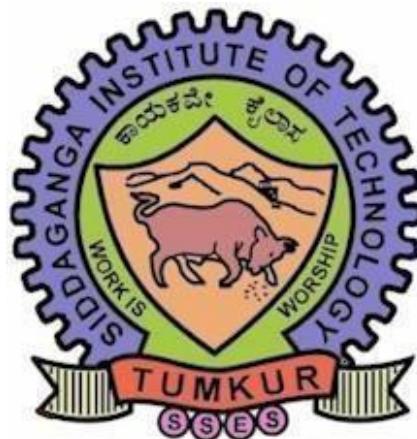


An  
**Activity Based Learning report**  
on

**Hospital OPD Management System**

**Submitted by**

- 1. Mahesh Basavaraj Teli 1SI24CI028**
- 2. Dhanush Ananda 1SI24CI012**
- 3. Sumanth J 1SI24CI058**
- 4. Siddappa Kariyappa Hirakannavar 1SI24CI056**



**Department of Artificial Intelligence and Machine Learning(CSE)**

**Siddaganga Institute of Technology, Tumakuru – 572103**

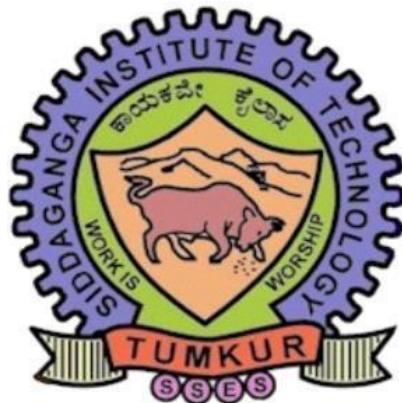
**(An Autonomous Institution, Affiliated to VTU, Belagavi & Recognized by AICTE, New Delhi)**

**2025 -2026**

## **SIDDAGANGA INSTITUTE OF TECHNOLOGY, TUMAKURU-3**

(An Autonomous Institution, Affiliated to VTU, Belagavi & Recognized by AICTE, New Delhi)

### **DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING (CSE)**



Name of student	USN	Marks
Mahesh Basavaraj Teli	1SI24CI028	
Dhanush Ananda	1SI24CI012	
Sumanth J	1SI24CI058	
Siddappa Kariyappa Hirakannavar	1SI24CI056	

**Signature of In charge Faculty**

## **ABSTRACT**

The Hospital OPD Management System is a web-based application designed to automate and streamline the Outpatient Department (OPD) activities of a hospital. Traditionally, OPD management in small and medium hospitals relies heavily on manual record keeping, paper files, and registers, which often leads to data loss, inefficiency, and difficulty in tracking patient history. This application provides a centralized digital platform where an administrator can securely manage patient records, doctor details, and appointment scheduling in an organized and efficient manner.

The system is developed using a modern full-stack architecture with a React and Tailwind CSS-based frontend for an interactive and user-friendly interface, a Spring Boot backend for handling business logic and RESTful APIs, and MongoDB as the database for storing hospital data. The application supports secure admin login, patient registration, doctor management, and appointment allocation by linking patients with doctors. By digitizing OPD operations, the system improves accuracy, reduces paperwork, saves time for hospital staff, and enhances the overall efficiency of hospital management.

## **PROBLEM STATEMENT**

In many hospitals and clinics, OPD management is still handled manually using paper-based registers, physical files, and handwritten appointment slips. This traditional approach creates several operational challenges such as difficulty in maintaining patient records, duplication of data, loss or damage of files, and delays in appointment scheduling. As the number of patients increases, it becomes increasingly hard to track patient history, doctor availability, and appointment details accurately.

Manual systems also lack proper access control and security, making sensitive patient information vulnerable to unauthorized access. There is no centralized system to efficiently link patients, doctors, and appointments, which often results in scheduling conflicts and inefficient utilization of hospital resources. Additionally, retrieving past records for follow-up visits is time-consuming and error-prone, negatively affecting both hospital staff productivity and patient experience.

Therefore, there is a strong need for a secure, scalable, and user-friendly OPD management system that can digitize hospital operations, provide role-based access (admin-controlled in the current version), and enable efficient handling of patients, doctors, and appointments through a single integrated platform.

## **OBJECTIVES OF THE SYSTEM**

1. To develop a secure admin-based OPD management system.
2. To maintain digital records of patients and doctors.
3. To enable efficient appointment scheduling by linking patients with doctors.
4. To reduce paperwork and manual errors.
5. To provide quick access to hospital data.
6. To create a scalable system that can be extended with additional modules such as billing, prescriptions, and role-based users in the future.

## **SCOPE OF THE PROJECT**

The scope of this project is limited to OPD management functionalities handled by a single administrator. The admin is responsible for managing patients, doctors, and appointments. Future enhancements may include doctor login, receptionist roles, billing modules, prescription management, and analytical dashboards.

## Tools Used

### 1. Frontend Technologies

- **React JS** – Used for building a dynamic and responsive user interface.
- **Tailwind CSS** – Used for designing modern, clean, and responsive UI components.
- **Axios** – Used for handling HTTP requests between frontend and backend.
- **Visual Studio Code** – Used as the primary code editor.

### 2. Backend Technologies

- **Java (JDK 21)** – Programming language used for backend development.
- **Spring Boot** – Framework used to develop RESTful APIs and manage backend logic.
- **Spring Data MongoDB** – Used for database interaction.
- **Spring Security (JWT)** – Used for authentication and admin login security.

### 3. Database

- **MongoDB** – NoSQL database used to store patient, doctor, appointment, and user data.
- **MongoDB Compass** – GUI tool used to visualize and manage database records.

### 4. Version Control

- **Git** – Used for source code management.
- **GitHub** – Used for hosting and sharing the project repository.

---

## Techniques Used

- **MVC Architecture** – Separation of concerns into Controller, Service, and Repository layers.
- **RESTful API Design** – Backend services are exposed as REST APIs.
- **JWT Authentication** – Secure login for admin user.

- **CRUD Operations** – Create, Read, Update, Delete operations implemented for all modules.
- **Modular Design** – Patient, Doctor, and Appointment modules developed independently.
- **Client–Server Communication** – Frontend communicates with backend via JSON-based APIs.

## PSEUDOCODE / CODE LOGIC

### **Admin Login (JWT Authentication):**

```

START
INPUT username, password
IF username and password are valid THEN
    Generate JWT token
    Send token to frontend
    Allow access to dashboard
ELSE
    Display error message
END IF
END

```

### **Add Patient Module:**

```

START
INPUT patient details (name, address, mobile)
VALIDATE input
STORE patient data in MongoDB
DISPLAY success message
REFRESH patient list
END

```

### **Add Doctor Module:**

START

INPUT doctor details (name, specialization, phone)

VALIDATE input

STORE doctor data in database

DISPLAY doctor list

END

### **Appointment Booking Module:**

START

SELECT patient from patient list

SELECT doctor from doctor list

SELECT date and time

GENERATE appointment record

STORE appointment in database

DISPLAY confirmation

END

### **SOME OF THE MAJOR JAVA FILES IN THE PROJECT :**

#### **1.AppointmentController.java-**

```
package com.hospital.opd.controllers;
```

```
import java.util.List;
```

```
import java.util.Optional;
```

```
import org.springframework.beans.factory.annotation.Autowired;
```

```
import org.springframework.http.HttpStatus;
```

```
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
```

```
import com.hospital.opd.models.Appointment;
import com.hospital.opd.service.AppointmentService;
```

```
/***
 * Appointment Controller
 * Handles all appointment-related HTTP requests
 */
```

```
@RestController
@RequestMapping("/api/appointments")
@CrossOrigin(origins = {"http://localhost:3000", "http://localhost:3001",
"http://localhost:3002"})
public class AppointmentController {
```

```
    @Autowired
    private AppointmentService appointmentService;
```

```
    /***
```

```
* Book an appointment
* POST /api/appointments
*/
@RequestMapping
public ResponseEntity<Appointment> bookAppointment(@RequestBody
Appointment appointment) {
    Appointment bookedAppointment =
appointmentService.bookAppointment(appointment);
    return new ResponseEntity<>(bookedAppointment, HttpStatus.CREATED);
}

/**
 * Get all appointments
 * GET /api/appointments
*/
@GetMapping
public ResponseEntity<List<Appointment>> getAppointments() {
    List<Appointment> appointments = appointmentService.getAppointments();
    return new ResponseEntity<>(appointments, HttpStatus.OK);
}

/**
 * Get appointments by patient ID
 * GET /api/appointments/patient/{patientId}
*/
@GetMapping("/patient/{patientId}")
public ResponseEntity<List<Appointment>>
getAppointmentsByPatientId(@PathVariable String patientId) {
```

```
    List<Appointment> appointments =
appointmentService.getAppointmentsByPatientId(patientId);
    return new ResponseEntity<>(appointments, HttpStatus.OK);
}

/***
 * Get appointments by doctor ID
 * GET /api/appointments/doctor/{doctorId}
 */
@GetMapping("/doctor/{doctorId}")
public ResponseEntity<List<Appointment>>
getAppointmentsByDoctorId(@PathVariable String doctorId) {
    List<Appointment> appointments =
appointmentService.getAppointmentsByDoctorId(doctorId);
    return new ResponseEntity<>(appointments, HttpStatus.OK);
}

/***
 * Get appointments by status
 * GET /api/appointments/status/{status}
 */
@GetMapping("/status/{status}")
public ResponseEntity<List<Appointment>>
getAppointmentsByStatus(@PathVariable String status) {
    List<Appointment> appointments =
appointmentService.getAppointmentsByStatus(status);
    return new ResponseEntity<>(appointments, HttpStatus.OK);
}
```

```

/***
 * Get appointment by ID
 * GET /api/appointments/{id}
 */
@GetMapping("/{id}")
public ResponseEntity<Appointment> getAppointmentById(@PathVariable String id)
{
    Optional<Appointment> appointment =
appointmentService.getAppointmentById(id);
    if (appointment.isPresent()) {
        return new ResponseEntity<>(appointment.get(), HttpStatus.OK);
    }
    return new ResponseEntity<>(HttpStatus.NOT_FOUND);
}

/***
 * Complete an appointment
 * PUT /api/appointments/{id}/complete
 */
@PutMapping("/{id}/complete")
public ResponseEntity<Appointment> completeAppointment(@PathVariable String
id) {
    Appointment appointment = appointmentService.completeAppointment(id);
    if (appointment != null) {
        return new ResponseEntity<>(appointment, HttpStatus.OK);
    }
    return new ResponseEntity<>(HttpStatus.NOT_FOUND);
}

```

```
}

/***
 * Cancel an appointment
 * PUT /api/appointments/{id}/cancel
 */
@PutMapping("/{id}/cancel")
public ResponseEntity<Appointment> cancelAppointment(@PathVariable String id)
{
    Appointment appointment = appointmentService.cancelAppointment(id);
    if (appointment != null) {
        return new ResponseEntity<>(appointment, HttpStatus.OK);
    }
    return new ResponseEntity<>(HttpStatus.NOT_FOUND);
}

/***
 * Delete an appointment
 * DELETE /api/appointments/{id}
 */
@DeleteMapping("/{id}")
public ResponseEntity<Void> deleteAppointment(@PathVariable String id) {
    appointmentService.deleteAppointment(id);
    return new ResponseEntity<>(HttpStatus.NO_CONTENT);
}
}
```

## **2.AuthController.java-**

```
package com.hospital.opd.controllers;

import com.hospital.opd.models.User;
import com.hospital.opd.services.AuthService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.Map;

@RestController
@RequestMapping("/api/auth")
@CrossOrigin(origins = {"http://localhost:3000", "http://localhost:3001",
"http://localhost:3002"})
public class AuthController {

    @Autowired
    private AuthService authService;

    @PostMapping("/register")
    public ResponseEntity<?> register(@RequestBody User user) {
        try {
            User registeredUser = authService.register(user);
            return ResponseEntity.ok(registeredUser);
        } catch (Exception e) {
            return ResponseEntity.badRequest().body(e.getMessage());
        }
    }
}
```

```

    }

    @PostMapping("/login")
    public ResponseEntity<?> login(@RequestBody Map<String, String> credentials) {
        try {
            String token = authService.login(
                credentials.get("username"),
                credentials.get("password")
            );
            return ResponseEntity.ok(token);
        } catch (Exception e) {
            return ResponseEntity.badRequest().body(e.getMessage());
        }
    }
}

```

### **3.DoctorController.java-**

```

package com.hospital.opd.controllers;

import java.util.List;
import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;

```

```
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.hospital.opd.models.Doctor;
import com.hospital.opd.service.DoctorService;

/**
 * Doctor Controller
 * Handles all doctor-related HTTP requests
 */
@RestController
@RequestMapping("/api/doctors")
@CrossOrigin(origins = {"http://localhost:3000", "http://localhost:3001",
"http://localhost:3002"})
public class DoctorController {

    @Autowired
    private DoctorService doctorService;

    /**
     * Add a new doctor
     * POST /api/doctors
     */
    @PostMapping
```

```
public ResponseEntity<Doctor> addDoctor(@RequestBody Doctor doctor) {  
    Doctor savedDoctor = doctorService.addDoctor(doctor);  
    return new ResponseEntity<>(savedDoctor, HttpStatus.CREATED);  
}  
  
/**  
 * Get all doctors  
 * GET /api/doctors  
 */  
@GetMapping  
public ResponseEntity<List<Doctor>> getAllDoctors() {  
    List<Doctor> doctors = doctorService.getAllDoctors();  
    return new ResponseEntity<>(doctors, HttpStatus.OK);  
}  
  
/**  
 * Get doctor by ID  
 * GET /api/doctors/{id}  
 */  
@GetMapping("/{id}")  
public ResponseEntity<Doctor> getDoctorById(@PathVariable String id) {  
    Optional<Doctor> doctor = doctorService.getDoctorById(id);  
    if (doctor.isPresent()) {  
        return new ResponseEntity<>(doctor.get(), HttpStatus.OK);  
    }  
    return new ResponseEntity<>(HttpStatus.NOT_FOUND);  
}
```

```
/**  
 * Get doctors by status (active/busy)  
 * GET /api/doctors/status/{status}  
 */  
  
@GetMapping("/status/{status}")  
public ResponseEntity<List<Doctor>> getDoctorByStatus(@PathVariable String  
status) {  
    List<Doctor> doctors = doctorService.getDoctorByStatus(status);  
    return new ResponseEntity<>(doctors, HttpStatus.OK);  
}  
  
/**  
 * Get doctors by specialization  
 * GET /api/doctors/specialization/{specialization}  
 */  
  
@GetMapping("/specialization/{specialization}")  
public ResponseEntity<List<Doctor>> getDoctorBySpecialization(@PathVariable  
String specialization) {  
    List<Doctor> doctors = doctorService.getDoctorBySpecialization(specialization);  
    return new ResponseEntity<>(doctors, HttpStatus.OK);  
}  
  
/**  
 * Update doctor  
 * PUT /api/doctors/{id}  
 */  
  
@PutMapping("/{id}")
```

```

public ResponseEntity<Doctor> updateDoctor(@PathVariable String id,
@RequestParam Doctor doctor) {
    Doctor updatedDoctor = doctorService.updateDoctor(id, doctor);
    if (updatedDoctor != null) {
        return new ResponseEntity<>(updatedDoctor, HttpStatus.OK);
    }
    return new ResponseEntity<>(HttpStatus.NOT_FOUND);
}

/**
 * Delete doctor
 * DELETE /api/doctors/{id}
 */
@DeleteMapping("/{id}")
public ResponseEntity<Void> deleteDoctor(@PathVariable String id) {
    doctorService.deleteDoctor(id);
    return new ResponseEntity<>(HttpStatus.NO_CONTENT);
}

```

#### **4.PatientController.java-**

```

package com.hospital.opd.controllers;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;

```

```
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.hospital.opd.models.Patient;
import com.hospital.opd.services.PatientService;

/**
 * REST Controller for Patient Management
 * Handles all patient-related HTTP requests
 *
 * @author Hospital OPD System
 * @version 1.0
 */

@RestController
@RequestMapping("/api/patients")
@CrossOrigin(origins = {"http://localhost:3000", "http://localhost:3001",
"http://localhost:3002"})
public class PatientController {

    @Autowired
    private PatientService patientService;
```

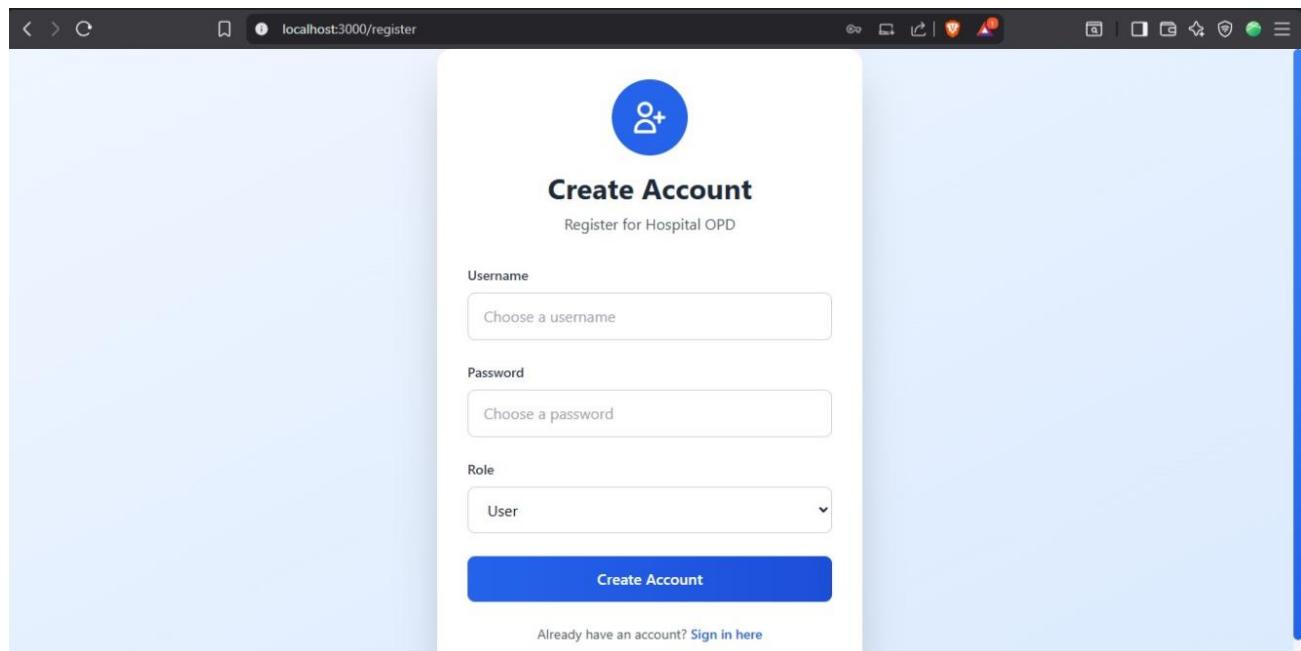
```
/**  
 * Get all patients from the database  
 *  
 * @return ResponseEntity containing list of all patients  
 * Endpoint: GET /api/patients  
 */  
  
@GetMapping  
public ResponseEntity<List<Patient>> getAllPatients() {  
    List<Patient> patients = patientService.getAllPatients();  
    return new ResponseEntity<>(patients, HttpStatus.OK);  
}  
  
/**  
 * Create a new patient record  
 *  
 * @param patient Patient object from request body  
 * @return ResponseEntity containing the saved patient with generated ID  
 * Endpoint: POST /api/patients  
 */  
  
@PostMapping  
public ResponseEntity<Patient> createPatient(@RequestBody Patient patient) {  
    Patient savedPatient = patientService.savePatient(patient);  
    return new ResponseEntity<>(savedPatient, HttpStatus.CREATED);  
}  
  
/**  
 * Delete a patient by ID  
 *
```

```

* @param id Patient ID to delete
* @return ResponseEntity with no content
* Endpoint: DELETE /api/patients/{id}
*/
@DeleteMapping("/{id}")
public ResponseEntity<Void> deletePatient(@PathVariable String id) {
    patientService.deletePatient(id);
    return new ResponseEntity<>(HttpStatus.NO_CONTENT);
}

```

## RESULTS:



Hospital OPD

## Patient Records

Manage and view all registered patients

+ Add Patient

TOTAL PATIENTS: 1

SHOWING: 1

SYSTEM STATUS: Active

Search by name, mobile, or address...

Refresh

NAME	ADDRESS	MOBILE	ACTIONS
Jhon Patient #1	tumkur	1234567890	Delete

Displaying 1 of 1 total patient

Hospital OPD

Doctors

## Add Doctor

Filter by Status: All

Name	Specialization	Email	Phone	Available Time	Status	Actions
abc	Dermatologist	harsh@gmail.com	1234567889	9 am to 5pm	Active	Delete
abc	Pediatrician	harsh@gmail.com	1234567890	9 am to 5pm	Active	Delete

---

```

    package com.hospital.opd.controllers;

import java.util.List;
import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

import com.hospital.opd.models.Doctor;
import com.hospital.opd.service.DoctorService;

/**
 * Doctor Controller
 * Handles all doctor-related HTTP requests
 */
@RestController
@RequestMapping("/api/doctors")
@CrossOrigin(origins = {"http://localhost:3000", "http://localhost:3001", "http://localhost:3002"})
public class DoctorController {

    @Autowired
    DoctorService doctorService;
}

```

The screenshot shows the MongoDB Compass application interface. On the left, the sidebar displays a list of connections, with 'localhost:27017' and its sub-database 'hospital\_db' selected. The 'patients' collection is highlighted. The main panel shows the 'Documents' tab with one document listed:

```

_id: ObjectId('6950c8c4282b09c0d0f95156')
name : "Jhon"
address : "tumkur"
mobile : "1234567890"
_class : "com.hospital.opd.models.Patient"

```

Below the document list are buttons for 'ADD DATA', 'EXPORT DATA', 'UPDATE', and 'DELETE'. At the bottom, there are navigation and search controls.

## REFERENCES:

### 1. Spring Boot Official Documentation

<https://spring.io/projects/spring-boot>

### 2. React Official Documentation

<https://react.dev>

### 3. MongoDB Documentation

<https://www.mongodb.com/docs>

### 4. Tailwind CSS Documentation

<https://tailwindcss.com/docs>

### 5. JWT Authentication Guide

<https://jwt.io/introduction>

### 6. GitHub Documentation

<https://docs.github.com>

## GITHUB REPOSITORY LINK:

<https://github.com/maheshteli07/hospital-opd.git>

