HOSPITAL MANAGEMENT SYSTEM

A MINI PROJECT REPORT

SUBMITTED BY

SHANTHOSH S **220701263**

SAIVISHWARAM R **220701239**

In partial fulfilment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE

RAJALAKSHMI ENGINEERING COLLEGE (AUTONOMOUS)

THANDALAM

CHENNAI-602105

2023 – 2024

Bonafide Certificate

Certified that this project report "**HOSPITAL MANAGEMENT SYSTEM**" is a bona fide work of "**SHANTHOSH S (220701263), SAIVISHWARAM R (220701239)**" 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
Science and Engineering,
Rajalakshmi Engineering College,(Autonomous)
Chennai – 602 105

Signature
Mrs. K. Maheshmeena
Assistant Professor (SG)
Computer Science and Engineering
Rajalakshmi Engineering College(Autonomous)
Chennai – 602 105

Internal Examiner

External Examiner

Abstract

Modern healthcare facilities rely heavily on the Hospital Management System to manage numerous administrative and clinical tasks effectively. Systems that are both reliable and complete are desperately needed in order to improve patient care, optimize organizational efficiency, and streamline operations. By offering a website solution designed specifically for healthcare facilities, the hospital management system project seeks to address these issues. The main features, functions, and advantages of the HMS are highlighted in this project report, which provides a thorough overview of the system's development, implementation, and evaluation.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

1.1) INTRODUCTION

The HMS system encompasses a wide range of functionalities, including patient management, doctor and staff management, appointment scheduling, medical record management, patient admission, room managements. By integrating these core modules into a unified platform, the HMS enables healthcare providers to automate to routine tasks, reduce paper works and optimize resource utilization, ultimately leading to improved patients outcomes and operational efficiency.

An HMS facilitates the seamless coordination and integration of diverse healthcare processes, enabling healthcare providers to deliver high-quality patient care while optimizing resource utilization and operational efficiency.

1.2) OBJECTIVE

The primary goal of a Hospital Management System is to enhance the efficiency, accuracy, and quality of healthcare delivery while improving patient satisfaction and outcomes; By providing instant access to patient information, facilitating seamless communication between the healthcare providers and automating routine tasks, an HMS enables hospitals to delivery timely and effective care to patients, thereby saving efforts for both healthcare providers and patients alike. It manages all the information about doctors, nurses, appointments, admission details, room availability. This project is totally built at administrative end and this only the administrator is guaranteed the access.

1.3) MODULES

- LOGIN
- ADD NEW DOCTORS, NURSES INTO THE SYSTEM
- UPDATE DOCTORS, NURSES DATA IN THE SYSTEM
- VIEW DOCTORS, NURSES DATA
- SCHEDULE APPOINTMENTS WITH DOCTOR
- ADMIT PATIENT IN WARD ROOMS

LOGIN

Purpose:

- To securely access the hospital management system.

Functionality:

- Authentication: The admin enters the username and password.
- Password Handling: The system uses a hashed password for verification. When the admin logs in, the entered password is hashed and compared to the stored hash.
- Session Management: Upon successful login, a session is created for the admin. Sessions have a timeout to automatically log the admin out after a period of inactivity to enhance security.
- Error Handling: Provides feedback for incorrect login attempts and will lock the account after a certain number of failed attempts.

MANAGE DOCTORS AND NURSES DATA

Purpose:

- To efficiently handle all CRUD (Create, Read, Update, Delete) operations related to medical staff (doctors and nurses).

Functionality:

- Data Entry (Create):
    - Purpose: To register new medical staff in the system.
    - Details Collected: Name, contact information, qualifications, specialization, department, etc.(ERD Diagram at section 3)
    - Role Assignment: Assign specific roles (doctor, nurse) and their permissions within the system.
    - Verification: Assumed to be performed externally before data entry.
- View Data (Read):
    - Purpose: To access and review detailed information about medical staff.
    - Functionality: Search and filter staff by various criteria such as department, role, name, etc.
    - Detailed View: Provides comprehensive details of each staff member.
    - Reports: Generate and export reports for analysis and record-keeping.

- Update Data (Update):
    - Purpose: To keep staff information accurate and up-to-date.
    - Functionality: Modify personal details, professional qualifications, and departmental assignments.
    - Track Changes: Maintain a log of all modifications for accountability and auditing.
- Delete Data:
    - Purpose: To remove staff members who are no longer part of the hospital.
    - Functionality: Allow the admin to delete a staff member's data from the system when necessary, ensuring that historical data can still be archived or retained for compliance purposes if needed.

CHAPTER 2: SURVEY OF TECHNOLOGY

2.1) SOFTWARE DESCRIPTION

Visual Studio Code:

Using VS code for building the HMS offer a seamless development experience with its lightweight yet powerful editor, extensive language support, and a rich ecosystem of extension, Its integrated features like IntelliSense, debugging tools, and version control integration streamline the development process, enabling efficient coding and collaboration. JS is used for building the frontend components of the application, allowing for the creation of a user-friendly interface that facilitates seamless interactions with the system.

2.2) PROGRAMMING LANGUAGES

2.2.1) JavaScript

It is a versatile programming language widely used for web development. It is a high level interpreted programming language that is primarily used for client side scripting in web development. JavaScript enables developers to add interactivity to web pages by manipulating DOM (Document Object Model) elements, handling events such as user inputs and clicks, and making asynchronous requests to fetch data from servers.

2.2.2) Python

Python is a high-level programming language known for its simplicity, readability and versatility. In the context of the backend development for the hospital management system, Python is utilized for implementing server-side logic, handling HTTP request and responses, and interacting with the database. This makes it easy to write, understand and maintain cod, leading to increased productivity and reduced development time.

2.3) FRAMEWORKS / MODULES USED

2.3.1) FRAMEWORKS USED WITH JAVASCRIPT

Next.js

Next.js is a React Framework that offers server-side rendering(SSR), static site generation(SSG), and client-side rendering(CSR), providing flexibility and performance optimization for web application. It simplifies development by offering built-in routing, code splitting and hot module replacement. It simplifies the process of building complex websites with development experience with features like automatic code splitting, server-side rendering and static site generation along with its intuitive API makes it a popular choice for building modern web application

React with Material-UI

React is a JavaScript Library for building user interfaces, known for its component-based architecture and efficient rendering. Material-UI is a React UI framework that provides pre-designed and customizable UI components following Google Material Designs, offering a sleek and modern user interface. React simplifies the process of building interactive user interfaced by breaking down the UI into reusable components, making development more efficient and maintainable.

2.3.2) MODULES USED IN PYTHON

Flask

It is a flexible web framework for python that provide essential tools and libraries for building web application. It simplifies the process of handling HTTP request, routing and template rendering, allowing to focus on implementing application logic rather than boilerplate code. Flask serves as the backend framework, handling incoming requests from the frontend interface and generating appropriate responses. It enables the implementation of APIs for managing resources such as patients, doctors, appointments and medical records, hence adding extra layer of security for accessing the data

CHAPTER 3: REQUIREMENTS AND ANALYSIS

3.1) REQUIREMENT SPECIFICATIONS

User Requirements:

- The hospital admin wants to efficiently manage patient records, appointments and medical histories while providing seamless communication between healthcare providers and patients.
- Streamline administrative tasks such as doctor and staff management, room controls and appointments scheduling to optimize resources utilization and enhance operational efficiency.

System Requirements:

- A database backup along with an operating system with a web browser to access the hospital management system.

3.2) HARDWARE SPECIFICATIONS

Software Requirements:

- Frontend - React.js with Material UI
- Backend – MongoDB with Python

System Requirements:

- Operating Systems: Windows 7 or later, macOS 10.12 or later, Ubuntu 16.04 or later
- Web browser: Google Chrome version 88 or later, Mozilla Firefox version 85 or later, Safari version 14 or later, Microsoft Edge version 88 or later
- Internet Connections: Stable broadband internet connection
- Hardware: Desktop with at least 2GB RAM and dual-core processor, laptop with similar specifications.
    - 64-bit OS to utilize the full capabilities of the hardware
    - 1024 x 768 resolution monitor for optimal display of application
- Authentication Credentials: Username and password of the administrator account.

3.3) Functional / Non- Functional Requirements

Login System: Secure authentication for admin access with redirection to a dashboard displaying an overview of schedules.

Staff Management: CRUD operations for doctors and nurses, including search functionality, accessible via a dedicated staff page.

Patient Management: CRUD operations for patients with search functionality, accessible via a separate patient page.

Appointment Scheduling: Admin can schedule appointments with specific doctors for patients, including options for rescheduling or cancelling appointments.

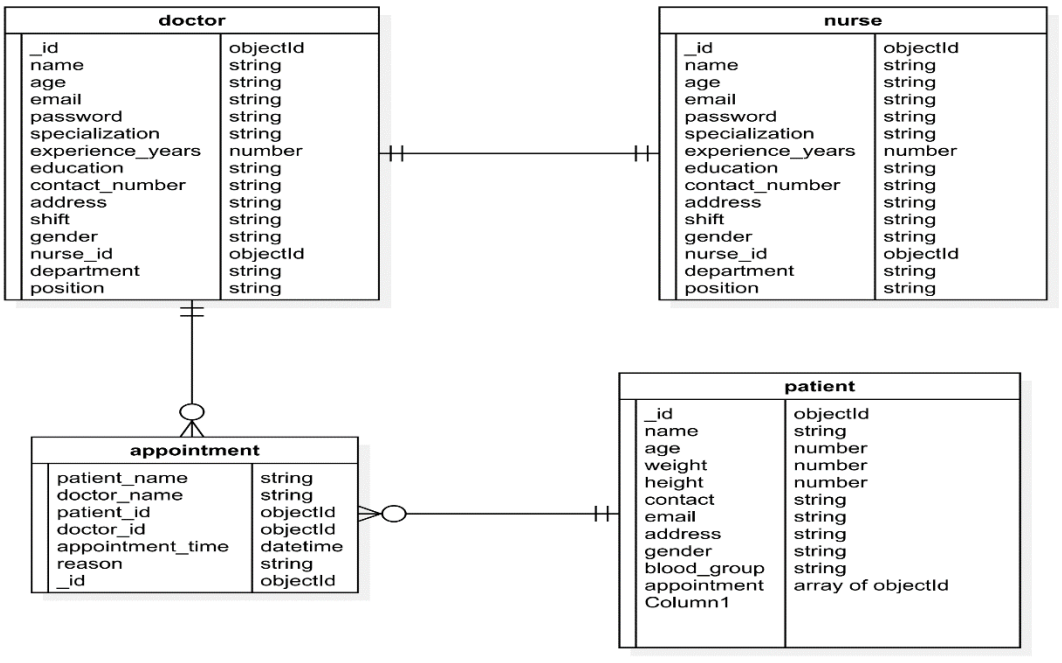
Scalability: The system should be able to accommodate an increasing number of users, patients, and staff members without significant degradation in performance.

Usability: The user interface should be intuitive and user-friendly, allowing admin users to navigate and perform tasks efficiently with minimal training.

Reliability: The system should have high uptime and minimal downtime to ensure continuous availability for users.

Performance: The system should be responsive and able to handle concurrent user requests efficiently, ensuring optimal performance even during peak usage periods.

3.4: ER Diagram



CHAPTER 4: Program Code

##patient.py

```
from flask import Blueprint, request, jsonify
from functions import convert_id, build_query, connect_to_database, convert_objectid


patient_bp = Blueprint('patient', __name__)
db = connect_to_database()


# Routes for Patients
@patient_bp.route('/', methods=['GET'])
def get_patients():
    try:
        patients = list(db.patient.find())
        patients_with_appointments = []

        for patient in patients:
            patient_id = str(patient['_id'])
            appointments = list(db.appointment.find({'patient_id': patient_id}))

            if appointments:
                patient['appointments'] = [convert_objectid(app) for app in appointments]
            else:
                patient['appointments'] = 'N/A'
            patient = convert_id(patient)
            patients_with_appointments.append(patient)

        return jsonify({'status': 'success', 'patients': patients_with_appointments})
    except Exception as e:
```

```
        return jsonify({'status': 'error', 'message': str(e)}), 500


@patient_bp.route('/add', methods=['POST'])
def add_patient():
    try:
        new_patient_data = request.json
        result = db.patient.insert_one(new_patient_data)
        inserted_id = str(result.inserted_id)
        new_patient_data['_id'] = inserted_id
        return jsonify({'status': 'success', 'message': 'Patient added', 'patient': new_patient_data})
    except Exception as e:
        return jsonify({'status': 'error', 'message': str(e)}), 500


@patient_bp.route('/get_patient_info', methods=['POST'])
def get_patient_info():
    try:
        request_data = request.json
        patient_name = request_data.get('name')
        patient_id = request_data.get('_id')
        if not patient_name and not patient_id:
            return jsonify({'status': 'error', 'message': 'Patient name or ObjectId not provided'}), 400
        query = build_query(name=patient_name, objectId=patient_id, email=None)
        patient = db.patient.find_one(query)
        if patient:
            patient = convert_id(patient)
            return jsonify({'status': 'success', 'patient': patient})
        else:
            return jsonify({'status': 'error', 'message': 'Patient not found'}), 404
```

```
    except Exception as e:
        return jsonify({'status': 'error', 'message': str(e)}), 500


@patient_bp.route('/update', methods=['PUT'])
def update_patient():
    try:
        updated_data = request.json
        patient_name = updated_data.get('name')
        patient_id = updated_data.get('_id')
        if not patient_name and not patient_id:
            return jsonify({'status': 'error', 'message': 'Patient name or ObjectId not provided'}), 400
        query = build_query(name=patient_name, objectId=patient_id, email=None)
        db.patient.update_one(query, {"$set": updated_data})
        return jsonify({'status': 'success', 'message': 'Patient updated'})
    except Exception as e:
        return jsonify({'status': 'error', 'message': str(e)}), 500


@patient_bp.route('/delete', methods=['DELETE'])
def delete_patient():
    try:
        request_data = request.json
        patient_name = request_data.get('name')
        patient_id = request_data.get('_id')
        if not patient_name and not patient_id:
            return jsonify({'status': 'error', 'message': 'Patient name or ObjectId not provided'}), 400
        query = build_query(name=patient_name, objectId=patient_id, email=None)
        result = db.patient.delete_one(query)
        if result.deleted_count == 1:
```

```
            return jsonify({'status': 'success', 'message': 'Patient deleted'})
        else:
            return jsonify({'status': 'error', 'message': 'Patient not found'}), 404
    except Exception as e:
        return jsonify({'status': 'error', 'message': str(e)}), 500
```

##doctor.py

```
from flask import Blueprint, request, jsonify
from functions import convert_id, build_query, connect_to_database
from werkzeug.security import check_password_hash

doctor_bp = Blueprint('doctor', __name__)
db = connect_to_database()

# Routes for Doctors
@doctor_bp.route('/', methods=['GET'])
def get_doctors():
    try:
        doctors = list(db.doctor.find())
        doctors = [convert_id(doctor) for doctor in doctors]
        return jsonify({'status': 'success', 'doctors': doctors})
    except Exception as e:
        return jsonify({'status': 'error', 'message': str(e)}), 500

@doctor_bp.route('/add', methods=['POST'])
def add_doctor():
    try:
        new_doctor_data = request.json
```

```
        result = db.doctor.insert_one(new_doctor_data)
        inserted_id = str(result.inserted_id)
        new_doctor_data['_id'] = inserted_id
        return jsonify({'status': 'success', 'message': 'Doctor added', 'doctor': new_doctor_data})
    except Exception as e:
        return jsonify({'status': 'error', 'message': str(e)}), 500
@doctor_bp.route('/login', methods=['POST'])
def doctor_login():
    try:
        request_data = request.json
        email = request_data.get('email')
        password = request_data.get('password')
        doctor = db.doctor.find_one({"email": email})
        if password == doctor["password"]:
            doctor = convert_id(doctor)
            return jsonify({'status': 'success', 'message': 'Doctor logged in', 'doctor': doctor}), 200
        else:
            return jsonify({'status': 'error', 'message': 'Invalid email or password'}), 401
    except Exception as e:
        return jsonify({'status': 'error', 'message': str(e)}), 500


@doctor_bp.route('/get_doctor_info', methods=['POST'])
def get_doctor_info():
    try:
        request_data = request.json
        doctor_name = request_data.get('name')
        doctor_id = request_data.get('objectId')
```

```
        doctor_email = request_data.get('email')
        if not doctor_name and not doctor_id and not doctor_email:
            return jsonify({'status': 'error', 'message': 'Doctor name or ObjectId or email not provided'}), 400
        query = build_query(name=doctor_name, objectId=doctor_id, email=doctor_email)
        doctor = db.doctor.find_one(query)
        if doctor:
            doctor = convert_id(doctor)
            return jsonify({'status': 'success', 'doctor': doctor})
        else:
            return jsonify({'status': 'error', 'message': 'Doctor not found'}), 404
    except Exception as e:
        return jsonify({'status': 'error', 'message': str(e)}), 500

@doctor_bp.route('/update', methods=['PUT'])
def update_doctor():
    try:
        updated_data = request.json
        doctor_name = updated_data.get('name')
        doctor_id = updated_data.get('_id')
        if not doctor_name and not doctor_id:
            return jsonify({'status': 'error', 'message': 'Doctor name or ObjectId not provided'}), 400
        query = build_query(name=doctor_name, objectId=doctor_id, email=None)
        db.doctor.update_one(query, {"$set": updated_data})
        return jsonify({'status': 'success', 'message': 'Doctor updated'})
    except Exception as e:
        return jsonify({'status': 'error', 'message': str(e)}), 500
```

```
@doctor_bp.route('/delete', methods=['DELETE'])
def delete_doctor():
    try:
        request_data = request.json
        doctor_name = request_data.get('name')
        doctor_id = request_data.get('objectId')
        doctor_email = request_data.get('email')
        if not doctor_name and not doctor_id and not doctor_email:
            return jsonify({'status': 'error', 'message': 'Doctor name or ObjectId or email not provided'}), 400
        query = build_query(name=doctor_name, objectId=doctor_id, email=doctor_email)
        result = db.doctor.delete_one(query)
        if result.deleted_count == 1:
            return jsonify({'status': 'success', 'message': 'Doctor deleted'})
        else:
            return jsonify({'status': 'error', 'message': 'Doctor not found'}), 404
    except Exception as e:
        return jsonify({'status': 'error', 'message': str(e)}), 500


@doctor_bp.route('/assign_nurse', methods=['PUT'])
def assign_nurse():
    try:
        request_data = request.json
        doctor_name = request_data.get('name')
        doctor_id = request_data.get('objectId')
        nurse_name = request_data.get('nurseName')
        nurse_email = request_data.get('nurseEmail')
```

```
        if not doctor_name and not doctor_id:
            return jsonify({'status': 'error', 'message': 'Doctor name or ObjectId not provided'}), 400
        if not nurse_name and not nurse_email:
            return jsonify({'status': 'error', 'message': 'Nurse name or email not provided'}), 400
        nurse_query = {}
        if nurse_name:
            nurse_query['name'] = nurse_name
        if nurse_email:
            nurse_query['email'] = nurse_email
        nurse = db.nurse.find_one(nurse_query)

        if nurse:
            nurse_id = str(nurse['_id'])
            query = build_query(name=doctor_name, objectId=doctor_id, email=None)
            db.doctor.update_one(query, {"$set": {"nurse_id": nurse_id}})
            return jsonify({'status': 'success', 'message': 'Nurse assigned to doctor'})
        else:
            return jsonify({'status': 'error', 'message': 'Nurse not found'}), 404
    except Exception as e:
        return jsonify({'status': 'error', 'message': str(e)}), 500
```

##nurse.py

```
from flask import Blueprint, request, jsonify
from functions import convert_id, build_query, connect_to_database

nurse_bp = Blueprint('nurse', __name__)
db = connect_to_database()
```

```
# Routes for Nurses
@nurse_bp.route('/', methods=['GET'])
def get_nurses():
    try:
        nurses = list(db.nurse.find())
        nurses = [convert_id(nurse) for nurse in nurses]
        return jsonify({'status': 'success', 'nurses': nurses})
    except Exception as e:
        return jsonify({'status': 'error', 'message': str(e)}), 500



@nurse_bp.route('/add', methods=['POST'])
def add_nurse():
    try:
        new_nurse_data = request.json
        result = db.nurse.insert_one(new_nurse_data)
        inserted_id = str(result.inserted_id)
        new_nurse_data['_id'] = inserted_id
        return jsonify({'status': 'success', 'message': 'Nurse added', 'nurse': new_nurse_data})
    except Exception as e:
        return jsonify({'status': 'error', 'message': str(e)}), 500


@nurse_bp.route('/get_nurse_info', methods=['POST'])
def get_nurse_info():
    try:
        request_data = request.json
        nurse_name = request_data.get('name')
        nurse_id = request_data.get('objectId')
```

```
        nurse_email = request_data.get('email')
        if not nurse_name and not nurse_id and not nurse_email:
            return jsonify({'status': 'error', 'message': 'Nurse name or ObjectId or email not provided'}), 400
        query = build_query(name=nurse_name, objectId=nurse_id, email=nurse_email)
        nurse = db.nurse.find_one(query)
        if nurse:
            nurse = convert_id(nurse)
            return jsonify({'status': 'success', 'nurse': nurse})
        else:
            return jsonify({'status': 'error', 'message': 'Nurse not found'}), 404
    except Exception as e:
        return jsonify({'status': 'error', 'message': str(e)}), 500
@nurse_bp.route('/update', methods=['PUT'])
def update_nurse():
    try:
        updated_data = request.json
        nurse_name = updated_data.get('name')
        nurse_id = updated_data.get('objectId')
        if not nurse_name and not nurse_id:
            return jsonify({'status': 'error', 'message': 'Nurse name or ObjectId not provided'}), 400
        query = build_query(name=nurse_name, objectId=nurse_id, email=None)
        db.nurse.update_one(query, {"$set": updated_data})
        return jsonify({'status': 'success', 'message': 'Nurse updated'})
    except Exception as e:
        return jsonify({'status': 'error', 'message': str(e)}), 500


@nurse_bp.route('/delete', methods=['DELETE'])
```

```
def delete_nurse():
    try:
        request_data = request.json
        nurse_name = request_data.get('name')
        nurse_id = request_data.get('objectId')
        nurse_email = request_data.get('email')
        if not nurse_name and not nurse_id and not nurse_email:
            return jsonify({'status': 'error', 'message': 'Nurse name or ObjectId or email not provided'}), 400
        query = build_query(name=nurse_name, objectId=nurse_id, email=nurse_email)
        result = db.nurse.delete_one(query)
        if result.deleted_count == 1:
            return jsonify({'status': 'success', 'message': 'Nurse deleted'})
        else:
            return jsonify({'status': 'error', 'message': 'Nurse not found'}), 404
    except Exception as e:
        return jsonify({'status': 'error', 'message': str(e)}), 500
```

##appointment.py

```
from flask import Blueprint, request, jsonify
from datetime import datetime
from bson.objectid import ObjectId
from functions import connect_to_database, check_appointment_collision, convert_objectid, convert_id

appointment_bp = Blueprint('appointment', __name__)
db = connect_to_database()

def update_appointment_with_ids(appointment_data):
```

```
    patient_name = appointment_data.get('patient_name')
    doctor_name = appointment_data.get('doctor_name')
    pipeline = [
        {"$match": {"name": {"$in": [patient_name, doctor_name]}}},
        {"$group": {"_id": "$name", "id": {"$first": "$_id"}}}
    ]
    patient = list(db.patient.aggregate(pipeline))
    doctor = list(db.doctor.aggregate(pipeline))
    patient_id = next((p['id'] for p in patient if p['_id'] == patient_name), None)
    doctor_id = next((d['id'] for d in doctor if d['_id'] == doctor_name), None)

    appointment_data['patient_id'] = patient_id
    appointment_data['doctor_id'] = doctor_id



# Routes for Appointments
@appointment_bp.route('/add', methods=['POST'])
def add_appointment():
    try:
        appointment_data = request.json
        doctor_id = appointment_data.get('doctor_id')
        appointment_time_str = appointment_data.get('appointment_time')
        appointment_time = datetime.strptime(appointment_time_str, "%Y-%m-%d %H:%M:%S")
        appointment_data['appointment_time'] = appointment_time
        if check_appointment_collision(doctor_id=doctor_id, appointment_time=appointment_time,
db=db):
            return jsonify(
                {'status': 'error', 'message': 'Appointment collides with existing appointment for the
doctor'}), 400
```

```
        update_appointment_with_ids(appointment_data)
        result = db.appointment.insert_one(appointment_data)
        appointment_data['_id'] = str(result.inserted_id)

        appointment_data['doctorName'] = appointment_data.pop('doctor_name', None)
        appointment_data = convert_objectid(appointment_data)

        return jsonify({'status': 'success', 'message': 'Appointment added', 'appointment': appointment_data})
    except Exception as e:
        return jsonify({'status': 'error', 'message': str(e)}), 500

@appointment_bp.route('/update', methods=['PUT'])
def update_appointment():
    try:
        appointment_data = request.json
        appointment_id = appointment_data.get('appointment_id')
        appointment_time_str = appointment_data.get('appointment_time')
        appointment_time = datetime.strptime(appointment_time_str, "%Y-%m-%d %H:%M:%S")
        appointment_data['appointment_time'] = appointment_time
        doctor_id = appointment_data.get('doctor_id')
        if check_appointment_collision(doctor_id=doctor_id, appointment_time=appointment_time, db=db):
            return jsonify(
                {'status': 'error', 'message': 'Appointment collides with existing appointment for the doctor'}), 400
        update_appointment_with_ids(appointment_data)
```

```
        db.appointment.update_one({"_id": ObjectId(appointment_id)}, {"$set": {"appointment_time": appointment_time}})
        appointment_data['_id'] = appointment_id
        appointment_data = convert_objectid(appointment_data)

        return jsonify({'status': 'success', 'message': 'Appointment time updated', 'appointment': appointment_data})
    except Exception as e:
        return jsonify({'status': 'error', 'message': str(e)}), 500

@appointment_bp.route('/', methods=['GET'])
def get_all_appointments():
    try:
        appointments = list(db.appointment.find())
        appointments = [convert_objectid(app) for app in appointments]
        return jsonify({'status': 'success', 'appointments': appointments})
    except Exception as e:
        return jsonify({'status': 'error', 'message': str(e)}), 500
```

##functions.py

```
from bson import ObjectId
from pymongo import MongoClient
from dotenv import load_dotenv
from datetime import datetime, timedelta
import os

load_dotenv()

def convert_id(data):
```

```
    if isinstance(data, list):
        for item in data:
            item['_id'] = str(item['_id'])
    else:
        data['_id'] = str(data['_id'])
    return data

# to build query dynamically
def build_query(name, objectId, email):
    query = {}
    if name:
        query['name'] = name
    if objectId:
        query['_id'] = ObjectId(objectId)
    if email:
        query['email'] = email
    return query

def connect_to_database():
    mongo_uri = os.getenv("MONGO_URI")
    client = MongoClient(mongo_uri)
    db = client['HospitalManagement']
    return db

def check_appointment_collision(doctor_id, appointment_time, db):
    time_before = appointment_time - timedelta(minutes=10)
    time_after = appointment_time + timedelta(minutes=10)
    existing_appointment = db.appointment.find_one({
```

```
        "doctor_id": doctor_id,
        "appointment_time": {"$gte": time_before, "$lte": time_after}
    })
    return existing_appointment is not None

def convert_objectid(data):
    if isinstance(data, dict):
        return {key: convert_objectid(value) for key, value in data.items()}
    elif isinstance(data, list):
        return [convert_objectid(item) for item in data]
    elif isinstance(data, ObjectId):
        return str(data)
    else:
        return data
```

##extra.py

```
from flask import Blueprint, jsonify, request
from functions import ObjectId, connect_to_database

extras_bp = Blueprint('extras', __name__)
db = connect_to_database()

def find_doctor_by_name(doctor_name):
    doctor = db.doctor.find_one({"name": doctor_name})
    return doctor

def find_nurse_by_id(nurse_id):
    nurse = db.nurse.find_one({"_id": ObjectId(nurse_id)})
```

```
    return nurse


@extras_bp.route('/find_nurse_from_doctor', methods=['POST'])
def find_doctor_and_nurse():
    try:
        request_data = request.json
        doctor_name = request_data.get('doctorName')
        if not doctor_name:
            return jsonify({'status': 'error', 'message': 'Doctor name not provided'}), 400

        doctor = find_doctor_by_name(doctor_name)
        if doctor:
            nurse_id = doctor.get('nurse_id')
            if nurse_id:
                nurse = find_nurse_by_id(nurse_id)
                if nurse:
                    doctor['_id'] = str(doctor['_id'])
                    nurse['_id'] = str(nurse['_id'])
                    return jsonify({'status': 'success', 'doctor': doctor, 'assigned_nurse': nurse})
                else:
                    return jsonify({'status': 'error', 'message': 'Nurse not found'}), 404
            else:
                return jsonify({'status': 'error', 'message': 'No nurse assigned to this doctor'}), 404
        else:
            return jsonify({'status': 'error', 'message': 'Doctor not found'}), 404
    except Exception as e:
        return jsonify({'status': 'error', 'message': str(e)}), 500
```

##app.py

```
from flask import Flask
from patient import patient_bp
from doctor import doctor_bp
from nurse import nurse_bp
from extra import extras_bp
from appointment import appointment_bp
from departments import department_bp
from flask_cors import CORS

app = Flask(__name__)
CORS(app)

app.register_blueprint(patient_bp, url_prefix='/patients')
app.register_blueprint(doctor_bp, url_prefix='/doctor')
app.register_blueprint(nurse_bp, url_prefix='/nurse')
app.register_blueprint(extras_bp, url_prefix='/extra')
app.register_blueprint(appointment_bp, url_prefix='/appointment')
app.register_blueprint(department_bp, url_prefix='/department')

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

//LoginPage.js

```
import React, {useEffect, useState} from 'react';
import {Button, Checkbox, FormControlLabel, TextField} from '@mui/material';
import {Email, Lock, Visibility, VisibilityOff} from '@mui/icons-material';
import Link from 'next/link';
```

```
import {useSnackbar} from 'notistack';
import {useRouter} from 'next/router';
import hospitalLoginImage from '../assets/hospital_login.jpg';
const LoginPage = () => {
 const [email, setEmail] = useState('');
 const [password, setPassword] = useState('');
 const [showPassword, setShowPassword] = useState(false);
 const [rememberMe, setRememberMe] = useState(false);
 const { enqueueSnackbar } = useSnackbar();
 const router = useRouter();
 useEffect(() => {
  const savedEmail = localStorage.getItem('rememberedEmail');
  if (savedEmail) {
   setEmail(savedEmail);
   setRememberMe(true);
  }
 }, []);
 const handleEmailChange = (event) => {
  setEmail(event.target.value);
 };
 const handlePasswordChange = (event) => {
  setPassword(event.target.value);
 };

 const handleShowPassword = () => {
  setShowPassword(!showPassword);
 };
 const handleRememberMeChange = (event) => {
```

```
    setRememberMe(event.target.checked);
  };
  const handleSubmit = (event) => {
    event.preventDefault();
    if (email === 'admin@rajalakshmi.edu.in' && password === 'admin') {
      enqueueSnackbar('Login successful', { variant: 'success', autoHideDuration: 5000 });
      if (rememberMe) {
        localStorage.setItem('rememberedEmail', email);
      } else {
        localStorage.removeItem('rememberedEmail');
      }
      setTimeout(() => {
        router.push('/Dashboard');
      }, 1000);
    } else {
      enqueueSnackbar('Invalid email or password', { variant: 'error', autoHideDuration: 5000 });
    }
  };

  return (
    <div className="min-h-screen flex items-center justify-center" style={{backgroundImage: `url(${hospitalLoginImage.src})`, backgroundSize: 'cover', backgroundPosition: 'center'}}>
      <div className="w-full max-w-md p-8 bg-white bg-opacity-60 rounded-lg shadow-md font-inter">
        <h1 className="text-2xl font-bold text-center mb-8">Log In</h1>
        <form onSubmit={handleSubmit}>
          <div className="mb-6">
            <TextField
              fullWidth
```

```
    label="Email"
    variant="outlined"
    value={email}
    onChange={handleEmailChange}
    required
    type="email"
    InputProps={{
     startAdornment: (
      <div className="pr-2">
       <Email className="text-gray-500" />
      </div>
     )
    }}
   />
  </div>
  <div className="mb-6 relative">
   <TextField
    fullWidth
    label="Password"
    variant="outlined"
    value={password}
    onChange={handlePasswordChange}
    required
    type={showPassword ? 'text' : 'password'}
    InputProps={{
     startAdornment: (
      <div className="pr-2">
       <Lock className="text-gray-500" />
```

```
        </div>
      ),
      endAdornment: (
        <Button onClick={handleShowPassword} className="focus:outline-none">
          {showPassword ? <VisibilityOff /> : <Visibility />}
        </Button>
      ),
    }}
   />
  </div>
  <FormControlLabel
   control={
    <Checkbox
     color="primary"
     checked={rememberMe}
     onChange={handleRememberMeChange}
    />
   }
   label="Remember me"
   className="mb-6"
  />
  <Button variant="contained" type="submit" color="primary" fullWidth>
   Log In
  </Button>
 </form>
 <Link href="/ForgotPasswordPage">
  <span className="text-blue-500 block mt-4 text-center hover:underline">
   Forgot password?
```

```
        </span>
       </Link>
     </div>
   </div>
  );
};

export default LoginPage;
```

//Dashbord.js

```
import React from 'react';
import Layout from '../components/Layout';
import StatCard from '../components/StatCard';
import Departments from '../components/Departments';
import Schedule from '../components/Schedule';
import {Box, Typography} from '@mui/material';
import {AccountCircleOutlined, EventOutlined, HotelOutlined, LocalHospitalOutlined,} from
'@mui/icons-material';

const Dashboard = () => {
 // Mock data for the statistics
 const stats = [
  { title: 'Total Patients', value: 500, icon: <AccountCircleOutlined /> },
  { title: 'Total Appointments', value: 750, icon: <EventOutlined />},
  { title: 'Total Surgeries', value: 100, icon: <LocalHospitalOutlined /> },
  { title: 'Total Wards', value: 20, icon: <HotelOutlined /> },
 ];
```

```
 return (
  <Layout>
   <Box sx={{ flexGrow: 1, ml: '16rem', overflowY: 'auto', p: 3, justifyContent: 'center'}}>
    <Typography variant="h6" gutterBottom>
     Dashboard
    </Typography>
    <div className="flex flex-wrap justify-center">
     {stats.map((stat, index) => (
      <StatCard key={index} title={stat.title} value={stat.value} icon={stat.icon} />
     ))}
    </div>
    <div className="flex">
     <div style={{ width: '70%', padding: '20px' }}>
      <Departments />
     </div>
     <div style={{ width: '50%', padding: '20px' }}>
      <Schedule />
     </div>
    </div>
   </Box>
  </Layout>
 );
};

export default Dashboard;
```

//patient.js

```
import React, { useEffect, useState } from 'react';
```

```
import { Box, Button, IconButton, InputAdornment, Paper, Table, TableBody, TableCell,
TableContainer, TableHead, TableRow, TableSortLabel, TextField, Typography, FormControl,
InputLabel, MenuItem, Select} from '@mui/material';

import { Add, SearchOutlined, Info } from '@mui/icons-material';

import Layout from '../components/Layout';

import Heading from '../components/Heading';

import NewPatientModal from '../components/NewPatientModal';

import PatientDetailModal from '../components/PatientDetailModal';

import { useDataContext } from '../context/DataContext';


const Patient = () => {
  const { patients, setPatients } = useDataContext();
  const [searchQuery, setSearchQuery] = useState('');
  const [filteredPatients, setFilteredPatients] = useState([]);
  const [openModal, setOpenModal] = useState(false);
  const [openDetailModal, setOpenDetailModal] = useState(false);
  const [selectedPatient, setSelectedPatient] = useState(null);
  const [newPatientDetails, setNewPatientDetails] = useState({
    name: '', age: '', weight: '', height: '', contact: '',
    email: '', address: '', gender: '', bloodGroup: '', appointments: ''
  });
  const [order, setOrder] = useState('asc');
  const [orderBy, setOrderBy] = useState('name');

  useEffect(() => {
    const filterPatients = () => {
      const lowerCaseQuery = searchQuery.toLowerCase();
      const filtered = patients.filter(
        (patient) =>
```

```
      patient.name.toLowerCase().includes(lowerCaseQuery) ||
      patient.id.toLowerCase().includes(lowerCaseQuery)
    );
    setFilteredPatients(filtered);
  };
  filterPatients();
}, [searchQuery, patients]);
const handleSearchChange = (e) => {
  setSearchQuery(e.target.value);
};
const handleOpenModal = () => {
  setOpenModal(true);
};

const handleCloseModal = () => {
  setOpenModal(false);
  resetForm();
};
const handleOpenDetailModal = (patient) => {
  setSelectedPatient(patient);
  setOpenDetailModal(true);
};
const handleCloseDetailModal = () => {
  setOpenDetailModal(false);
  setSelectedPatient(null);
};
const resetForm = () => {
  setNewPatientDetails({
```

```
    name: '', age: '', weight: '', height: '', contact: '',
    email: '', address: '', gender: '', bloodGroup: '', appointments: ''
  });
};
const handleNewPatientChange = (e) => {
  const { name, value } = e.target;
  setNewPatientDetails({ ...newPatientDetails, [name]: value });
};
const handleNewPatientSubmit = async () => {
  try {
    await addPatient(newPatientDetails);
    handleCloseModal();
  } catch (error) {
    console.error('Failed to add patient:', error);
  }
};
const handleDeletePatient = (patientId) => {
  setPatients((prev) => prev.filter((patient) => patient.id !== patientId));
};
const handleUpdatePatient = (updatedPatient) => {
  if (!updatedPatient) return;
  setPatients((prev) =>
    prev.map((patient) => (patient.id === updatedPatient.id ? updatedPatient : patient))
  );
};
const handleRequestSort = (property) => {
  const isAsc = orderBy === property && order === 'asc';
  setOrder(isAsc ? 'desc' : 'asc');
```

```
  setOrderBy(property);
};
const sortComparator = (a, b, orderBy) => {
  if (a[orderBy] < b[orderBy]) {
    return -1;
  }
  if (a[orderBy] > b[orderBy]) {
    return 1;
  }
  return 0;
};
const sortedPatients = filteredPatients.sort((a, b) => (
  order === 'asc' ? sortComparator(a, b, orderBy) : sortComparator(b, a, orderBy)
));
return (
  <Layout>
    <Box sx={{ flexGrow: 1, ml: '16rem', overflowY: 'auto', maxHeight: '100vh-10px', p: 3 }}>
      <Heading text="Patient Management" />
      <Box sx={{ mb: 3 }}>
        <TextField
          label="Search Patients"
          variant="outlined"
          fullWidth
          value={searchQuery}
          onChange={handleSearchChange}
          InputProps={{
            endAdornment: (
              <InputAdornment position="end">
```

```
            <IconButton>
              <SearchOutlined />
            </IconButton>
          </InputAdornment>
        ),
      }}
    />
  </Box>
  <Box sx={{ display: 'flex', justifyContent: 'space-between', alignItems: 'center', mb: 3 }}>
    <Typography variant="h6">Patient List</Typography>
    <Button variant="contained" color="primary" startIcon={<Add />}
onClick={handleOpenModal}>
      Add New Patient
    </Button>
  </Box>
  <TableContainer component={Paper}>
    <Table>
      <TableHead>
        <TableRow>
          <TableCell sortDirection={orderBy === 'id' ? order : false}>
            <TableSortLabel
              active={orderBy === 'id'}
              direction={orderBy === 'id' ? order : 'asc'}
              onClick={() => handleRequestSort('id')}
            >
              Patient ID
            </TableSortLabel>
          </TableCell>
```

```
    <TableCell sortDirection={orderBy === 'name' ? order : false}>
     <TableSortLabel
      active={orderBy === 'name'}
      direction={orderBy === 'name' ? order : 'asc'}
      onClick={() => handleRequestSort('name')}
     >
      Name
     </TableSortLabel>
    </TableCell>
    <TableCell sortDirection={orderBy === 'age' ? order : false}>
     <TableSortLabel
      active={orderBy === 'age'}
      direction={orderBy === 'age' ? order : 'asc'}
      onClick={() => handleRequestSort('age')}
     >
      Age
     </TableSortLabel>
    </TableCell>
    <TableCell>Contact</TableCell>
    <TableCell>Appointments</TableCell>
    <TableCell>Actions</TableCell>
   </TableRow>
  </TableHead>
  <TableBody>
   {sortedPatients.map((patient) => (
    <TableRow key={patient.id}>
     <TableCell>{patient.id}</TableCell>
     <TableCell>{patient.name}</TableCell>
```

```
            <TableCell>{patient.age}</TableCell>
            <TableCell>{patient.contact}</TableCell>
            <TableCell>{patient.appointments}</TableCell>
            <TableCell>
              <Button variant="outlined" startIcon={<Info />} onClick={() =>
handleOpenDetailModal(patient)}>
                Show More
              </Button>
            </TableCell>
          </TableRow>
        ))}
      </TableBody>
    </Table>
  </TableContainer>
</Box>

<NewPatientModal
  open={openModal}
  onClose={handleCloseModal}
  newPatientDetails={newPatientDetails}
  onChange={handleNewPatientChange}
  onSubmit={handleNewPatientSubmit}
  />
   <PatientDetailModal
    open={openDetailModal}
    onClose={handleCloseDetailModal}
    selectedPatient={selectedPatient}
    onDelete={handleDeletePatient}
```

```
        onUpdate={handleUpdatePatient}
      />
    </Layout>
  );
};

export default Patient;
```

//Staff.js

```
import React, { useEffect, useState } from 'react';
import {
Box, Button, IconButton, InputAdornment, Paper, Table, TableBody, TableCell, TableContainer, TableHead, TableRow, TableSortLabel, TextField, Typography, Select, MenuItem, FormControl, InputLabel} from '@mui/material';
import { Add, Info, SearchOutlined } from '@mui/icons-material';
import Layout from '../components/Layout';
import Heading from '../components/Heading';
import NewStaffModal from '../components/NewStaffModal';
import StaffDetailsModal from '../components/StaffDetailsModal';
import { useDataContext } from '../context/DataContext';

const Staff = () => {
  const { doctors, nurses, setDoctors } = useDataContext();  // Assuming `addStaff` is a function in context to add new staff
  const [searchQuery, setSearchQuery] = useState('');
  const [filteredStaff, setFilteredStaff] = useState([]);
  const [selectedStaffType, setSelectedStaffType] = useState('doctors');
  const [openModal, setOpenModal] = useState(false);
  const [openDetailModal, setOpenDetailModal] = useState(false);
```

```
  const [selectedStaff, setSelectedStaff] = useState(null);
  const [newStaffDetails, setNewStaffDetails] = useState({
    name: '', position: '', department: '', contact: '',
    email: '', address: '', gender: '', experience: '',
    shift: '', assignedDoctor: ''
  });
  const [order, setOrder] = useState('asc');
  const [orderBy, setOrderBy] = useState('name');

  useEffect(() => {
    const filterStaff = () => {
      const lowerCaseQuery = searchQuery.toLowerCase();
      const staff = selectedStaffType === 'doctors' ? doctors : nurses;
      const filtered = staff.filter(
        (s) => s.name.toLowerCase().includes(lowerCaseQuery) ||
s.id.toLowerCase().includes(lowerCaseQuery)
      );
      setFilteredStaff(filtered);
    };
    filterStaff();
  }, [searchQuery, selectedStaffType, doctors, nurses]);

  const handleSearchChange = (e) => {
    setSearchQuery(e.target.value);
  };

  const handleStaffTypeChange = (e) => {
    setSelectedStaffType(e.target.value);
```

```
};

const handleOpenModal = () => {
  setOpenModal(true);
};

const handleCloseModal = () => {
  setOpenModal(false);
  resetForm();
};

const handleOpenDetailModal = (staffMember) => {
  setSelectedStaff(staffMember);
  setOpenDetailModal(true);
};

const handleCloseDetailModal = () => {
  setOpenDetailModal(false);
  setSelectedStaff(null);
};

const resetForm = () => {
  setNewStaffDetails({
    name: '',
    position: '',
    department: '',
    contact: '',
    email: '',
```

```
    address: '',
    gender: '',
    experience: '',
    shift: '',
    assignedDoctor: '',
  });
};

const handleNewStaffChange = (e) => {
  const { name, value } = e.target;
  setNewStaffDetails({ ...newStaffDetails, [name]: value });
};

const handleNewStaffSubmit = async () => {
  try {
    await addStaff(newStaffDetails);
    handleCloseModal();
  } catch (error) {
    console.error('Failed to add staff:', error);
  }
};

const handleDeleteStaff = (staffId) => {
  setFilteredStaff((prev) => prev.filter((staff) => staff.id !== staffId));
};

const handleUpdateStaff = (updatedStaff) => {
  if (!updatedStaff) return;
```

```
  if (selectedStaffType === 'doctors') {
    setDoctors((prev) =>
      prev.map((doctor) => (doctor.id === updatedStaff.id ? updatedStaff : doctor))
    );
  } else {
    setNurses((prev) =>
      prev.map((nurse) => (nurse.id === updatedStaff.id ? updatedStaff : nurse))
    );
  }
};

const handleRequestSort = (property) => {
  const isAsc = orderBy === property && order === 'asc';
  setOrder(isAsc ? 'desc' : 'asc');
  setOrderBy(property);
};

const sortComparator = (a, b, orderBy) => {
  if (a[orderBy] < b[orderBy]) {
    return -1;
  }
  if (a[orderBy] > b[orderBy]) {
    return 1;
  }
  return 0;
};
```

```
const sortedStaff = filteredStaff.sort((a, b) => (
  order === 'asc' ? sortComparator(a, b, orderBy) : sortComparator(b, a, orderBy)
));



return (
  <Layout>
    <Box sx={{ flexGrow: 1, ml: '16rem', overflowY: 'auto', maxHeight: '100vh-10px', p: 3 }}>
      <Heading text="Staff Management" />
      <Box sx={{ mb: 3 }}>
        <TextField
          label="Search Staff"
          variant="outlined"
          fullWidth
          value={searchQuery}
          onChange={handleSearchChange}
          InputProps={{
            endAdornment: (
              <InputAdornment position="end">
                <IconButton>
                  <SearchOutlined />
                </IconButton>
              </InputAdornment>
            ),
          }}
        />
      </Box>
      <Box sx={{ display: 'flex', justifyContent: 'space-between', alignItems: 'center', mb: 3 }}>
```

```
        <Typography variant="h6">Staff List</Typography>
        <Button variant="contained" color="primary" startIcon={<Add />}
onClick={handleOpenModal}>
          Add New Staff
        </Button>
      </Box>
      <Box sx={{ display: 'flex', justifyContent: 'space-between', alignItems: 'center', mb: 3 }}>
        <FormControl variant="outlined" sx={{ minWidth: 200 }}>
          <InputLabel id="select-staff-type-label">Select Staff Type</InputLabel>
          <Select
            labelId="select-staff-type-label"
            value={selectedStaffType}
            onChange={handleStaffTypeChange}
            label="Select Staff Type"
          >
            <MenuItem value="doctors">Doctors</MenuItem>
            <MenuItem value="nurses">Nurses</MenuItem>
          </Select>
        </FormControl>
      </Box>
      <TableContainer component={Paper}>
        <Table>
          <TableHead>
            <TableRow>
              <TableCell sortDirection={orderBy === 'id' ? order : false}>
                <TableSortLabel
                  active={orderBy === 'id'}
                  direction={orderBy === 'id' ? order : 'asc'}
```

```
    onClick={() => handleRequestSort('id')}
   >
    Staff ID
   </TableSortLabel>
  </TableCell>
  <TableCell sortDirection={orderBy === 'name' ? order : false}>
   <TableSortLabel
    active={orderBy === 'name'}
    direction={orderBy === 'name' ? order : 'asc'}
    onClick={() => handleRequestSort('name')}
   >
    Name
   </TableSortLabel>
  </TableCell>
  <TableCell sortDirection={orderBy === 'position' ? order : false}>
   <TableSortLabel
    active={orderBy === 'position'}
    direction={orderBy === 'position' ? order : 'asc'}
    onClick={() => handleRequestSort('position')}
   >
    Position
   </TableSortLabel>
  </TableCell>
  <TableCell>Specialization</TableCell>
  <TableCell sortDirection={orderBy === 'department' ? order : false}>
   <TableSortLabel
    active={orderBy === 'department'}
    direction={orderBy === 'department' ? order : 'asc'}
```

```
                onClick={() => handleRequestSort('department')}
              >
                Department
              </TableSortLabel>
            </TableCell>
            <TableCell>Contact</TableCell>
            <TableCell>Actions</TableCell>
          </TableRow>
        </TableHead>
        <TableBody>
          {sortedStaff.map((staffMember) => (
            <TableRow key={staffMember.id}>
              <TableCell>{staffMember.id}</TableCell>
              <TableCell>{staffMember.name}</TableCell>
              <TableCell>{staffMember.position}</TableCell>
              <TableCell>{staffMember.specialization}</TableCell>
              <TableCell>{staffMember.department}</TableCell>
              <TableCell>{staffMember.contact}</TableCell>
              <TableCell>
                <Button variant="outlined" startIcon={<Info />} onClick={() =>
handleOpenDetailModal(staffMember)}>
                  Show More
                </Button>
              </TableCell>
            </TableRow>
          ))}
        </TableBody>
      </Table>
```

```
      </TableContainer>
    </Box>

    <NewStaffModal
      open={openModal}
      onClose={handleCloseModal}
      newStaffDetails={newStaffDetails}
      onChange={handleNewStaffChange}
      onSubmit={handleNewStaffSubmit}
    />
    <StaffDetailsModal
      open={openDetailModal}
      onClose={handleCloseDetailModal}
      selectedStaff={selectedStaff}
      onDelete={handleDeleteStaff}
      onUpdate={handleUpdateStaff}
    />
  </Layout>
 );
};
export default Staff;
```

//Appointment scheduling.js

```
import React, { useEffect, useState } from 'react';
import {  Box, Button, Divider, FormControl, Grid, IconButton, InputAdornment, InputLabel, MenuItem, Modal, Select, Stack, TextField, Typograph } from '@mui/material';
import { Add, EventAvailable, SearchOutlined } from '@mui/icons-material';
import Layout from '../components/Layout';
```

```
import AppointmentTable from '../components/AppointmentTable';
import Heading from '../components/Heading';
import { useDataContext } from '../context/DataContext';
import { useSnackbar } from 'notistack';
import { format } from 'date-fns'; // Import date-fns

const AppointmentScheduling = () => {
 const { doctors, patients, appointments, setAppointments } = useDataContext();
 const [searchQuery, setSearchQuery] = useState('');
 const [filteredAppointments, setFilteredAppointments] = useState([]);
 const [openModal, setOpenModal] = useState(false);
 const [patientDetails, setPatientDetails] = useState({ id: '', name: '', contact: '' });
 const [doctorDetails, setDoctorDetails] = useState({ id: '', name: '', department: '' });
 const [appointmentDateTime, setAppointmentDateTime] = useState('');
 const [reason, setReason] = useState('');
 const { enqueueSnackbar } = useSnackbar();

 useEffect(() => {
  setFilteredAppointments(appointments);
 }, [appointments]);

 useEffect(() => {
  const filterAppointments = () => {
   const lowerCaseQuery = searchQuery.toLowerCase();
   const filtered = appointments.filter((appointment) => {
    const doctorName = appointment.doctorName?.toLowerCase() || '';
    const patientName = appointment.patientName?.toLowerCase() || '';
    return doctorName.includes(lowerCaseQuery) || patientName.includes(lowerCaseQuery);
```

```
    });
    setFilteredAppointments(filtered);
  };
  filterAppointments();
}, [searchQuery, appointments]);
const handleSearchChange = (e) => {
  setSearchQuery(e.target.value);
};
const handleOpenModal = () => {
  setOpenModal(true);
};
const handleCloseModal = () => {
  setOpenModal(false);
  resetForm();
};
const resetForm = () => {
  setPatientDetails({ id: '', name: '', contact: '' });
  setDoctorDetails({ id: '', name: '', department: '' });
  setReason('');
  setAppointmentDateTime('');
};

const handlePatientChange = (e) => {
  const { name, value } = e.target;
  setPatientDetails({ ...patientDetails, [name]: value });

  if (name === 'id') {
    const patient = patients.find((p) => p.id === value);
```

```
    if (patient) {
      setPatientDetails(patient);
    }
  } else if (name === 'name') {
    const patient = patients.find((p) => p.name === value);
    if (patient) {
      setPatientDetails(patient);
    }
  }
};
const handleDoctorChange = (e) => {
  const { name, value } = e.target;
  setDoctorDetails({ ...doctorDetails, [name]: value });
  if (name === 'id') {
    const doctor = doctors.find((d) => d.id === value);
    if (doctor) {
      setDoctorDetails(doctor);
    }
  } else if (name === 'department') {
    const doctor = doctors.find((d) => d.department === value);
    if (doctor) {
      setDoctorDetails(doctor);
    }
  }
};
const handleAppointmentDateTimeChange = (e) => {
  setAppointmentDateTime(e.target.value);
};
```

```
  const handleAppointmentSubmit = async () => {
    try {
      const formattedDateTime = format(new Date(appointmentDateTime), 'yyyy-MM-dd
HH:mm:ss');
      const appointmentData = {
        patientId: patientDetails.id,
        patientName: patientDetails.name,
        contact: patientDetails.contact,
        department: doctorDetails.department,
        doctorName: doctorDetails.name,
        appointment_time: formattedDateTime,
        reason: reason
      };
      console.log('Appointment data:', appointmentData); // Added for troubleshooting

      const response = await fetch('http://127.0.0.1:5000/appointment/add', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json'
        },
        body: JSON.stringify(appointmentData)
      });
      console.log('Response:', response); // Added for troubleshooting
      if (!response.ok) {
        throw new Error('Failed to submit appointment');
      }

      const newAppointment = await response.json();
```

```
    console.log('New Appointment:', newAppointment); // Added for troubleshooting

    setAppointments([...appointments, newAppointment]);
    enqueueSnackbar('Appointment scheduled successfully', { variant: 'success' });
    handleCloseModal();
  } catch (error) {
    console.error('Error submitting appointment:', error.message);
    enqueueSnackbar('Failed to schedule appointment', { variant: 'error' });
  }
};

return (
  <Layout>
    <Box sx={{ flexGrow: 1, ml: '16rem', overflowY: 'auto', p: 3 }}>
      <Heading text="Appointments" />
      <Box sx={{ mb: 3 }}>
        <TextField
          label="Search Appointments"
          variant="outlined"
          fullWidth
          value={searchQuery}
          onChange={handleSearchChange}
          InputProps={{
            endAdornment: (
              <InputAdornment position="end">
                <IconButton>
                  <SearchOutlined />
                </IconButton>
```

```
            </InputAdornment>
          ),
        }}
      />
    </Box>
    <Box sx={{ display: 'flex', justifyContent: 'space-between', alignItems: 'center', mb: 3 }}>
      <Typography variant="h6">Todays Appointments</Typography>
      <Button variant="contained" color="primary" startIcon={<Add />} onClick={handleOpenModal}>
        Create Appointment
      </Button>
    </Box>
    {filteredAppointments.length ? (
      <AppointmentTable appointments={filteredAppointments} />
    ) : (
      <Typography>No appointments scheduled for today! Have a great day!!!</Typography>
    )}
  </Box>

  <Modal open={openModal} onClose={handleCloseModal}>
    <Box sx={{ width: '600px', p: 4, bgcolor: 'background.paper', m: '50px auto', borderRadius: 2, boxShadow: 24, overflowY: 'auto', maxHeight: '80vh' }}>
      <Stack spacing={3}>
        <Box sx={{ display: 'flex', justifyContent: 'space-between', alignItems: 'center' }}>
          <Typography variant="h6">Create Appointment</Typography>
          <Button variant="contained" color="primary" onClick={handleAppointmentSubmit} startIcon={<EventAvailable />}>
            Confirm Appointment
          </Button>
```

```
</Box>
<Divider />
<Grid container spacing={2}>
 <Grid item xs={6}>
  <TextField
   label="Patient ID"
   variant="outlined"
   name="id"
   value={patientDetails.id}
   onChange={handlePatientChange}
   fullWidth
  />
 </Grid>
 <Grid item xs={6}>
  <TextField
   label="Patient Name"
   variant="outlined"
   name="name"
   value={patientDetails.name}
   onChange={handlePatientChange}
   fullWidth
  />
 </Grid>
</Grid>
<TextField
 label="Contact"
 variant="outlined"
 name="contact"
```

```
 value={patientDetails.contact}
 fullWidth
 disabled
/>
<Grid container spacing={2}>
 <Grid item xs={6}>
  <FormControl fullWidth variant="outlined">
   <InputLabel>Department</InputLabel>
   <Select
    name="department"
    value={doctorDetails.department}
    onChange={handleDoctorChange}
    label="Department"
   >
    {Array.from(new Set(doctors.map(doc => doc.department))).map(dept => (
     <MenuItem key={dept} value={dept}>{dept}</MenuItem>
    ))}
   </Select>
  </FormControl>
 </Grid>
 <Grid item xs={6}>
  <FormControl fullWidth variant="outlined">
   <InputLabel>Doctor</InputLabel>
   <Select
    name="id"
    value={doctorDetails.id}
    onChange={handleDoctorChange}
    label="Doctor"
```

```
        >
          {doctors
            .filter(doc => doc.department === doctorDetails.department)
            .map(doc => (
              <MenuItem key={doc.id} value={doc.id}>{doc.name}</MenuItem>
            ))}
        </Select>
      </FormControl>
    </Grid>
  </Grid>
  <TextField
    label="Date and Time"
    variant="outlined"
    type="datetime-local"
    name="appointmentDateTime"
    value={appointmentDateTime}
    onChange={handleAppointmentDateTimeChange}
    fullWidth
  />
  <TextField
    label="Reason for Appointment"
    variant="outlined"
    name="reason"
    value={reason}
    onChange={(e) => setReason(e.target.value)}
    fullWidth
  />
</Stack>
```

```
      </Box>
    </Modal>
  </Layout>
 );
};
export default AppointmentScheduling
```

//index.js

```
import LoginPage from './LoginPage';
export default LoginPage
```

//staffDetailsModal.jsx

```
import React, { useEffect, useState } from 'react';
import { Box, Button, Divider, IconButton, List, ListItem, ListIcon, ListItemText, Modal, Stack, TextField, Typography} from '@mui/material';
import { Business, Email, Event, Home, Person, School, Wc, Work, Edit, Delete, Cake, BarChart, Fingerprint, LocalHospital, PhoneAndroid } from '@mui/icons-material';
import PropTypes from 'prop-types';
import { useSnackbar } from 'notistack';

const StaffDetailsModal = ({ open, onClose, selectedStaff, onDelete, onUpdate }) => {
 const { enqueueSnackbar } = useSnackbar();
 const [isEditing, setIsEditing] = useState(false);
 const [staffDetails, setStaffDetails] = useState(selectedStaff || {});
 useEffect(() => {
  setStaffDetails(selectedStaff || {});
 }, [selectedStaff]);
 const handleEditClick = () => {
  setIsEditing(true);
 };
```

```
const handleInputChange = (e) => {
  const { name, value } = e.target;
  setStaffDetails({ ...staffDetails, [name]: value });
};

const handleSaveClick = async () => {
  const endpoint = selectedStaff.position === 'Doctor' ? '/doctor/update' : '/nurse/update';
  const payload = {
    _id: staffDetails._id,
    name: staffDetails.name,
    email: staffDetails.email,
    ...staffDetails
  };
  try {
    const response = await fetch(`http://127.0.0.1:5000${endpoint}`, {
      method: 'PUT',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify(payload),
    });
    const result = await response.json();
    if (response.ok && result.status === 'success') {
      enqueueSnackbar('Details updated successfully', { variant: 'success' });
      onUpdate(staffDetails);
      setIsEditing(false);
    } else {
      enqueueSnackbar(`Error: ${result.message}`, { variant: 'error' });
```

```
    }
  } catch (error) {
    enqueueSnackbar('Error updating staff', { variant: 'error' });
    console.error('Error updating staff:', error);
  }
};

const handleDeleteClick = async () => {
  const endpoint = selectedStaff.position === 'Doctor' ? '/doctor/delete' : '/nurse/delete';
  const payload = {
    _id: staffDetails._id,
    name: staffDetails.name,
    email: staffDetails.email
  };
  try {
    const response = await fetch(`http://127.0.0.1:5000${endpoint}`, {
      method: 'DELETE',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify(payload),
    });
    const result = await response.json();
    if (result.status === 'success') {
      enqueueSnackbar('Staff deleted successfully', { variant: 'success' });
      onDelete(selectedStaff._id);
      onClose();
    } else {
```

```
      enqueueSnackbar(`Error: ${result.message}`, { variant: 'error' });
    }
   } catch (error) {
    enqueueSnackbar('Error deleting staff', { variant: 'error' });
    console.error('Error deleting staff:', error);
   }
  };

  return (
   <Modal open={open} onClose={onClose}>
    <Box sx={{ width: '400px', p: 4, bgcolor: 'background.paper', m: '50px auto', borderRadius: 2, boxShadow: 24, overflowY: 'auto', maxHeight: '80vh' }}>
     {selectedStaff && (
      <Stack spacing={2}>
       <Box sx={{ display: 'flex', justifyContent: 'space-between', alignItems: 'center' }}>
        <Typography variant="h6">Staff Details</Typography>
        {!isEditing && (
         <IconButton onClick={handleEditClick}>
          <Edit />
         </IconButton>
        )}
       </Box>
       <Divider />
       <List>
        {Object.entries(staffDetails).map(([key, value]) => (
         <ListItem key={key}>
          <ListItemIcon>
           {key === 'name' && <Person />}
```

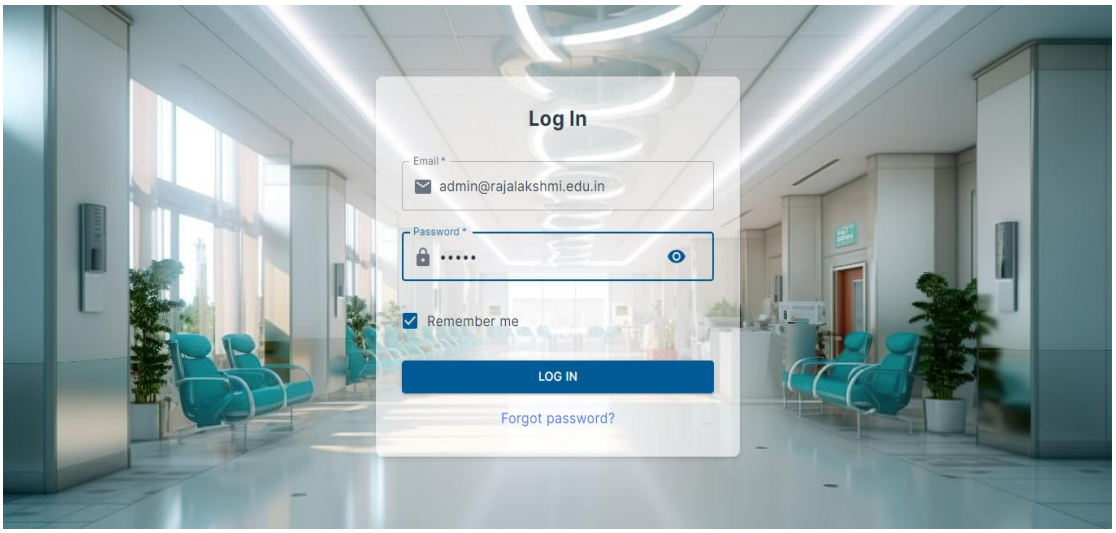
```
  {key === 'age' && <Cake />}
  {key === 'contact' && <PhoneAndroid/>}
  {key === 'email' && <Email />}
  {key === 'address' && <Home />}
  {key === 'experience' && <BarChart />}
  {key === 'education' && <School />}
  {key === 'shift' && <Event />}
  {key === 'gender' && <Wc />}
  {key === 'department' && <Business />}
  {key === 'specialization' && <LocalHospital />}
  {key === 'id' && <Fingerprint />}
  {key === 'position' && <Work />}
 </ListItemIcon>
 <ListItemText
  primary={
   isEditing ? (
    <TextField
     fullWidth
     name={key}
     value={value}
     onChange={handleInputChange}
    />
   ) : (
    `${key.charAt(0).toUpperCase() + key.slice(1)}: ${value}`
   )
  }
 />
</ListItem>
```

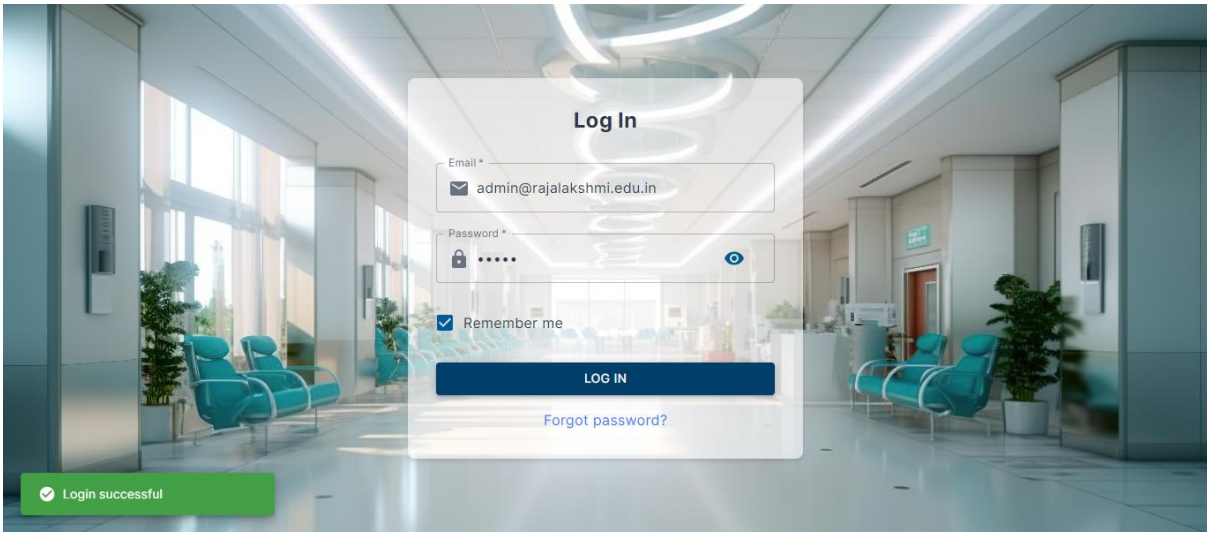
```
          ))}
        </List>
        <Box sx={{ display: 'flex', justifyContent: 'space-between' }}>
          {isEditing && (
            <Button variant="contained" onClick={handleSaveClick}>
              Save
            </Button>
          )}
          <Button variant="contained" color="error" onClick={handleDeleteClick}>
            Delete
          </Button>
          <Button variant="contained" onClick={onClose}>Close</Button>
        </Box>
      </Stack>
    )}
  </Box>
 </Modal>
 );
};

StaffDetailsModal.propTypes = {
 open: PropTypes.bool.isRequired,
 onClose: PropTypes.func.isRequired,
 selectedStaff: PropTypes.object,
 onDelete: PropTypes.func.isRequired,
 onUpdate: PropTypes.func.isRequired,
};
```

```
StaffDetailsModal.defaultProps = {
  selectedStaff: null,
};

export default StaffDetailsModal;

// Layout.js
import React from 'react';
import Sidebar from './Sidebar';

const Layout = ({ children }) => {
  return (
    <div style={{ display: 'flex' }}>
      <Sidebar />
      <div style={{ flex: 1 }}>
        {children}
      </div>
    </div>
  );
};

export default Layout;
```

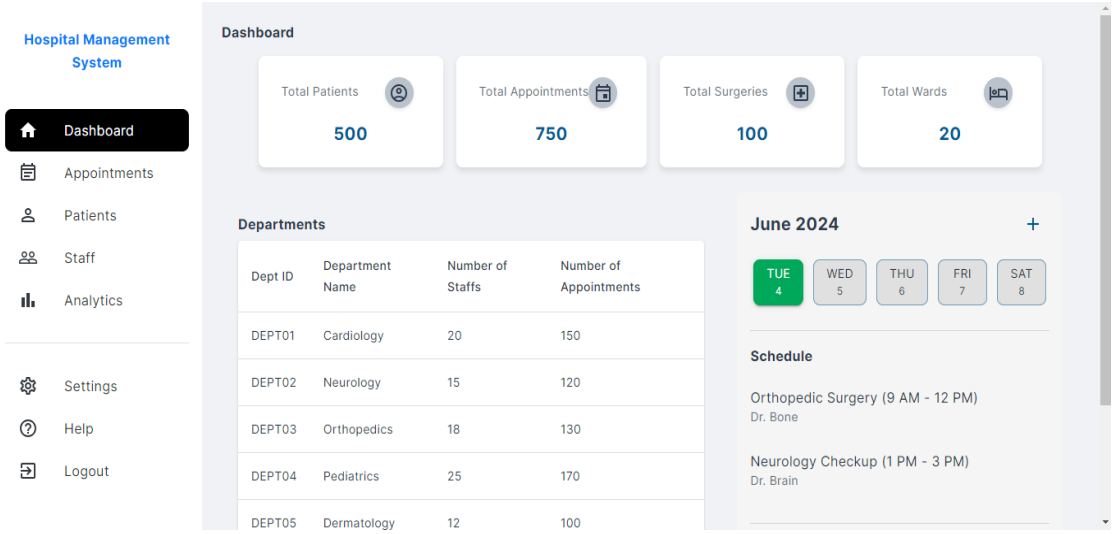
CHAPTER 5: RESULT

1.) Login interface

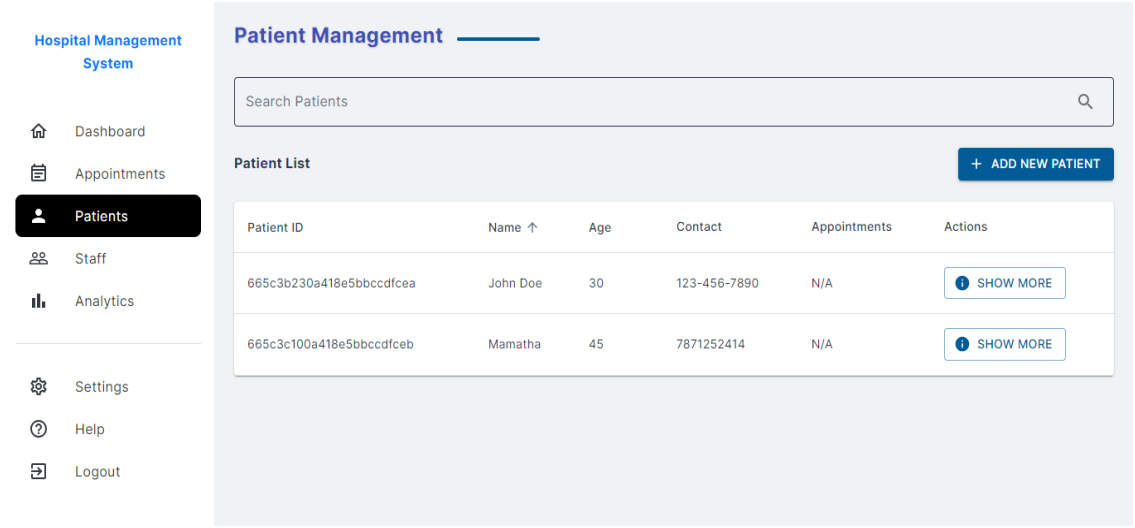


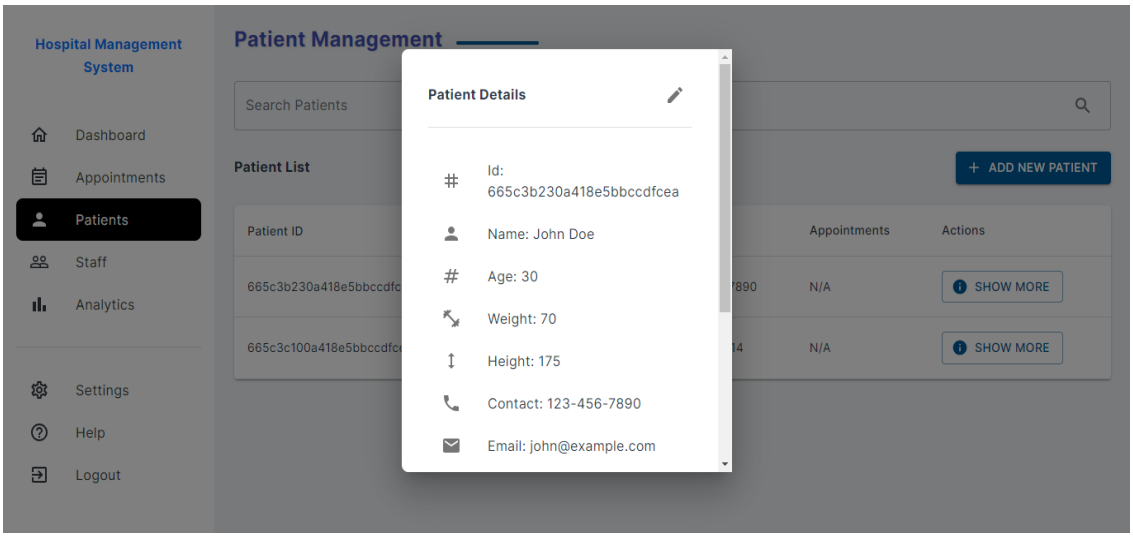


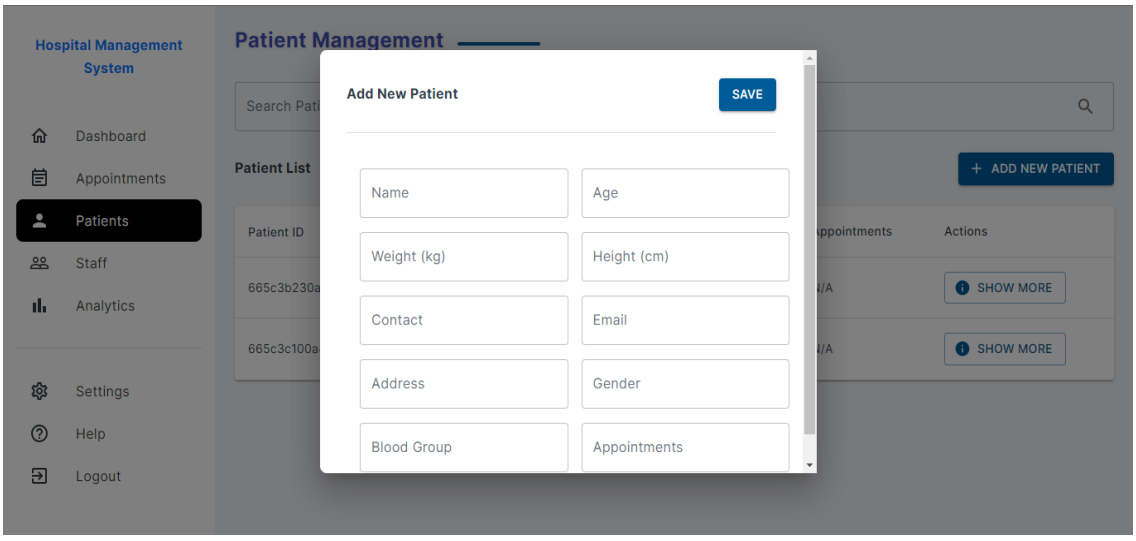
2.) Dashboard Interface



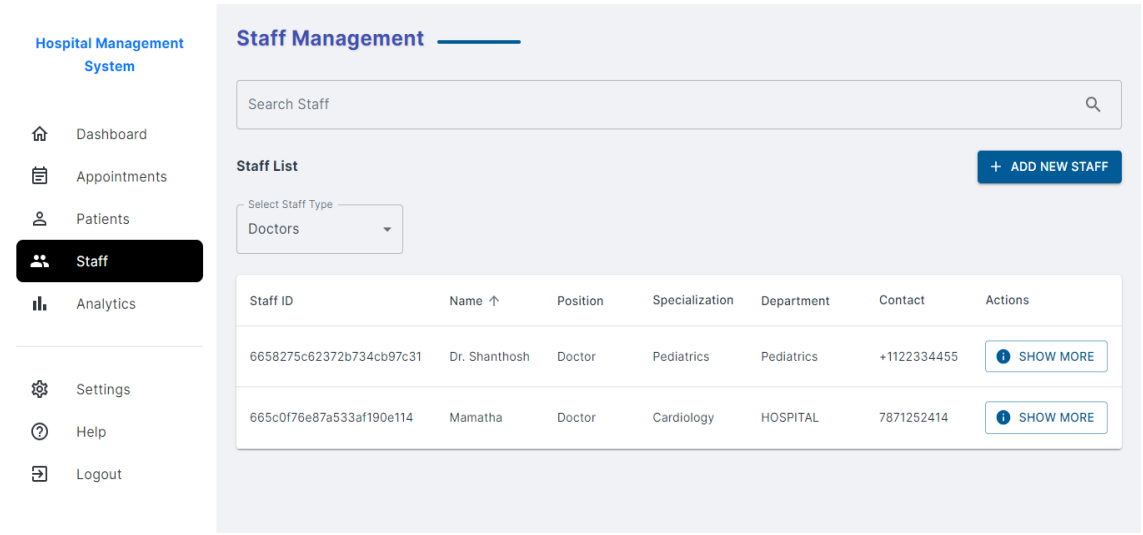
3.) Patient Page Interface

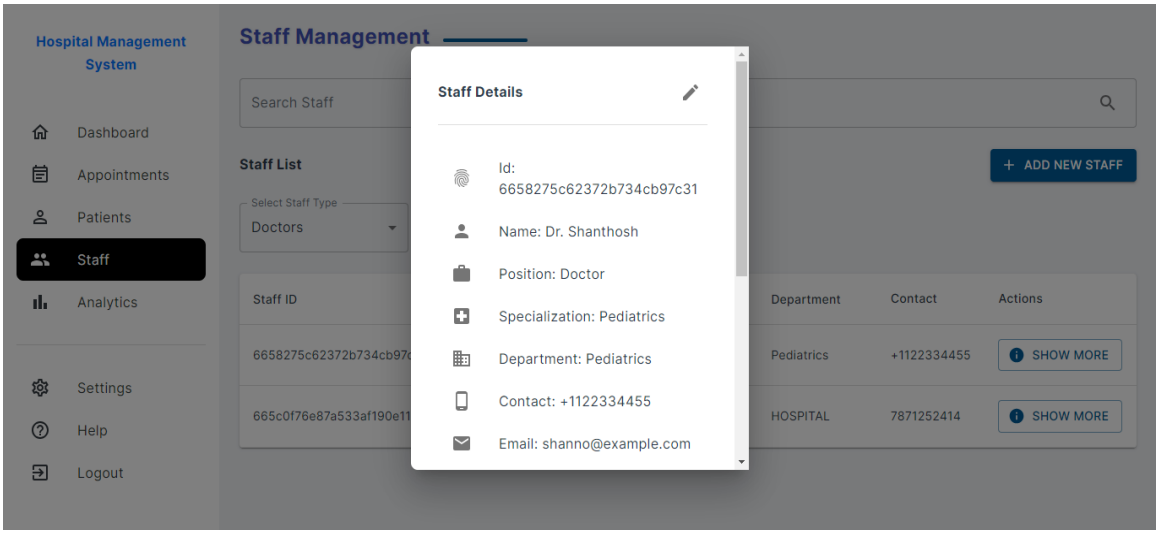


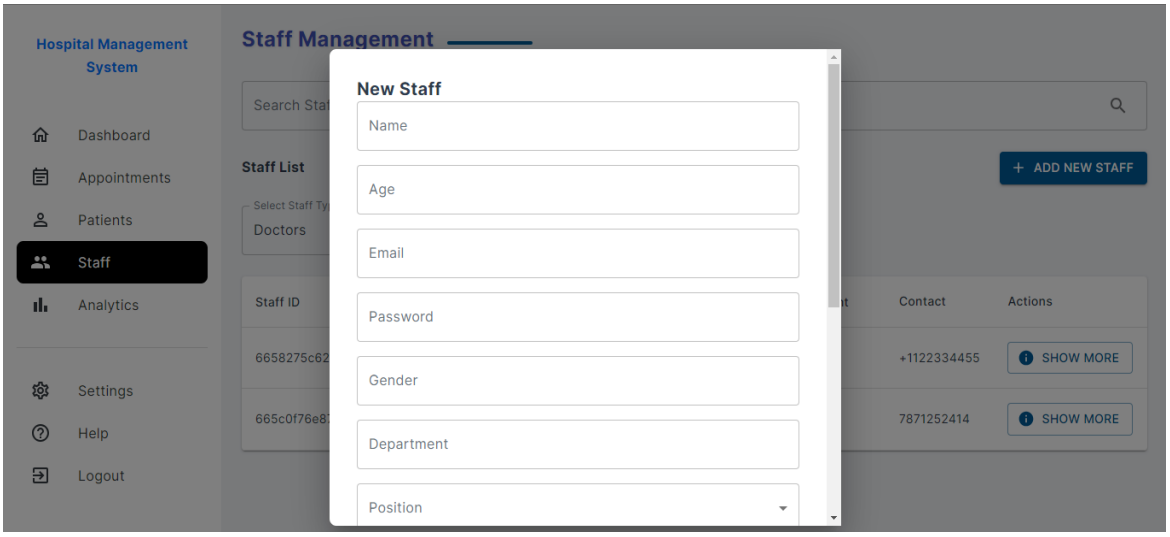




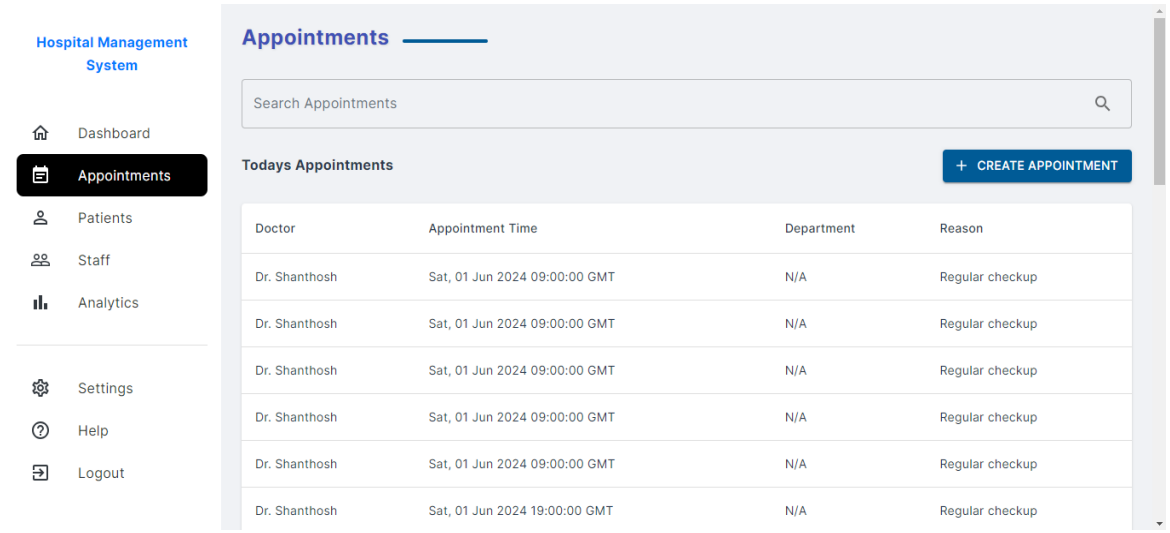
4.) Staff Page Interface

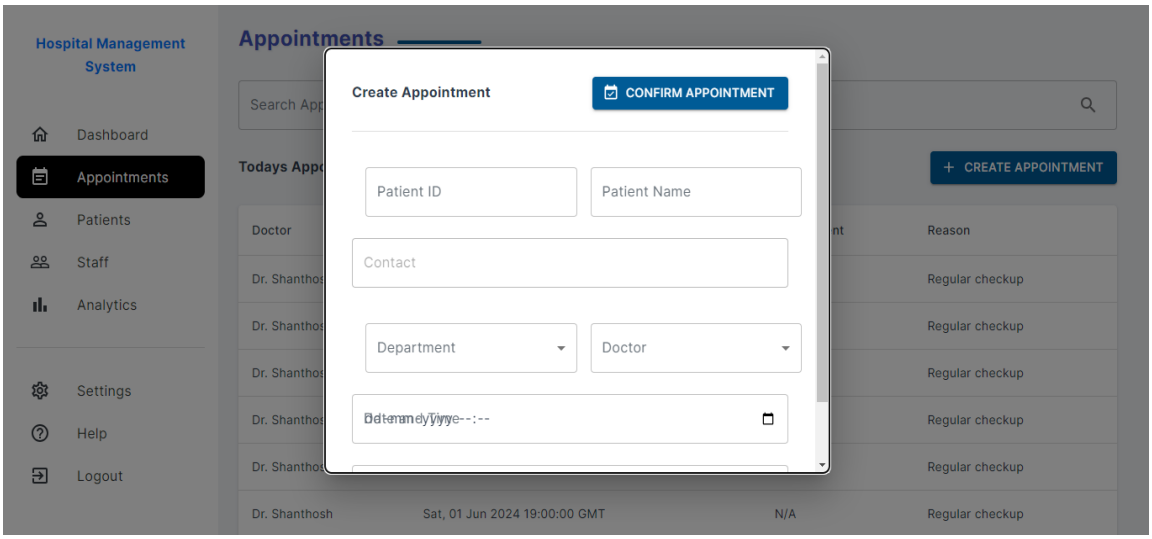






5.) Appointment Page Interface


Hospital Management System
Dashboard
Appointments
Patients
Staff
Analytics
Settings
Help
Logout
Appointments
Search Appointments
Todays Appointments
+ CREATE APPOINTMENT
Doctor
Appointment Time
Department
Reason
Dr. Shanthosh
Sat, 01 Jun 2024 09:00:00 GMT
N/A
Regular checkup
Dr. Shanthosh
Sat, 01 Jun 2024 09:00:00 GMT
N/A
Regular checkup
Dr. Shanthosh
Sat, 01 Jun 2024 09:00:00 GMT
N/A
Regular checkup
Dr. Shanthosh
Sat, 01 Jun 2024 09:00:00 GMT
N/A
Regular checkup
Dr. Shanthosh
Sat, 01 Jun 2024 09:00:00 GMT
N/A
Regular checkup
Dr. Shanthosh
Sat, 01 Jun 2024 19:00:00 GMT
N/A
Regular checkup


Hospital Management System
Dashboard
Appointments
Patients
Staff
Analytics
Settings
Help
Logout
Appointments
Create Appointment
CONFIRM APPOINTMENT
Patient ID
Patient Name
Contact
Department
Doctor
+ CREATE APPOINTMENT
Reason
Regular checkup
Dr. Shanthosh
Sat, 01 Jun 2024 19:00:00 GMT
N/A
Regular checkup

CHAPTER – 6: CONCLUSION

This hospital management system project has culminated in a powerful digital platform designed to streamline operations and significantly improve patient care. By implementing this admin-based system, hospitals can empower administrators to manage staff and patient data with ease. This includes performing CRUD (Create, Read, Update, Delete) operations on both staff and patient records, ensuring accurate and up-to-date information. Additionally, the system facilitates seamless patient-doctor appointment scheduling, eliminating the need for cumbersome phone calls or manual bookings. To further improve efficiency, doctors are automatically notified of upcoming appointments, ensuring they are prepared to deliver optimal care. This comprehensive system paves the way for a more efficient and patient-centric hospital experience.