
- **Security:**
  - SSL/TLS encryption for secure data transmission.
  - OAuth2 and JWT for secure authentication and authorization.
  - Regular security audits and compliance with data protection regulations (e.g., HIPAA, GDPR).

- Deployment:
    - Cloud-based deployment using platforms such as AWS, Azure, or Google Cloud for scalability and reliability.
    - Containerization with Docker for consistent environment management and easy scaling.

Benefits:

- Improved Data Accuracy:
    - Real-time validation and error-checking reduce the risk of incorrect data entry.
- Enhanced Efficiency:
    - Streamlined workflows reduce administrative burdens and shorten patient wait times.
- Secure Data Handling:
    - Advanced encryption and access controls ensure the confidentiality and integrity of patient data.
- Better Patient Experience:
    - Intuitive interfaces make it easy for patients to complete registration forms, improving their overall experience.
- Operational Integration:
    - Seamless integration with existing systems enhances the efficiency and coordination of healthcare services.

2.2 LANGUAGES

2.2.1 SQL

SQL, or Structured Query Language, is a standardized programming language used for managing and manipulating relational databases. It is the primary language for interacting with database systems, allowing users to perform a wide range of operations, including:

1. **Data Retrieval:**
    - SQL is used to query databases and retrieve specific data through commands such as **SELECT**, **FROM**, **WHERE**, **JOIN**, and **GROUP BY**.

2. **Data Manipulation:**
    - It enables users to insert (**INSERT**), update (**UPDATE**), and delete (**DELETE**) records in a database, thus modifying the data stored within tables.

3. **Database Management:**
    - SQL provides commands for defining database structures (**CREATE**, **ALTER**, **DROP**) and managing access controls (**GRANT**, **REVOKE**).

4. **Data Integrity:**
    - It supports the enforcement of data integrity through constraints like primary keys, foreign keys, unique constraints, and check constraints.

5. **Transaction Control:**
    - SQL facilitates transaction management with commands such as **BEGIN TRANSACTION**, **COMMIT**, and **ROLLBACK**, ensuring data consistency and reliability.

2.2.2 PYTHON

Python, traditionally known for its backend capabilities, is also used in front-end development, primarily through frameworks like Django and Flask with template engines (e.g., Jinja2) and tools like PyScript. Although JavaScript is the predominant language for client-side scripting, Python can contribute to front-end development in the following ways:

1. **Web Frameworks:**
    - Django and Flask can render dynamic HTML content using template engines, allowing developers to create sophisticated web pages with Python.

2. **PyScript:**
    - PyScript is an emerging tool that allows developers to write Python code directly in web pages, expanding Python's role in front-end development.

3. **Integration with JavaScript:**
    - Python can work alongside JavaScript, handling server-side logic while generating HTML, CSS, and JavaScript for the client side.

4. **Rapid Prototyping:**
    - Python's simplicity and readability make it an excellent choice for quickly prototyping web applications.

3. REQUIREMENTS AND ANALYSIS

3.1 REQUIREMENTS SPECIFICATION

Software Requirement Specification:

1. Introduction

1.1 Purpose: The purpose of this document is to provide a detailed specification for the Patient Registration Form software, which will streamline the patient intake process by replacing traditional paper-based forms with a digital, user-friendly, and secure system.

1.2 Scope: The software will capture and manage patient information, including personal details, medical history, insurance information, emergency contacts, and appointment scheduling. It will ensure data accuracy, security, and integration with existing hospital management systems.

1.3 Definitions, Acronyms, and Abbreviations:

- SRS: Software Requirement Specification
- EHR: Electronic Health Record
- UI: User Interface
- UX: User Experience

1.4 References:

- IEEE SRS Standard
- HIPAA Regulations
- GDPR Regulations

2. Overall Description

2.1 Product Perspective: The Patient Registration Form software will function as a web application, integrating with existing hospital management systems and EHRs. It will consist of a frontend interface for users and a backend system for data processing and storage.

2.2 Product Functions:

- User Authentication and Access Control
- Patient Information and Medical History Management
- Insurance and Emergency Contact Management
- Appointment Scheduling and Management
- Data Security and Integration

2.3 User Characteristics:

- Patients: Various ages and technical proficiency levels.
- Healthcare Staff: Administrators, doctors, and nurses with basic computer skills.

2.4 Constraints:

- Must comply with HIPAA and GDPR regulations.
- High availability and reliability requirements.
- Responsive design for accessibility on multiple devices.

3. Specific Requirements

3.1 Functional Requirements:

3.1.1 User Authentication and Access Control:

- Users must be able to register and log in securely.
- System must support multi-factor authentication.
- Role-based access control to differentiate permissions for patients and healthcare staff.

3.1.2 Patient Information and Medical History Management:

- Form to capture patient personal details (name, date of birth, gender, contact information).
- Fields for detailed medical history (past illnesses, surgeries, allergies, medications, family medical history).
- Ability to upload and attach medical documents.

3.1.3 Insurance and Emergency Contact Management:

- Fields to capture insurance provider details, policy numbers, and coverage specifics.
- Ability to add multiple emergency contacts, including name, relationship, and contact information.

3.1.4 Appointment Scheduling and Management:

- Patients must be able to schedule, reschedule, and cancel appointments.
- Integration with the hospital’s calendar system to check availability.
- Automated email/SMS reminders for appointments.

3.1.5 Data Security and Integration:

- All data must be encrypted during transmission and storage.
- Integration with existing hospital management systems and EHRs for synchronized data.

- Regular security audits and compliance checks.

3.2 Non-Functional Requirements:

3.2.1 Performance:

- System should handle concurrent access by multiple users efficiently.
- Response time for form submission should be under 2 seconds.

3.2.2 Usability:

- User-friendly interface with clear instructions and prompts.
- Responsive design for use on desktops, tablets, and smartphones.
- Accessibility features to support users with disabilities.

3.2.3 Reliability:

- System uptime of 99.9%.
- Regular backups and disaster recovery mechanisms in place.

3.2.4 Security:

- Compliance with HIPAA and GDPR regulations.
- Secure authentication mechanisms.
- Regular security updates and patches.

3.2.5 Maintainability:

- Modular codebase for ease of updates and maintenance.
- Comprehensive documentation for developers and users.

3.2.6 Scalability:

- System should be scalable to accommodate increasing numbers of users and data volume.

4. External Interface Requirements

4.1 User Interfaces:

- Web-based user interface accessible via standard web browsers.
- Clean and intuitive design with clear navigation.

4.2 Hardware Interfaces:

- Compatible with standard computer and mobile device hardware.

4.3 Software Interfaces:

- Integration APIs for connecting with hospital management systems and EHRs.
- Secure communication protocols for data exchange.

4.4 Communication Interfaces:

- Internet connectivity for accessing the web application.
- Secure data transmission using HTTPS/SSL.

5. Other Requirements

5.1 Compliance:

- Ensure compliance with healthcare regulations (HIPAA, GDPR).
- Accessibility standards compliance (e.g., WCAG).

5.2 Documentation:

- User manuals and online help resources.
- Technical documentation for developers and system administrators.

3.2 SOFTWARE AND HARDWARE REQUIREMENTS

1. Hardware requirements:

. Development Machines:

- **Processor:** Intel i5 or AMD Ryzen 5 and above
- **RAM:** 8 GB minimum, 16 GB recommended
- **Storage:** 256 GB SSD minimum, 512 GB SSD recommended
- **Operating System:** Windows 10/11, macOS, or Linux

2. Server Requirements:

For Small to Medium Deployment:

- **Processor:** Intel Xeon E3 or AMD EPYC 7002 Series and above
- **RAM:** 16 GB minimum, 32 GB recommended
- **Storage:** 512 GB SSD minimum, 1 TB SSD recommended
- **Network:** High-speed internet connection, 1 Gbps network interface card

For Large Deployment:

- **Processor:** Dual Intel Xeon Gold or AMD EPYC 7003 Series
- **RAM:** 64 GB minimum, 128 GB recommended
- **Storage:** 1 TB NVMe SSD minimum, 2 TB NVMe SSD recommended
- **Network:** High-speed internet connection, 10 Gbps network interface card

3. Backup and Recovery:

- **Backup Storage:** External hard drives, network-attached storage (NAS), or cloud storage solutions (AWS S3, Azure Blob Storage)
- **Recovery Tools:** Regular backups using software like Veeam or Acronis

4. Additional Requirements:

- **Firewall:** Hardware or software firewall for network security

1. **Frontend:**

- **Languages like** Html,CSS or python
- **Other Tools:** Bootstrap or Material-UI for UI components

2. **Backend:**

- **Languages:** Node.js, Python (Django), or Java (Spring Boot)
- **Frameworks:** Express.js (for Node.js), Django (for Python), or Spring Boot (for Java)
- **Database:** MySQL, PostgreSQL, or MongoDB
- **API Design:** RESTful API for communication between frontend and backend

3. **Server:**

- **Web Server:** Nginx or Apache
- **Application Server:** Node.js, Gunicorn (for Python), or Tomcat (for Java)

4. **Security:**

- **Encryption:** SSL/TLS for secure data transmission
- **Authentication:** OAuth2, JWT
- **Compliance:** Tools to ensure HIPAA and GDPR compliance

5. **Development Tools:**

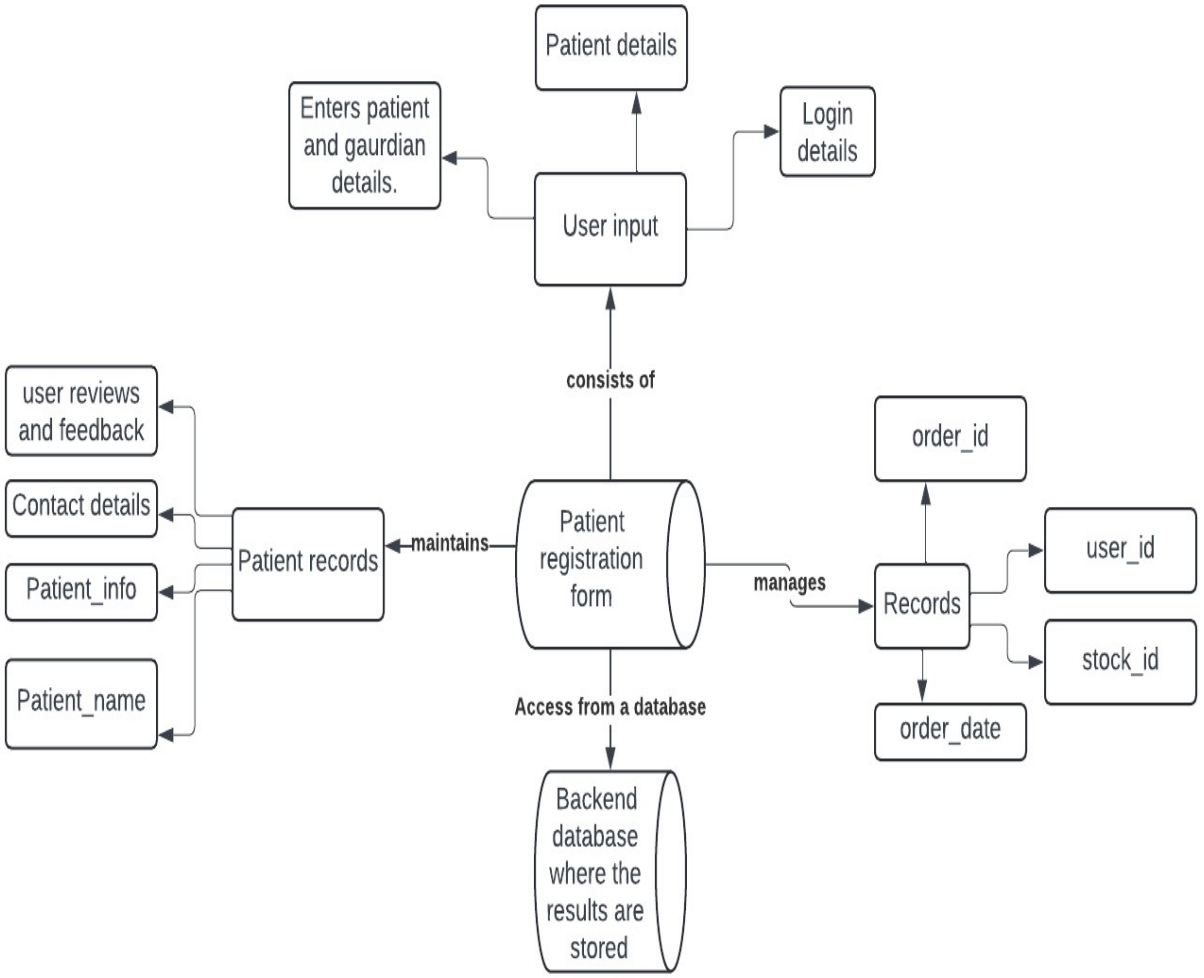
- **IDE:** Visual Studio Code, PyCharm, or IntelliJ IDEA
- **Version Control:** Git and GitHub or GitLab
- **Project Management:** Jira or Trello

3.3 ARCHITECTURE DIAGRAM

An architecture diagram is a visual representation of the structure, components, and interactions of a software system or application. It provides a high-level overview of the system's architecture, including its various layers, modules, and dependencies. Architecture diagrams typically depict the relationships between different components,

such as client-server interactions, data flows, and communication protocols. They help stakeholders, including developers, architects, and project managers, to understand the overall design and functionality of the system, facilitating communication and decision-making throughout the development process. Architecture diagrams are invaluable tools for designing, documenting, and communicating complex software architectures, enabling teams to build scalable, maintainable, and robust systems.

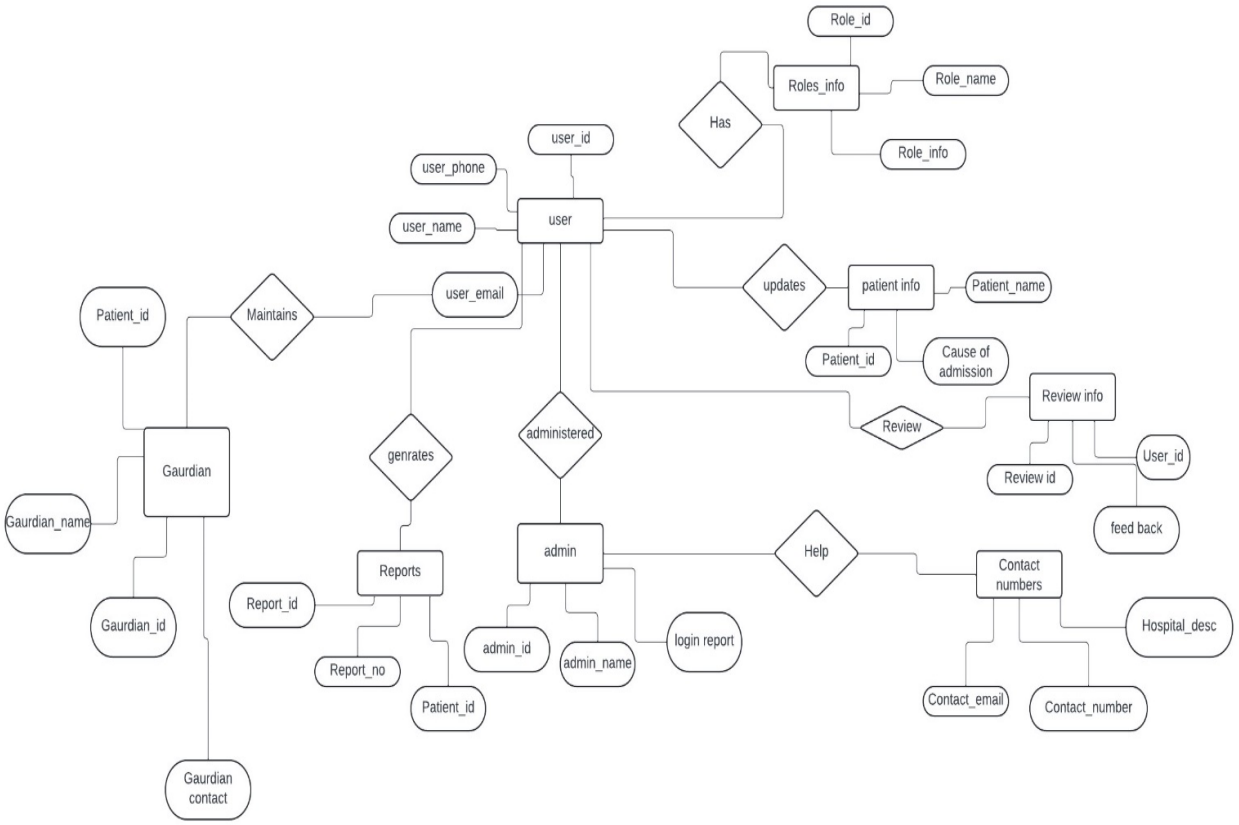
ARCHITECTURE DIAGRAM FOR PATIENT REGISTRATION FORM


Patient details
Enters patient and gaurdian details.
Login details
User input
consists of
user reviews and feedback
Contact details
Patient_info
Patient_name
Patient records
maintains
Patient registration form
manages
order_id
Records
user_id
stock_id
order_date
Access from a database
Backend database where the results are stored

3.4 ENTITY RELATIONSHIP DIAGRAM

An Entity-Relationship (ER) diagram is a visual representation of the relationships between entities in a database. It illustrates the logical structure of the database, depicting entities as rectangles, attributes as ovals, and relationships as lines connecting entities. ER diagrams are crucial in database design as they provide a clear understanding of how different entities relate to each other. They help database designers to identify key entities, define their attributes, and establish relationships between them, such as one-to-one, one-to-many, or many-to-many relationships. ER diagrams serve as a blueprint for database implementation, aiding in the creation of efficient and well-organized database systems that accurately reflect real-world scenarios.

ER DIAGRAM FOR PATIENT REGISTRATION FORM



3.5 NORMALISATION

Normalizing patient registration data in a mini-project involves organizing the data in a structured manner to eliminate redundancy and ensure data integrity.

Identify Data Entities:

Start by identifying the different entities involved in patient registration. Common entities might include patients, doctors, appointments, medical history, etc.

Define Attributes:

List down all the attributes (or fields) associated with each entity. For example, for the 'patients' entity, attributes might include patient ID, name, date of birth, contact information, etc.

Eliminate Redundancy:

Analyze the attributes to identify any redundant data. For instance, if both patient name and patient ID are stored in the same table, it's redundant because the patient ID can uniquely identify the patient.

Organize Data into Tables:

Based on identified entities and attributes, organize the data into separate tables. Each table should represent a single entity, and each attribute should correspond to a column in the table.

Apply Normalization Rules:

Apply normalization rules to eliminate data redundancy and dependency issues. This typically involves breaking down large tables into smaller ones and establishing relationships between them.

First Normal Form (1NF):

PatientID	Name	DOB	Phone	Doctor	Specialty
1	John Smith	1980-05-25	123-456-7890	Dr. Johnson	Pediatrics, Neurology
2	Jane Doe	1975-08-15	987-654-3210	Dr. Adams	Cardiology

Ensure that each column contains atomic values, meaning no multiple values or repeating groups within a single column.

Patients Table:

PatientID	Name	DOB	Phone

1 | John Smith | 1980-05-25 | 123-456-7890

2 | Jane Doe | 1975-08-15 | 987-654-3210

Doctors Table:

DoctorID | Doctor | Specialty

1 | Dr. Johnson | Pediatrics

2 | Dr. Johnson | Neurology

3 | Dr. Adams | Cardiology

Second Normal Form (2NF):

Make sure that every non-key attribute is fully functionally dependent on the primary key. If an attribute depends on only part of a composite key, it should be moved to a separate table.

Patients Table:

PatientID | Name | DOB | Phone

1 | John Smith | 1980-05-25 | 123-456-7890

2 | Jane Doe | 1975-08-15 | 987-654-3210

Doctors Table:

DoctorID | Doctor | Specialty

1 | Dr. Johnson | Pediatrics

2 | Dr. Johnson | Neurology

3 | Dr. Adams | Cardiology

Third Normal Form (3NF):

Eliminate transitive dependencies, meaning that no non-key attribute should depend on another non-key attribute. Move such attributes to their own tables.

Patients Table:

PatientID | Name | DOB | Phone

1 | John Smith | 1980-05-25 | 123-456-7890

2 | Jane Doe | 1975-08-15 | 987-654-3210

Doctors Table:

DoctorID | Doctor

1 | Dr. Johnson

2 | Dr. Johnson

3 | Dr. Adams

Specialties Table:

DoctorID | Specialty

1 | Pediatrics

2 | Neurology

3 | Cardiology

BCNF|FOURTH Normal Form:

Depending on the complexity of your data and specific requirements, you may need to apply higher normal forms like Boyce-Codd Normal Form (BCNF) or Fourth Normal Form (4NF).

Patients Table:

PatientID | Name | DOB | Phone

1 | John Smith | 1980-05-25 | 123-456-7890

2 | Jane Doe | 1975-08-15 | 987-654-3210

Doctors Table:

DoctorID | Doctor

2 | Dr. Johnson

3 | Dr. Johnson

4 | Dr. Adams

Specialties Table:

DoctorID | Specialty

5 | Pediatrics

6 | Neurology

7 | Cardiology

FIFTH Normal Form:

5NF further decomposes tables to ensure that no join dependencies exist between them.

Patients Table:

PatientID | Name | DOB | Phone

1 | John Smith | 1980-05-25 | 123-456-7890

2 | Jane Doe | 1975-08-15 | 987-654-3210

Doctors Table:

DoctorID | Doctor

1 | Dr. Johnson

2 | Dr. Johnson

3 | Dr. Adams

Specialties Table:

DoctorID | Specialty

1 | Pediatrics

2 | Neurology

3 | Cardiology

Establish Relationships:

Define relationships between the tables using foreign keys. This ensures data integrity and facilitates querying related data across different tables.

Validate Data Model:

Validate your data model to ensure it accurately represents the real-world scenario of patient registration and meets the project requirements.

Implement in Database:

Finally, implement your normalized data model in a database management system (DBMS) of your choice, such as MySQL, PostgreSQL, or SQLite.

4 PROGRAM CODE

```
import streamlit as st

from pymongo import MongoClient

# Connect to MongoDB

client = MongoClient("mongodb://localhost:27017/")

db = client["hospital"]

patients_collection = db["patients"]

# Streamlit interface

st.title("Patient Registration Form")

# Sidebar options

st.sidebar.title("Options")

option = st.sidebar.selectbox("Choose an action", ["Add Patient", "Update Patient", "Delete Patient"])

def get_patient_by_id(patient_id):

    return patients_collection.find_one({"patient_id": patient_id})

# Form for adding a new patient

if option == "Add Patient":

    st.header("Add Patient Details")

    patient_id = st.text_input("Patient ID")

    patient_location = st.text_area("Patient Location")

    registration_date = st.text_input("Registration Date")



    name = st.text_input("Patient Name")
```

```
aadhar_number = st.text_input("Aadhar Number")

age = st.number_input("Patient Age", min_value=0, max_value=120, step=1)

gender = st.selectbox("Gender", [" ","Male", "Female", "Other"])

date_of_birth = st.text_input("Birth Date")

blood_group = st.selectbox("Blood Group", [" ","O+","O-","A+","A-","B+","B-","AB+","AB-"])

address = st.text_area("Address")

patient_contact = st.text_input("Patient Contact Number")

email = st.text_input("Patient Email")

parent_name = st.text_input("Parent/Guardian Name")

parent_contact_number = st.text_input("Parent/Guardian Number")

emergency_case = st.select_slider("Emergency?", ["Yes", "No"])

medical_history = st.text_input("Medical History")

admitted_date = st.text_input("Admitted Date")

admitted_time = st.text_input("Admitted Time")

insurance_id = st.text_input("Insurance ID")

status = st.select_slider("Status",["Alive", "Dead"])

if st.button("Add Patient"):

    if patient_id and name and patient_contact:

        # Add patient to database

        patient = {



            "patient_id": patient_id,
```

```
    "patient_location": patient_location,
    "registration_date": registration_date,
    "name": name,
    "aadhar_number": aadhar_number,
    "age": age,
    "gender": gender,
    "date_of_birth": date_of_birth,
    "blood_group": blood_group,
    "address": address,
    "patient_contact": patient_contact,
    "email": email,
    "parent_name": parent_name,
    "parent_contact_number": parent_contact_number,
    "emergency_case": emergency_case,
    "medical_history": medical_history,
    "admitted_date": admitted_date,
    "admitted_time": admitted_time,
    "insurance_id": insurance_id,
    "status": status
}



patients_collection.insert_one(patient)
```

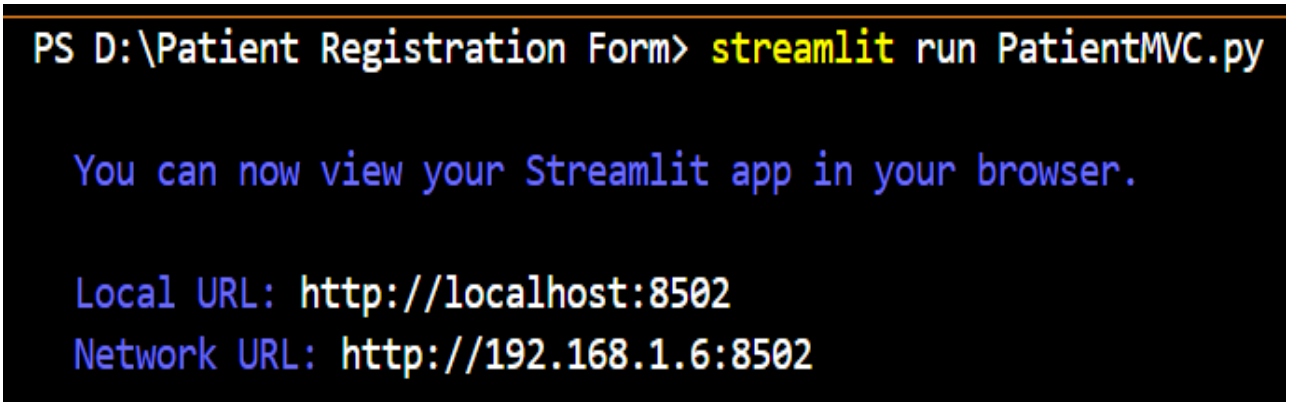
```
            st.success("Patient added successfully!")
        else:
            st.error("Please fill in all required fields.")
# Form for updating patient details
elif option == "Update Patient":
    st.header("Update Patient Details")
    update_patient_id = st.text_input("Enter Patient ID to update")
    if update_patient_id:
        patient = get_patient_by_id(update_patient_id)
        if patient:
            st.write("Patient found. Update the details below:")
            patient_location = st.text_input("Patient Location",
value=patient.get("patient_location", ""))
            registration_date = st.text_input("Registration Date",
value=patient.get("registration_id", ""))
            name = st.text_input("Name", value=patient.get("name", ""))
            aadhar_number = st.text_input("Aadhar Number",
value=patient.get("aadhar_number", ""))
            age = st.number_input("Age", min_value=0, max_value=120, step=1,
value=patient.get("age", 0))
            gender = st.selectbox("Gender", [" ","Male", "Female", "Other"], index=[" ","Male",
"Female", "Other"].index(patient.get("gender", "")))
            date_of_birth = st.text_input("Birth Date", value=patient.get("date.of.birth", ""))
```

```
        address = st.text_area("Address", value=patient.get("address", ""))

        patient_contact = st.text_input("Patient Contact Number",
value=patient.get("patient_contact", ""))

        blood_group = st.selectbox("Blood Group", [" ","O+", "O-", "B+", "B-", "AB+", "AB-", "A+",
"A-"], index=[" ","O+", "O-", "B+", "B-", "AB+", "AB-", "A+", "A-"].index(patient.get("bloodgroup",
"")))

        email = st.text_input("Patient Email", value=patient.get("email", ""))

        parent_name = st.text_input("Parent/Guardian Name",
value=patient.get("parent_name", ""))

        parent_contact_number=st.text_input("Parent/Guardian Number",
value=patient.get("parent_contact_number", ""))

        emergency_case = st.select_slider("Emergency?", ["Yes", "No"], index=["Yes",
"No"].index(patient.get("emergency_case", "")))

        medical_history = st.text_input("Medical History", value=patient.get("medical
history", ""))

        admitted_date = st.text_input("Admitted Date", value=patient.get("admitted_date",
""))

        admitted_time = st.text_input("Admitted Time", value=patient.get("admitted_time",
""))

        insurance_id = st.text_input("Insurance ID", value=patient.get("insurance_id", ""))

        status = st.select_slider("Status", ["Alive", "Dead"], index=["Alive",
"Dead"].index(patient.get("status", "")))

        if st.button("Update Patient"):

            updated_patient = {

                "patient_location": patient_location,
```

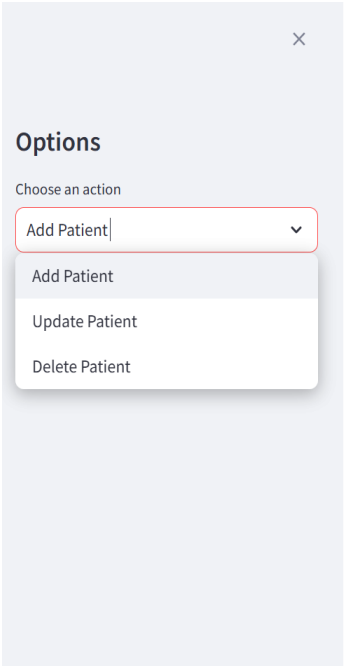
```
                "registration_date": registration_date,
                "name": name,
                "aadhar_number": aadhar_number,
                "age": age,
                "gender": gender,
                "date_of_birth": date_of_birth,
                "blood_group": blood_group,
                "address": address,
                "patient_contact": patient_contact,
                "email": email,
                "parent_name": parent_name,
                "parent_contact_number": parent_contact_number,
                "emergency_case": emergency_case,
                "medical_history": medical_history,
                "admitted_date": admitted_date,
                "admitted_time": admitted_time,
                "insurance_id": insurance_id,
                "status": status
            }
            patients_collection.update_one({"patient_id": update_patient_id}, {"$set":
updated_patient})
```

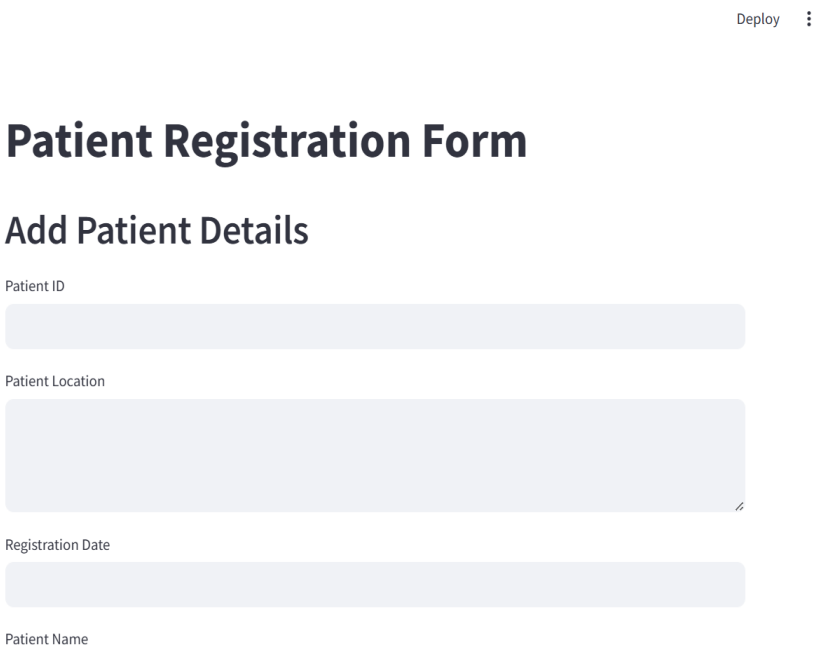
```
                st.success("Patient updated successfully!")
            else:
                st.error("No patient found with the given Patient ID.")
    # Form for deleting a patient
    elif option == "Delete Patient":
        st.header("Delete Patient")
        patient_id = st.text_input("Patient ID to delete")
        if st.button("Delete Patient"):
            if patient_id:
                result = patients_collection.delete_one({"patient_id": patient_id})
                if result.deleted_count > 0:
                    st.success("Patient deleted successfully!")
                else:
                    st.error("No patient found with the given Patient ID.")
            else:
                st.error("Please provide a valid Patient ID.")
```

5 RESULTS AND DISCUSSION



FRONT END





Add Patient

Patient added successfully!

Options

Choose an action

Update Patient

- Add Patient
- Update Patient
- Delete Patient

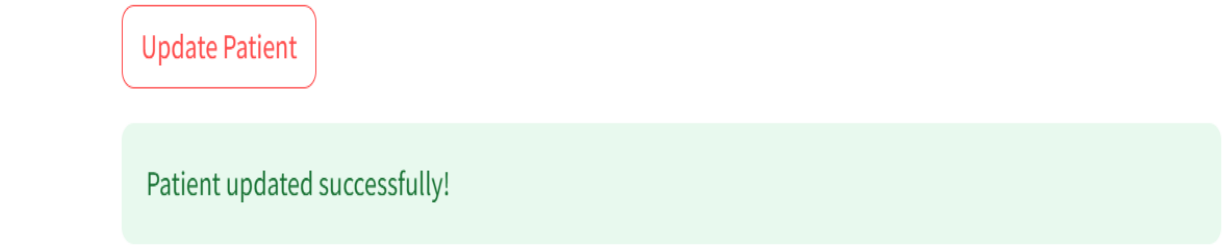
Patient Registration Form

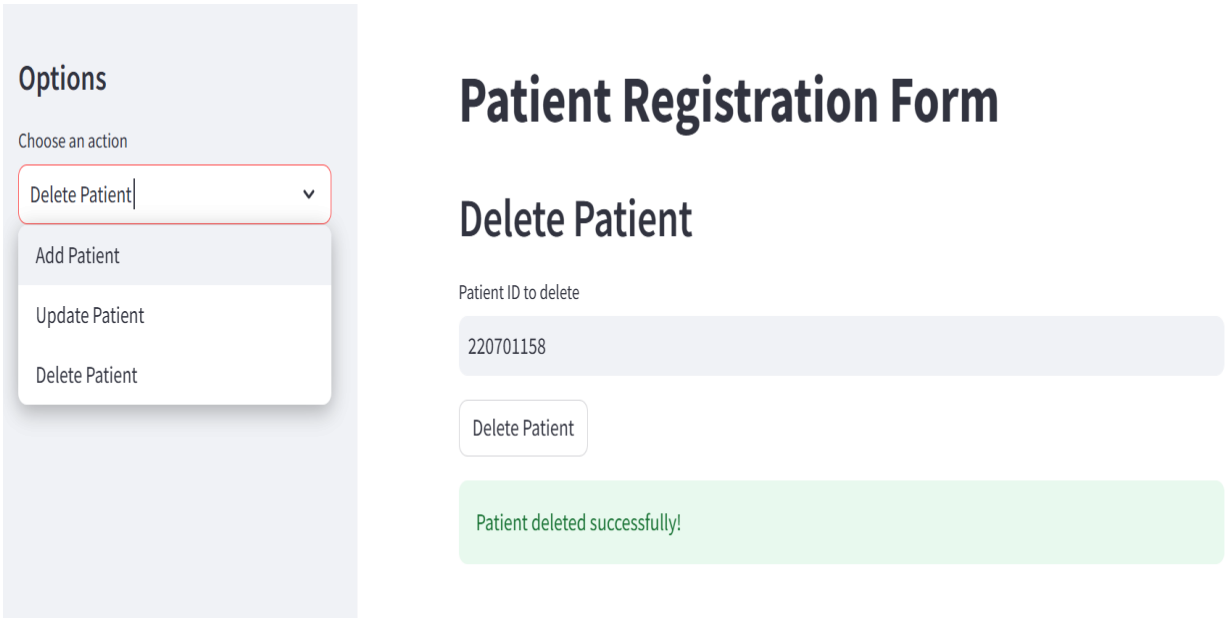
Update Patient Details

Enter Patient ID to update

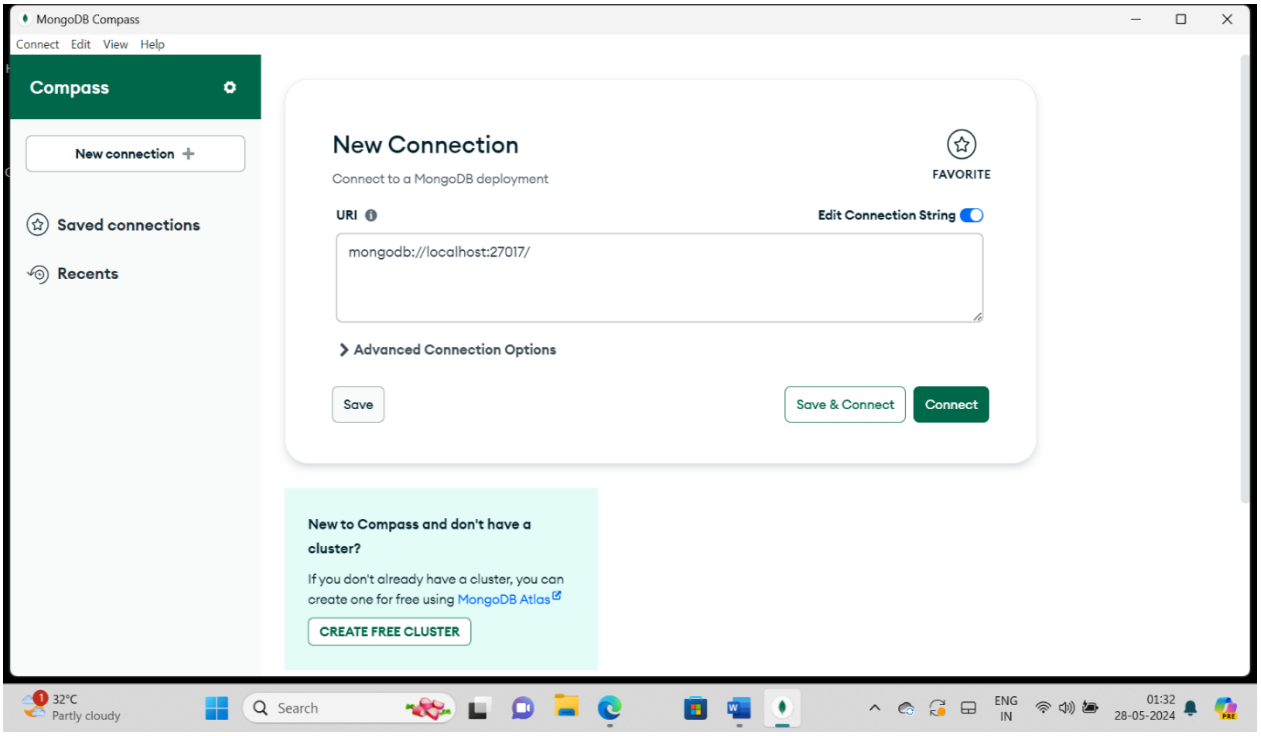
158

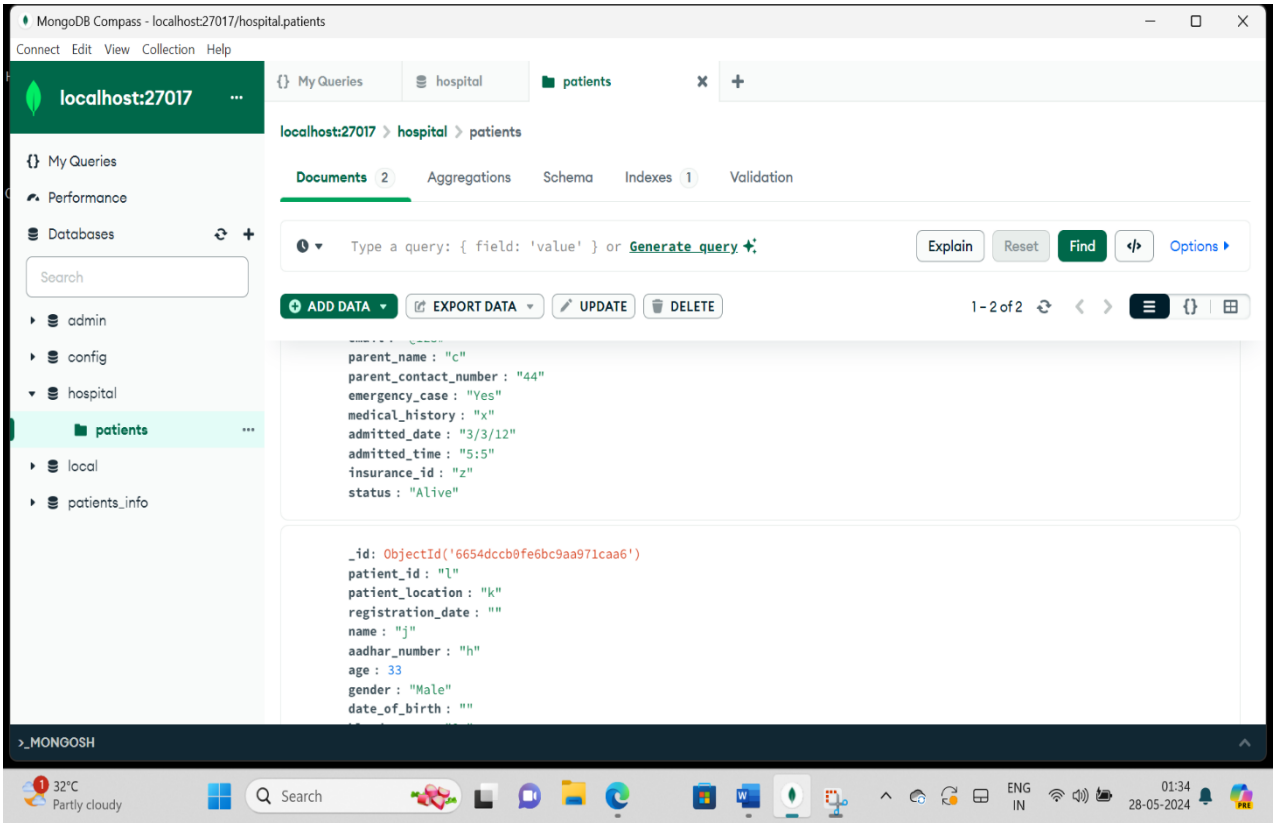
No patient found with the given Patient ID.


Update Patient
Patient updated successfully!


Options
Choose an action
Delete Patient
Add Patient
Update Patient
Delete Patient
Patient Registration Form
Delete Patient
Patient ID to delete
220701158
Delete Patient
Patient deleted successfully!

BACKEND


MongoDB Compass
Connect Edit View Help
Compass
New connection
Saved connections
Recents
New Connection
Connect to a MongoDB deployment
FAVORITE
URI
Edit Connection String
mongodb://localhost:27017/
Advanced Connection Options
Save
Save & Connect
Connect
New to Compass and don't have a cluster?
If you don't already have a cluster, you can create one for free using MongoDB Atlas
CREATE FREE CLUSTER
32°C
Partly cloudy
Search
ENG
IN
01:32
28-05-2024


MongoDB Compass - localhost:27017/hospital.patients
Connect Edit View Collection Help
localhost:27017
My Queries
Performance
Databases
Search
admin
config
hospital
patients
local
patients_info
My Queries
hospital
patients
localhost:27017 > hospital > patients
Documents 2
Aggregations
Schema
Indexes 1
Validation
Type a query: { field: 'value' } or Generate query
Explain
Reset
Find
Options
ADD DATA
EXPORT DATA
UPDATE
DELETE
1 - 2 of 2
parent_name : "c"
parent_contact_number : "44"
emergency_case : "Yes"
medical_history : "x"
admitted_date : "3/3/12"
admitted_time : "5:5"
insurance_id : "z"
status : "Alive"
_id: ObjectId('6654dccb0fe6bc9aa971caa6')
patient_id : "l"
patient_location : "k"
registration_date : ""
name : "j"
aadhar_number : "h"
age : 33
gender : "Male"
date_of_birth : ""
>_MONGOSH
32°C
Partly cloudy
Search
ENG
IN
01:34
28-05-2024

6 CONCLUSION

The patient registration mini project is a comprehensive database system designed to streamline the process of patient onboarding within a healthcare facility. Through meticulous normalization techniques, the database efficiently organizes patient information into distinct tables, ensuring data integrity and reducing redundancy.

The user-friendly interface allows patients to input their personal details and medical history securely, while validation mechanisms guarantee data accuracy. Additionally, features such as appointment scheduling, doctor allocation, and communication tools enhance patient care and streamline administrative tasks.

The Python-based mini project connected with MongoDB presents a streamlined solution for managing patient registration data within a healthcare setting. Leveraging Python's versatility and MongoDB's flexible document-oriented database, the system efficiently stores and retrieves patient information. Through a user-friendly interface, patients can input their personal details, medical history, and contact information seamlessly. The integration with MongoDB ensures data persistence and scalability, accommodating the growing volume of patient records with ease. Moreover, Python's extensive ecosystem offers powerful libraries for data validation, manipulation, and analysis, enhancing the project's functionality.

In conclusion, this mini project showcases the synergy between Python and MongoDB in developing robust and scalable database solutions for healthcare applications, facilitating efficient patient registration and management processes. Compliance with healthcare regulations, integration with existing systems, and feedback mechanisms further enrich the system's functionality. In conclusion, this mini project demonstrates the efficacy of a well-structured database in managing patient registration processes, ultimately contributing to

improved healthcare delivery and patient satisfaction.

7 REFERENCES

1. "Database Systems: Design, Implementation, & Management" by Carlos Coronel, Steven Morris, and Peter Rob.
2. "Database Management Systems" by Raghu Ramakrishnan and Johannes Gehrke.
3. "Fundamentals of Database Systems" by Ramez Elmasri and Shamkant B. Navathe.
4. "Database Design for Mere Mortals: A Hands-On Guide to Relational Database Design" by Michael J. Hernandez.
5. Online resources such as academic journals, research papers, and websites like PubMed, IEEE Xplore, and ScienceDirect can also provide valuable information and references specific to healthcare database management and patient registration systems.