

ONLINE DOCTOR CONSULTATION

A report submitted in partial fulfillment of the requirements for the Award of Degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

By

Paturi Mercy Jemima

Regd. No.: 20B91A05M8

Under Supervision of Mr. Venu Gopal

Henotic IT Solutions, Hyderabad

(Duration: 5th June 2023 to 4th August 2023)



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SAGI RAMA KRISHNAM RAJU ENGINEERING COLLEGE

(Autonomous)

SRKR MARG, CHINNA AMIRAM, BHIMAVARAM – 534204, A.P

(Recognized by A.I.C.T.E New Delhi) (Accredited by NBA & NAAC)

(Affiliated to JNTU, KAKINADA)

SAGI RAMA KRISHNAM RAJU ENGINEERING COLLEGE

(Autonomous)

Chinna Amiram, Bhimavaram

DEPARTMENT OF
COMPUTER SCIENCE AND ENGINEERING



CERTIFICATE

This is to certify that the Summer Internship Report titled “Online Doctor Consultation” is the bonafide work done by Ms. Paturi Mercy Jemima [20B91A05M8] at the end of third year second semester at Henotic IT Solutions, Hyderabad from 5th June 2023 to 4th August 2023 in partial fulfilment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Engineering.

**Department Internship
coordinator**

Dean -T & P Cell

Head of the Department

ACKNOWLEDGEMENT

I take immense pleasure in thanking **Dr.K.V.Murali Krishnam Raju**, beloved principal of S.R.K.R Engineering College, Bhimavaram and **Dr.V.Chandra Sekhar**, esteemed Head of the Department (C.S.E), for having permitted me to carry out this software engineering project work.

Needless to mention that, **Dr.K.R.Satyanarayana, Dean T&P cell**, for his timely guidance in the conduct of my project work. Besides, this project made me realize the value of working together as a team and as a new experience in the working environment, which challenges me every minute.

I wish to express my deep sense of gratitude especially to my project Guide, **Sri. Venu Gopal, Henotic IT Solutions, Hyderabad**, for his able guidance and useful suggestions, which helped me in completing the project work, in time.

Finally, yet importantly, I would like to express my heart full thanks to my beloved parents for their blessings, my friends/classmates for their help and wishes for the successful completion of this project.

Ms.Motamarri J N V K K Mani Tejaswini (20B91A05I8)

Ms.NeerukondaLakshmi Nikhitha (20B91A05K8)

Ms.Paturi Mercy Jemima (20B91A05M8)

Ms. Kambhampati Bindhu Sri (20B91A5423)

Ms. Nallaparaju Renu Sathvika (20B91A5440)

Table of Contents

1.0 Abstract.....	1
2.0 Introduction	
2.1 Spring Vs Spring Boot.....	2
2.2 Why Spring Boot?	3
2.3 What is Reat and Why React?	4
3.0 Modules.....	7
4.0 System Requirements	8
5.0 Code.....	9
6.0 Screenshots	29

1.0 ABSTRACT

In the realm of virtual healthcare, this project outlines the development of a cutting-edge Doctor Consultation Website using a comprehensive Java full-stack technology stack. The primary goal is to revolutionize the online medical consultation process, ensuring a seamless and user-friendly experience for both healthcare professionals and patients.

This Full-Stack Java-based Doctor Consultation Website is crafted to enhance the efficiency of virtual healthcare services, providing a modern solution for medical professionals to connect with patients. The system boasts a robust back-end infrastructure, ensuring secure data management and streamlined communication between doctors and patients. On the front end, the website offers an intuitive interface, making it easy for patients to schedule appointments, communicate with healthcare providers, and access medical information. The platform prioritizes user experience, emphasizing accessibility and simplicity.

Key components of the project include user authentication, and user can book appointments based on the doctor schedule. The available slots are displayed on the home page where patient can book his appointment of his choice. After booking the appointment the patient can download the receipt. Whereas doctor can view the data of their patients and doctor can check his appointments and doctors can also update their schedule in the website. Security and data privacy are prioritized, with end-to-end encryption and stringent access controls.

The online doctor consultation platform not only offers convenience to patients but also empowers healthcare providers to reach a wider audience. This project, through the utilization of modern full stack technologies, significantly contributes to improving healthcare accessibility, reducing the burden on physical healthcare facilities, and enhancing the overall healthcare experience for patients.

2.0 INTRODUCTION

Java, with its robust and versatile features, serves as the foundation for the backend of "Online Doctor Consultation" project. Java has long been a stalwart in the world of software development, prized for its portability, reliability, and scalability. As the backend language of choice, Java empowers our application with the capability to handle complex operations, manage data securely, and ensure seamless interactions between users and the system.

Java's object-oriented nature allows us to design modular and maintainable code, facilitating efficient development and future enhancements. Its strong typing and comprehensive standard library provide developers with a wide range of tools and resources to build high-performance backend services.

In the context of the "Online Doctor Consultation" project, Java's Spring Boot framework is employed, streamlining the development of RESTful APIs and ensuring robust security measures. Whether it's managing user authentication, handling property listings, or facilitating smooth communication, Java's prowess in the backend is instrumental in delivering a reliable and feature-rich platform.

2.1 Spring Vs Spring Boot

Spring

Spring is a widely used Java framework that offers a comprehensive set of modules for building enterprise-level applications. It provides a range of features, including dependency injection, Aspect-Oriented Programming (AOP), Java Database Connectivity (JDBC), and more. Spring is known for its modularity, allowing developers to choose and configure specific modules according to their application's requirements. However, building a Spring application typically involves significant XML or Java-based configuration to set up various components. This flexibility can be powerful but may also require more manual configuration.

Spring Boot

Spring Boot is an extension of the Spring framework that focuses on simplifying the development of production-ready applications. It achieves this by emphasizing convention over configuration. Spring Boot comes with predefined configurations and auto-configuration options, reducing the need for extensive setup. Developers can get started quickly with minimal configuration, thanks to sensible defaults provided by Spring Boot. This approach significantly accelerates development and ensures that best practices are followed, making it an excellent choice for microservices and rapid application development.

In summary, Spring offers extensive flexibility and modularity, while Spring Boot prioritizes ease of use and rapid development through convention-driven configuration. The choice between them depends on the specific requirements of your project and your preferences for configuration and development speed.

2.2 Why Spring Boot?

Simplicity and Rapid Development:

Spring Boot was selected as the backend framework due to its emphasis on simplicity and rapid development. It streamlines the development process by providing a set of conventions and defaults that reduce the need for extensive configuration. This approach allows developers to concentrate more on crafting the project's unique business logic while minimizing the overhead of boilerplate code.

Robust Ecosystem:

Another compelling reason for adopting Spring Boot is its integration into the extensive Spring ecosystem. Spring offers a comprehensive suite of modules and tools for various aspects of enterprise application development. With Spring Boot as the foundation, the project benefits from seamless integration with Spring Data for data access, Spring Security for robust security measures, and many other components that enhance the project's capabilities and maintainability.

Microservices Support:

Spring Boot's innate support for microservices architecture aligns well with the project's goals. It facilitates the creation of RESTful APIs, making it an ideal choice for building the backend of a modern, scalable application. Embracing microservices allows the project to achieve greater flexibility and scalability as it evolves and grows to meet user demands.

Security:

Security is of paramount importance in an application dealing with sensitive user data. Spring Boot, coupled with Spring Security, delivers robust security features right out of the box. These features include authentication, authorization, and protection against common security vulnerabilities, ensuring the safeguarding of user information and interactions within the project platform.

Community and Documentation:

Spring Boot boasts a vibrant and active developer community. This community support translates into extensive documentation, tutorials, and a wealth of resources that are indispensable for addressing issues, staying current with best practices, and finding solutions to familiar challenges. It ensures that the project remains well supported and continuously improved.

Testing and Maintenance:

Spring Boot promotes best practices in testing, making it easier to create unit and integration tests for the application. This rigorous testing approach ensures the reliability of the platform and simplifies ongoing maintenance efforts, allowing for swift identification and resolution of issues.

Scalability and Performance:

Designed with scalability in mind, Spring Boot is an apt choice for applications expecting a growing user base and transaction load. It offers performance optimization and the flexibility to fine-tune the application to meet specific performance requirements, ensuring a responsive and efficient user experience as the project expands.

Cross-Platform Compatibility:

Java and Spring Boot are cross-platform, meaning the backend code can seamlessly run on various operating systems and cloud platforms. This cross-platform compatibility ensures flexibility and portability, facilitating easy deployment and scaling to accommodate the evolving needs of the project.

2.3 What is React and Why React?

React.js, often referred to as React, is a widely used open-source JavaScript library developed by Facebook. It is primarily designed for building user interfaces (UIs) and UI components in web applications. React is known for its efficiency and ease of use, making it a popular choice among developers.

Component-Based Architecture:

One of React's fundamental principles is its component-based architecture. In React, a user interface is constructed using reusable components. These components are like building blocks, where each represents a specific part of the UI. Developers can create and nest components to build complex user interfaces efficiently.

Virtual DOM:

React employs a Virtual DOM (Document Object Model) mechanism, a key innovation that sets it apart. Instead of directly manipulating the real DOM, React creates a virtual representation of it. When changes occur in a component's state or props, React updates this virtual DOM before efficiently updating the actual DOM. This approach minimizes costly DOM manipulations and enhances performance.

Declarative Programming:

React promotes a declarative programming paradigm, where developers describe how the UI should look based on the current state and props of components. Unlike imperative programming, which specifies step-by-step UI changes, the declarative approach simplifies development by focusing on what the UI should be, not how to achieve it.

Reactivity and Data Flow:

React is designed to be reactive. When data changes, React automatically updates the relevant components, ensuring that the UI remains in sync with the underlying data. This reactive nature simplifies UI development and helps prevent common bugs related to data inconsistency.

JSX (JavaScript XML):

React uses JSX, a syntax extension for JavaScript. JSX allows developers to write HTML-like code within JavaScript files, making it more intuitive to define the structure and appearance of UI components. JSX is transpiled into regular JavaScript using tools like Babel.

State and Props:

React components can manage two types of data: state and props. State represents internal component data that can change over time, while props (short for properties) are values passed from parent components to child components. Props are read-only and facilitate data sharing between components.

Unidirectional Data Flow:

React enforces a unidirectional data flow, meaning that data flows in one direction—typically from parent components to child components. This approach ensures a predictable data flow and simplifies debugging, as changes to data are systematically managed.

React Ecosystem:

React has a vibrant community, resulting in a vast ecosystem of libraries, tools, and extensions that complement its capabilities. Additionally, React can be employed not only for web applications but also for mobile app development through React Native. This allows developers to build mobile apps using React and JavaScript, sharing code between platforms.

Getting Started:

To begin using React, developers usually set up a development environment using tools like Node.js and npm (Node Package Manager). They then create React components, define their behavior, and build interactive and dynamic user interfaces for their web applications or mobile apps.

3.0 MODULES

1.Login Page: The Unified Login System module serves as the gateway for both doctors and patients to access the consultation platform. This feature consolidates login functionalities into a single, user-friendly page where individuals enter their credentials, choose their role (doctor or patient), and gain access accordingly. The system prioritizes clarity in error handling, ensures role-based access control, and incorporates user-friendly options for password recovery, contributing to a seamless and secure login experience.

2. Registration Page: The Registration Page module facilitates the onboarding process for doctors and patients by providing a streamlined account creation mechanism. Through this module, users input essential information during registration, triggering distinct access privileges based on their role within the healthcare system. Attention is given to robust data security, email verification protocols, terms acceptance procedures, and user-friendly password recovery options to enhance the overall registration experience.

3. Doctor Dashboard: The Doctor Dashboard module offers doctors a dedicated platform to manage their profiles and interactions within the consultation system. Here, doctors can efficiently edit personal details, oversee and adjust appointment schedules, and communicate seamlessly with their patients. Data privacy and security are paramount, ensuring that doctors have a centralized and secure environment to optimize their involvement within the system.

4. Patient Dashboard: Tailored for patients, the Patient Details module provides a user-friendly interface for managing their profiles within the consultation platform. Patients can conveniently view and update personal information, review appointment histories, and engage in communication with their respective doctors. The module underscores the importance of data privacy and security, fostering trust among patients as they navigate and interact with the platform.

5. Appointment Management: The Appointment Management module empowers both doctors and patients with efficient tools for scheduling and tracking appointments. Doctors gain a comprehensive view of their schedules and can seamlessly manage appointments, while patients can easily book appointments, check doctor availability, and receive timely confirmation notifications. The module ensures an intuitive interface, streamlining the appointment process for a positive user experience.

4.0 SOFTWARE REQUIREMENTS

Frontend (React JS):

Node.js: Node.js is used for running the development server, building the React application, and managing project dependencies. It's essential for executing JavaScript code on the server and automating tasks like testing and deployment.

Code Editor: Visual Studio Code is used for writing and editing React components, JavaScript code, and other frontend-related files.

React Dependencies: React and related libraries such as React Router, Axios, and UI components are used to create the user interface, manage routing, make HTTP requests to the backend, and enhance the user experience.

Backend (Spring Boot):

Java Development Kit (JDK): The JDK is the foundation for developing the backend of the restaurant website using Spring Boot. It enables the creation of Java-based RESTful APIs, data access, and business logic implementation.

Integrated Development Environment (IDE): An IDE like IntelliJ IDEA or Spring Tool Suite provides a development environment for writing, debugging, and deploying Spring Boot applications. It offers tools and features to streamline Java development.

Spring Boot Dependencies: Spring Boot dependencies, including Spring Web, Spring Data JPA, and Spring Security, are used to set up the backend application. Spring Web is used to build RESTful APIs, Spring Data JPA simplifies database interactions, and Spring Security handles authentication and authorization.

Database: MySQL is used to store data related to menus, orders, customers, and payments. Spring Boot integrates with the chosen database system to perform CRUD (Create, Read, Update, Delete) operations.

Database Requirements:

Database Server: The database server hosts the restaurant's data and provides services for storing and retrieving information, ensuring the website can access and manipulate data as needed.

5.0 CODES

DoctorApplication.java

```
package com.project.doctor;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
public class DoctorApplication {

    public static void main(String[] args)

    {

        SpringApplication.run(DoctorApplication.class, args);

    }

}
```

DoctorController.java

```
package com.project.doctor.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Controller;
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.project.doctor.entity.Doctor;
import com.project.doctor.service.DoctorService;

@CrossOrigin(origins="http://localhost:3000")
@RestController
```

```

@RequestMapping("/doctor")
public class DoctorController
{
    @Autowired
    DoctorService doctorService;

    @GetMapping(produces="application/json")
    public ResponseEntity<List<Doctor>>getAllDoctors()
    {
        return new
        ResponseEntity<List<Doctor>>(doctorService.getAllDoctors(),HttpStatus.OK);
    }

    @GetMapping(value="/{doctorId}",produces="application/json")
    public ResponseEntity<Doctor> getOneDoctor(@PathVariable int doctorId)
    {
        Doctor d=doctorService.getOneDoctor(doctorId);
        if(d!=null)
            return new ResponseEntity<Doctor>(d,HttpStatus.OK);
        return new ResponseEntity<Doctor>(d,HttpStatus.NOT_FOUND);
    }

    @PostMapping(value="/",consumes = "application/json")
    public HttpStatus addDoctor(@RequestBody Doctor doc)
    {
        if(doctorService.insertDoctor(doc))
            return HttpStatus.OK;
        return HttpStatus.NOT_MODIFIED;
    }

    @PutMapping(value="/",consumes = "application/json")
    public HttpStatus modifyDoctor(@RequestBody Doctor doc)
    {
        if(doctorService.modifyDoctor(doc))
            return HttpStatus.OK;
        return HttpStatus.NOT_MODIFIED;
    }

    @DeleteMapping("/{doctorId}")

```

```

public HttpStatus deleteDoctor(@PathVariable int doctorId)
{
    if(doctorService.deleteDoctor(doctorId))
        return HttpStatus.OK;
    return HttpStatus.NOT_FOUND;
}
}

```

PatientController.java

```

package com.project.doctor.controller;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Controller;
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.project.doctor.entity.Patient;
import com.project.doctor.service.PatientService;
@CrossOrigin(origins="http://localhost:3000")
@RestController
@RequestMapping("/patient") //http://localhost:8092/patient
public class PatientController
{
    @Autowired
    PatientService patientService;
    @GetMapping(produces="application/json")
    public ResponseEntity<List<Patient>>getAllPatients()

```



```

{
    return new
    ResponseEntity<List<Patient>>(patientService.getAllPatients(),HttpStatus.OK);
}
@GetMapping(value="/{patientId}",produces="application/json")
public ResponseEntity<Patient> getOnePatient(@PathVariable int patientId)
{
    Patient p=patientService.getOnePatient(patientId);
    if(p!=null)
        return new ResponseEntity<Patient>(p,HttpStatus.OK);
    return new ResponseEntity<Patient>(p,HttpStatus.NOT_FOUND);
}
@PostMapping(value="/",consumes = "application/json")
public HttpStatus addPatient(@RequestBody Patient pat)
{
    if(patientService.insertPatient(pat))
        return HttpStatus.OK;
    return HttpStatus.NOT_MODIFIED;
}
@PutMapping(value="/",consumes = "application/json")
public HttpStatus modifyPatient(@RequestBody Patient pat)
{
    if(patientService.modifyPatient(pat))
        return HttpStatus.OK;
    return HttpStatus.NOT_MODIFIED;
}
@DeleteMapping("/{patientId}")
public HttpStatus deletePatient(@PathVariable int patientId)
{
    if(patientService.deletePatient(patientId))
        return HttpStatus.OK;
    return HttpStatus.NOT_FOUND;
}
}

```

Doctor.java

```
package com.project.doctor.entity;
import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.Table;
@Entity
@Table(name="doctor")
public class Doctor {
    @Id
    @Column(name="doctorid")
    private int doctorId;
    @Column(name="dname")
    private String doctorName;
    @Column(name="specialization")
    private String doctorSpecialization;
    @Column(name="qual")
    private String qualification;
    @Column(name="hname")
    private String hospitalname;
    @Column(name="yearsofexperience")
    private int yearsofexperience;
    @Column(name="ratings")
    private int ratings;
    public Doctor() { }
    public Doctor(int doctorId, String doctorName, String doctorSpecialization, String
    qualification,String hospitalname, int yearsofexperience, int ratings) {
        super();
        this.doctorId = doctorId;
        this.doctorName = doctorName;
        this.doctorSpecialization = doctorSpecialization;
        this.qualification = qualification;
        this.hospitalname = hospitalname;
        this.yearsofexperience = yearsofexperience;
    }
}
```

```

        this.ratings = ratings;
    }
    public int getDoctorId() {
        return doctorId;
    }
    public void setDoctorId(int doctorId) {
        this.doctorId = doctorId;
    }
    public String getDoctorName() {
        return doctorName;
    }
    public void setDoctorName(String doctorName) {
        this.doctorName = doctorName;
    }
    public String getDoctorSpecialization() {
        return doctorSpecialization;
    }
    public void setDoctorSpecialization(String doctorSpecialization) {
        this.doctorSpecialization = doctorSpecialization;
    }
    public String getQualification() {
        return qualification;
    }
    public void setQualification(String qualification) {
        this.qualification = qualification;
    }
    public String getHospitalname() {
        return hospitalname;
    }
    public void setHospitalname(String hospitalname) {
        this.hospitalname = hospitalname;
    }
    public int getYearsofexperience() {
        return yearsofexperience;
    }

```

```

    }

    public void setYearsofexperience(int yearsofexperience) {
        this.yearsofexperience = yearsofexperience;
    }

    public int getRatings() {
        return ratings;
    }

    public void setRatings(int ratings) {
        this.ratings = ratings;
    }
}

```

Patient.java

```

package com.project.doctor.entity;
import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.Table;
@Entity
@Table(name="patient")
public class Patient {

    @Id
    @Column(name="patientid")
    private int patientId;

    @Column(name="pfname")
    private String patientfirstName;

    @Column(name="plname")
    private String patientlastName;

    @Column(name="phno")
    private String phonenumber;

    @Column(name="addr")
    private String address;

    @Column(name="email")
    private String email;

    public Patient() {}
}

```

```

    public Patient(int patientId, String patientfirstName, String patientlastName, String
    phonenumber, String address,
                    String email) {
        super();
        this.patientId = patientId;
        this.patientfirstName = patientfirstName;
        this.patientlastName = patientlastName;
        this.phonenumber = phonenumber;
        this.address = address;
        this.email = email;
    }
    public int getPatientId() {
        return patientId;
    }
    public void setPatientId(int patientId) {
        this.patientId = patientId;
    }
    public String getPatientfirstName() {
        return patientfirstName;
    }
    public void setPatientfirstName(String patientfirstName) {
        this.patientfirstName = patientfirstName;
    }
    public String getPatientlastName() {
        return patientlastName;
    }
    public void setPatientlastName(String patientlastName) {
        this.patientlastName = patientlastName;
    }
    public String getPhonenumber() {
        return phonenumber;
    }
    public void setPhonenumber(String phonenumber) {
        this.phonenumber = phonenumber;
    }

```

```

    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }
}

```

DoctorRepository.java

```

package com.project.doctor.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import com.project.doctor.entity.Doctor;

public interface DoctorRepository extends JpaRepository<Doctor,Integer>
{
}

```

PatientRepository.java

```

package com.project.doctor.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import com.project.doctor.entity.Patient;

public interface PatientRepository extends JpaRepository<Patient,Integer>
{
}

```

DoctorService.java

```

package com.project.doctor.service;

import java.util.List;
import java.util.Optional;
import org.springframework.beans.factory.annotation.Autowired;

```

```

import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import com.project.doctor.entity.Doctor;
import com.project.doctor.repository.DoctorRepository;

@Service
public class DoctorService
{
    @Autowired
    DoctorRepository doctorRepository;

    @Transactional(readOnly=true)
    public List<Doctor> getAllDoctors()
    {
        return doctorRepository.findAll();
    }

    @Transactional(readOnly=true)
    public Doctor getOneDoctor(int doctorId)
    {
        Optional<Doctor> doc=doctorRepository.findById(doctorId);
        if(doc.isPresent())
            return doc.get();
        return null;
    }

    @Transactional
    public boolean insertDoctor(Doctor doc)
    {
        return doctorRepository.save(doc)!=null;
    }

    @Transactional
    public boolean modifyDoctor(Doctor doc)
    {
        return doctorRepository.save(doc)!=null;
    }

    @Transactional

```

```

        public boolean deleteDoctor(int doctorId)
        {
            long count=doctorRepository.count();
            doctorRepository.deleteById(doctorId);
            return count>doctorRepository.count();
        }
    }

```

PatientService.java

```

package com.project.doctor.service;

import java.util.List;
import java.util.Optional;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import com.project.doctor.entity.Patient;
import com.project.doctor.repository.PatientRepository;

@Service
public class PatientService
{
    @Autowired
    PatientRepository patientRepository;

    @Transactional(readOnly=true)
    public List<Patient> getAllPatients()
    {
        return patientRepository.findAll();
    }

    @Transactional(readOnly=true)
    public Patient getOnePatient(int patientId)
    {
        Optional<Patient> pat=patientRepository.findById(patientId);
        if(pat.isPresent())
            return pat.get();
    }
}

```



```

        return null;
    }

    @Transactional
    public boolean insertPatient(Patient pat)
    {
        return patientRepository.save(pat)!=null;
    }

    @Transactional
    public boolean modifyPatient(Patient pat)
    {
        return patientRepository.save(pat)!=null;
    }

    @Transactional
    public boolean deletePatient(int patientId)
    {
        long count=patientRepository.count();
        patientRepository.deleteById(patientId);
        return count>patientRepository.count();
    }
}

```

App.Js:

```

import React, { useEffect, useState } from "react";
import "bootstrap";
import "bootstrap/dist/css/bootstrap.css";
import "bootstrap/dist/js/bootstrap.js";
import "bootstrap/dist/css/bootstrap.min.css";
import { BrowserRouter as Router, Route, Switch } from "react-router-dom";
import "./App.css";
import Home from "./Pages/Home";
import DoctorLogin from "./Pages/DoctorLogin";

```

```

import DoctorDashboard from "./Pages/DoctorDashboard";
import PaitentDashboard from "./Pages/PaitentDashboard";
import Error from "./Pages/Error";
import { AuthContext } from "./Auth/AuthContext";
import PhoneNumber from "./components/PhoneNumber";
import PersonalDetails from "./Doctor/PersonalDetails";
import SearchDoctor from "./Patient/SearchDoctor";
import PerviousAppointments from "./Patient/PerviousAppointments";
import Spinner from "react-bootstrap/Spinner";
import Selectdate from "./Patient/Selectdate";
import BookingSlots from "./Doctor/BookingSlots";
import AppointmentStatus from "./Patient/AppointmentStatus";

function App() {
  const [token, setToken] = useState(window.localStorage.getItem("token"));
  const [googleId, setGoogleId] = useState(
    window.localStorage.getItem("googleId")
  );
  const [apiLoaded, setApiLoaded] = useState(false);
  useEffect(() => {
    if (window.gapi !== undefined) {
      setApiLoaded(false);
      window.gapi.load("client:auth2", initClient);
      function initClient() {
        window.gapi.client.init({
          apiKey: process.env.REACT_APP_API_KEY,
          clientId: process.env.REACT_APP_CLIENT_ID,
          discoveryDocs: [process.env.REACT_APP_DISCOVERY_DOCS],
          scope: process.env.REACT_APP_SCOPE,
        })
      }
    }
  }).then(
    function () {
      if (window.gapi.auth2.getAuthInstance().isSignedIn.get()) {
        console.log(`Is signed in? ${window.gapi.auth2.getAuthInstance().isSignedIn.get()}`);
      }
    }
  );
}

```

```

    } else {
      console.log("Currently Logged Out!!");
    }
    setApiLoaded(true);
  },
  function (error) {
    console.log(`error ${JSON.stringify(error)}`);
    setApiLoaded(true);
  });
  setApiLoaded(true);
    } else {
      console.log("[Google] inside the else block line 54 App.js");
      setApiLoaded(false);
    }
  return apiLoaded ? (
    <Router>
    <AuthContext.Provider value={{ token, setToken, googleId, setGoogleId }}>
      <Switch>
        <Route exact path="/" component={Home} />
        <Route exact path="/doctorlogin" component={DoctorLogin} />
        <Route exact path="/doctor" component={DoctorDashboard} />
        <Route exact path="/patient/searchdoctor" component={SearchDoctor} />
        <Route exact path="/patient" component={PaitentDashboard} />
        <Route exact path="/patient/update-phone" component={PhoneNumber} />
        <Route exact path="/patient/previousappointments" component={PerviousAppointments} />
        <Route exact path="/doctor/perosnaldetails" component={PersonalDetails} />
        <Route exact path="/doctor/payment-history" component={DocAppointments} />
        <Route exact path="/patient/selectdate" component={Selectdate} />
        <Route exact path="/patient/book-slot" component={BookingSlots} />
        <Route exact path="/patient/appointment-status" component={AppointmentStatus} />
        <Route path="*" />
        <Error />
      </Route>
    </Switch>

```

```

</AuthContext.Provider>

</Router>

    ): (

<div style={{ width: "100%", display: "flex", justifyContent: "center" }}>

<Spinner animation="border" variant="danger" role="status">

<span className="sr-only">Loading...</span>

</Spinner>

</div>

    );

}

export default App;

Auth Context Js:

import { createContext } from 'react';

export const AuthContext = createContext();

LoginForm Js:

import React, { useContext, useState } from 'react';
import { Redirect, useHistory } from "react-router-dom";
import axios from 'axios';

import { AuthContext } from '../Auth/AuthContext';

const LoginForm = () => {

    const [username, setUsername] = useState("");
    const [password, setPassword] = useState("");
    const [status, setStatus] = useState(0);
    const { token, setToken, googleId, setGoogleId } = useContext(AuthContext);
    const history = useHistory();

    async function login() {
        try {
            const res = await axios.post(
                `${process.env.REACT_APP_SERVER_URL}/doctors/login/`, {
                username: username,
                password: password});
            setStatus(res.status);

            const token = res.data.token;
            if (res.status === 200) {

```

```

        window.localStorage.setItem("token", token);
        window.localStorage.removeItem("googleId");
        setGoogleId(null);
        setToken(token);
        history.push('/doctor');}}
    catch (err) {
console.log(err);}}
if (token && !googleId) {
    return <Redirect to="/doctor" />
    return (<Container className='text-center'>
<Row>
<className='offset-lg-3 mt-5 '>
<Card>
<Form>
<CardHeader className="">Welcome back, Doc</CardHeader>
<CardBody >
    FormGroup row>
    <Label for='email' sm={3}>Username
    </Label>
    <Input type='text' name='username' id='username'placeholder='Doctor ID'
    onChange={(e) => setUsername(e.target.value)}/>
    </Col>
    </FormGroup>
    <FormGroup row>
    <Label for='password' sm={3}>Password</Label><Col sm={9}>
    <Input type='password' name='password' id='password' placeholder='Password'
    onChange={(e) => setPassword(e.target.value)}onKeyPress={(target) => {
    if (target.charCode === 13) {
    login();}} } />
    </Col>
    </FormGroup>
    {status === 201 && <p className="warning" style={{ color: "red", fontSize: "15px"
    }}>Wrong username or password! Please try again</p>}
    </CardBody>

```

```

<CardFooter>
  <Button block color="primary" onClick={login}>Sign In </Button></CardFooter>
</Form>
</Card>
</Col>
</Row>
</Container>);}

export default LoginForm;

```

Doctor Dashboard:

```

import React from "react";
import Navbar from "../Basic/Navbar";
import Leftside from "../Dashbaord/LeftsideDoctor";
import TodaysSchedule from "../Doctor/TodaysSchedule";
import PatientDetails from "../Patient/PatientDetails";
import "../Dashbaord/dashboard.css";

const DoctorDashboard = () => {
  return (
    <div className="bg-dark" style={{ height: "100vh" }}>
      <Navbar />
      <div>
        <div className="row m-5" style={{ maxWidth: "100%" }}>
          <div className="col-3 col-md-3 p-4 bg-white" style={{ height: "80vh" }} >
            <Leftside /></div>
          <div
            className="col-9 col-md-9 p-4"
            style={{
              border: "15px solid black ",
              height: "80vh",
              backgroundColor: "#6c757d",
            }}
          >
            <TodaysSchedule /></div> </div> </div></div>);
  };
export default DoctorDashboard;

```

Patient Dashboard:

```
import React, { useContext } from "react";
import Navbar from "../Basic/Navbar";
import Leftside from "../Dashbaord/LeftsidePatient";
import { useState, useEffect } from "react";
import Axios from "axios";
import "../Dashbaord/dashboard.css";
import { AuthContext } from "../Auth/AuthContext";
const PersonalDetails = () => {
  const [patient, setPatient] = useState({});
  const [loading, setLoading] = useState(true);
  const { googleId } = useContext(AuthContext);
  useEffect(() => {
    setLoading(true);
    const getPatientDetails = async () => {
      const res = await Axios.get(
        `${process.env.REACT_APP_SERVER_URL}/patients/getPatientDetails/${googleId}`
      );
      if (res.status === 200) {
        setPatient(res.data);
        window.localStorage.setItem("user", JSON.stringify(res.data));
        setLoading(false);
      } else {
        console.log(res.data.message);
        setLoading(false);
      }
    };
    getPatientDetails();
  }, [googleId]);
  return (
    <div className="bg-dark" style={{ height: "100vh" }}>
      <Navbar />
      {loading ? (
        <div className="row justify-content-center position-relative">
          <div className="spinner-border align-middle d-flex justify-content-center position-absolute top-50 start-50 translate-middle" style={{ width: "10rem", height: "10rem" }} role="status"></div></div>

```

```

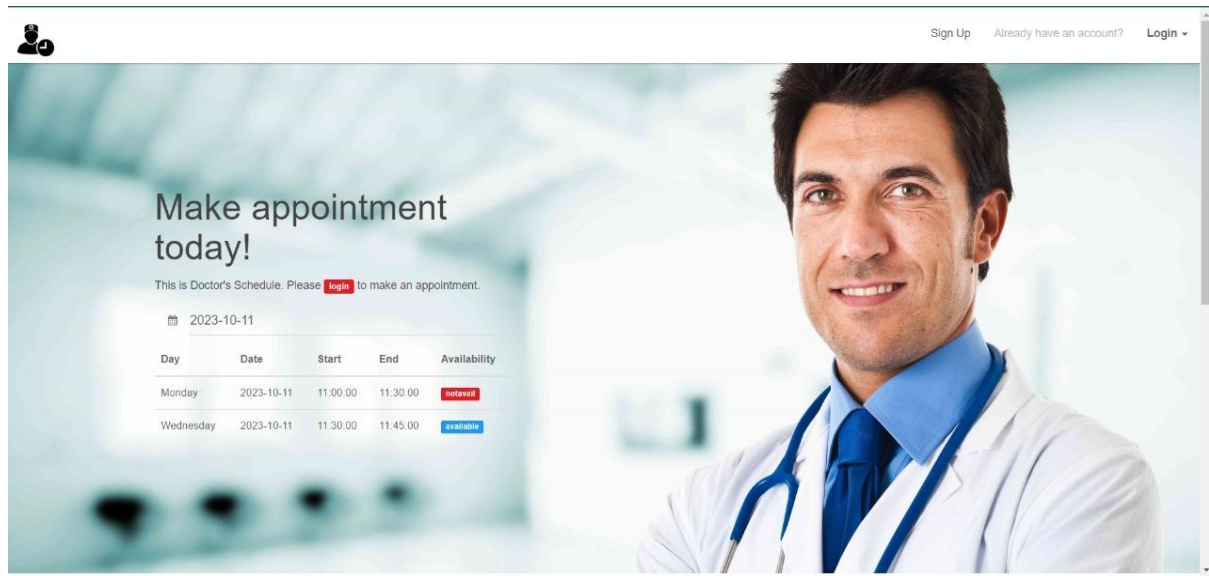
): (
<div>
  <div className="row m-5" style={{ maxWidth: "100%" }}>
    <div className="col-3 col-md-3 p-4 bg-white "style={{ height: "80vh" }}>
<Leftside /></div>
<div className="col-9 col-md-9 p-4" style={{ border: "15px solid yellow ",height:
"80vh", backgroundColor: "#6c757d",}}>
  <div className="row ">
    <div className="col-9 col-md-9 p-4">
      <div className="card mb-4">
        <h4 className="card-header">Personal Details</h4>
        <ul className="list-group">
          <li className="list-group-item">
            <span className="badge badge-success mr-2 p-2"> Name:</span>
            {patient.name}
          </li>
          <li className="list-group-item">
            <span className="badge badge-success mr-2 p-2">
              Email:
            </span>
            {patient.email}
          </li>
          <li className="list-group-item">
            <span className="badge badge-success mr-2 p-2">
              Phone No:</span>{patient.phoneNumber}
          </li>
        </ul>
      </div>
    </div>
  <div className="col-3 col-md-3 p-4 ">
    <img src={patient.picture} style={{ width: "100%" }} alt=""/>
  </div>
</div>
</div>
</div>

```

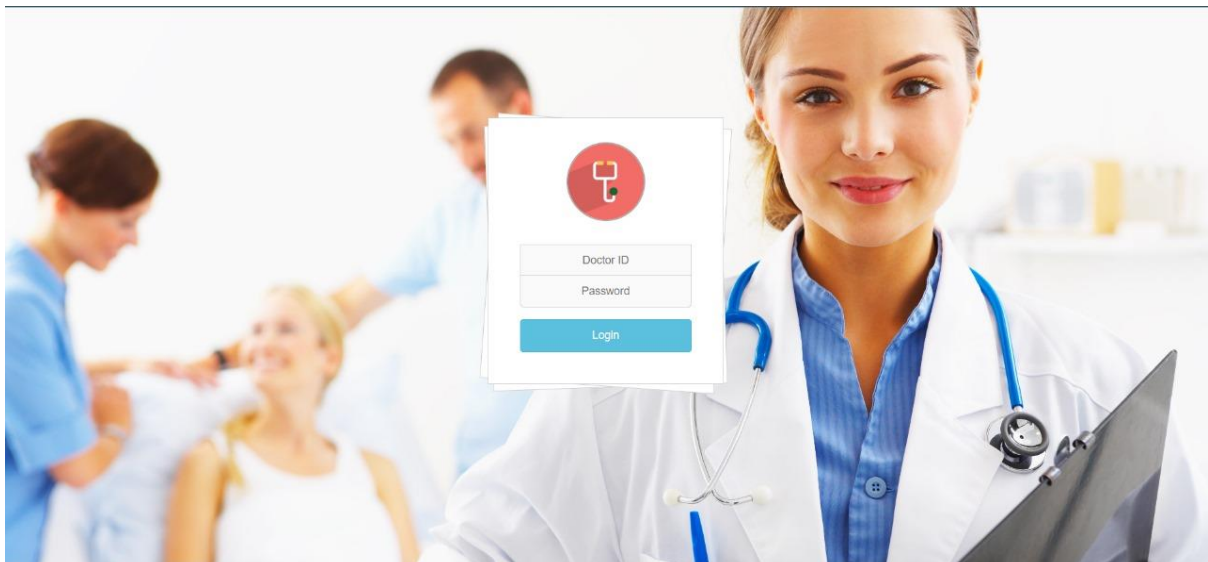


```
    </div>
  </div>
})
</div>
);
};
export default PersonalDetails;
```

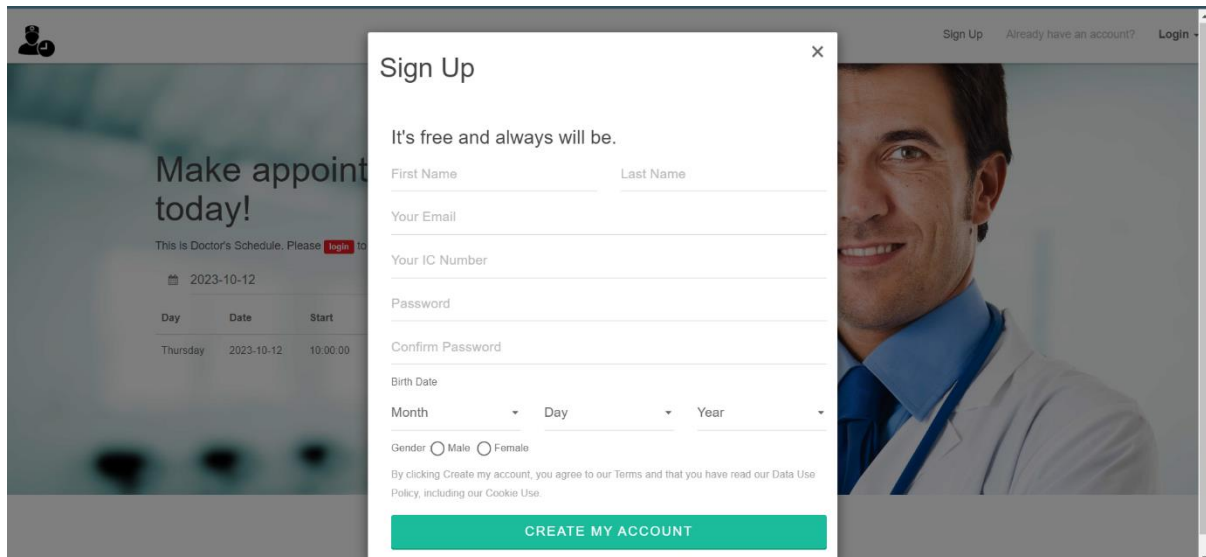
6.0 SCREENSHOTS



Home Page



Doctor Login



A screenshot of a web application showing a 'Sign Up' modal. The modal is white with a green 'CREATE MY ACCOUNT' button. It contains fields for First Name, Last Name, Email, IC Number, Password, Confirm Password, Birth Date (Month, Day, Year), and Gender (Male, Female). A checkbox for 'I agree to our Terms and that you have read our Data Use Policy, including our Cookie Use' is also present. The background shows a blurred view of a doctor's schedule and a smiling doctor.

Sign Up

It's free and always will be.

First Name Last Name

Your Email

Your IC Number

Password

Confirm Password

Birth Date

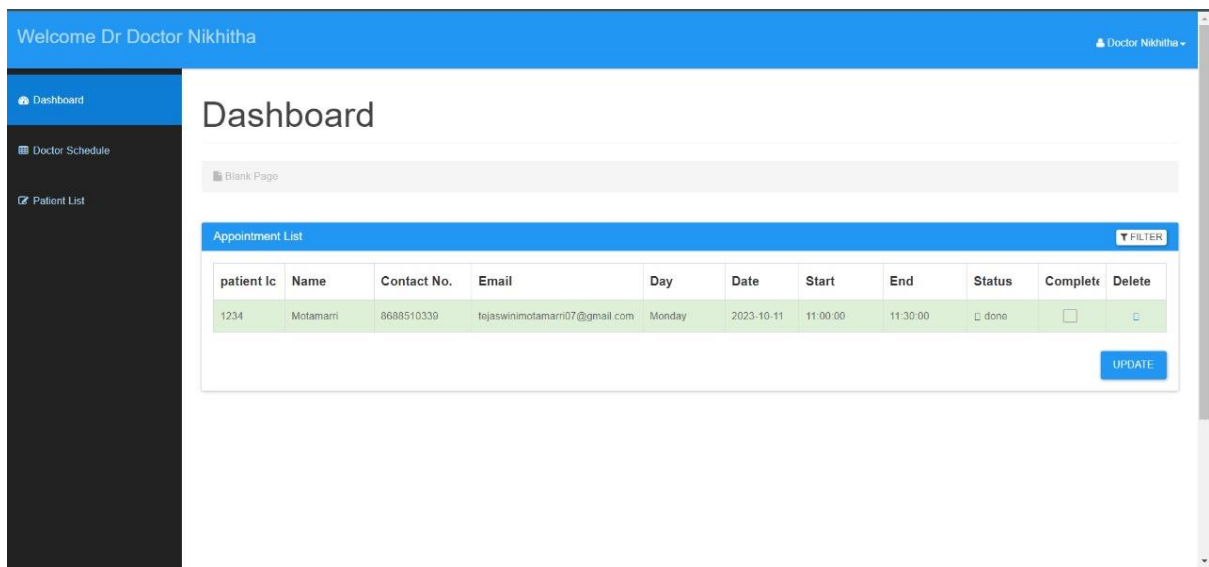
Month Day Year

Gender ☐ Male ☐ Female

By clicking Create my account, you agree to our Terms and that you have read our Data Use Policy, including our Cookie Use.

CREATE MY ACCOUNT

Patient Sign Up



A screenshot of a doctor's dashboard. The top bar is blue with 'Welcome Dr Doctor Nikhitha' and a user profile icon. The left sidebar is dark blue with links to Dashboard, Doctor Schedule, and Patient List. The main content area is white and titled 'Dashboard'. It features a 'Blank Page' button and an 'Appointment List' table. The table has columns for patient ID, Name, Contact No., Email, Day, Date, Start, End, Status, Complete, and Delete. A single appointment is listed for patient 1234, Dr. Motamari, on Monday, 2023-10-11, from 11:00:00 to 11:30:00. The status is 'done' and the complete checkbox is checked. An 'UPDATE' button is at the bottom right of the table.

Welcome Dr Doctor Nikhitha

Doctor Nikhitha

Dashboard

Blank Page

Appointment List

patient Id	Name	Contact No.	Email	Day	Date	Start	End	Status	Complete	Delete
1234	Motamari	8688510339	tejaswinimotamari07@gmail.com	Monday	2023-10-11	11:00:00	11:30:00	done	<input checked="" type="checkbox"/>	<input type="button" value="C"/>

UPDATE

Doctor Dashboard

Welcome Dr Doctor Nikhitha
Doctor Nikhitha

Dashboard
Doctor Schedule
Patient List

Doctor Schedule

Schedule

Add Schedule

Date *
Day *
Start Time *
End Time *
Availability *
Submit

Add Schedule

List of Patients						FILTER
scheduleId	scheduleDate	scheduleDay	startTime	endTime	bookAvail	
46	2023-10-11	Monday	11:00:00	11:30:00	notavail	
47	2023-10-11	Wednesday	11:30:00	11:45:00	available	
48	2023-10-12	Thursday	10:00:00	10:30:00	available	
						UPDATE

Appointments

Welcome Dr Doctor Nikhitha
Doctor Nikhitha

Dashboard
Doctor Schedule
Patient List

Patient List

Patient List

List of Patients
FILTER

patient Ic	Name	Password	ContactNo.	Gender	Status	Birthdate	Address	
101	Tata	hello		female		2003-09-06		
102	Neerukonda	hello		female		2003-11-06		
1234	Motamarri	1234	8688510339	female	single	2003-07-14	D no-3A-9-4, near clock tower, eluru1	


UPDATE

Patient Data

Welcome Dr Doctor Nikhitha
Doctor Nikhitha

Dashboard
Doctor Schedule
Patient List

Doctor Profile



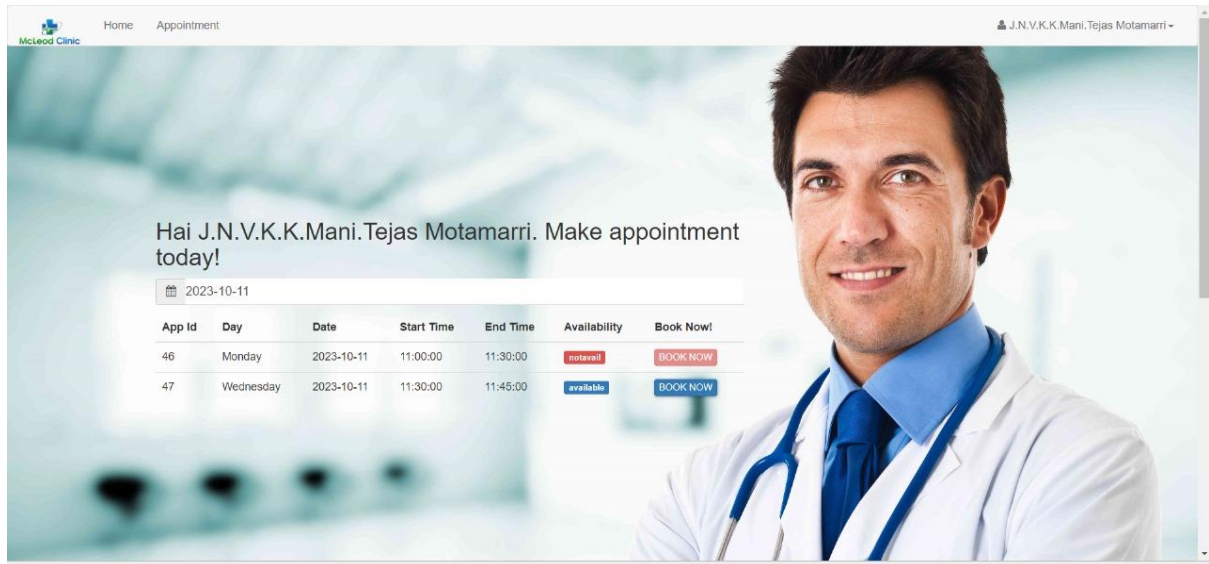
Doctor Nikhitha
Doctor

UPDATE PROFILE

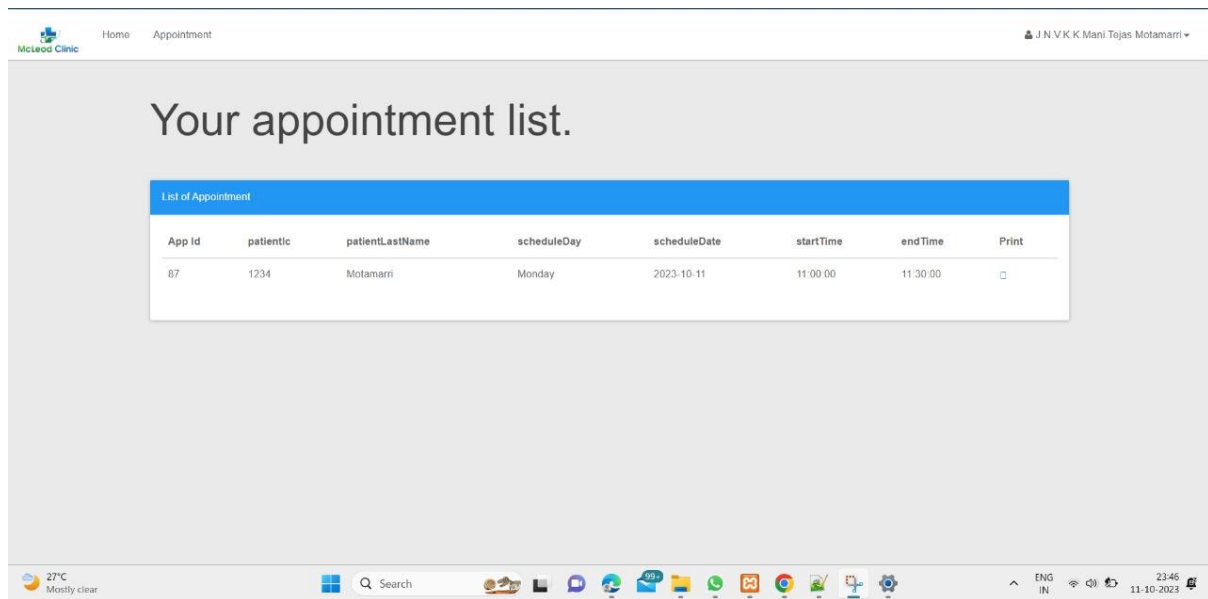
Doctor Details

Doctor ID	123
IC Number	123456789
Address	Bhimavaram
Contact Number	0173567758
Email	nikhitha@gmail.com
Birthdate	2000.11.06

Doctor Profile



Patient Home Page



Appointments

J.N.V.K.K.Mani.Tejas Motamarri

PatientMaritalStatus	single
PatientDOB	2003-07-14
PatientGender	female
PatientAddress	D.no-3A-9-4,near clock tower,eluru1
PatientPhone	8688510339
PatientEmail	tejaswinimotamarri07@gmail.com

Patient Profile

Patient Information

Patient Name: J.N.V.K.K.Mani.Tejas Motamarri
 Patient IC: 1234
 Contact Number: 8688510339
 Address: D.no-3A-9-4,near clock tower,eluru1

Appointment Information

Day: Wednesday
 Date: 2023-10-11
 Time: 11:30:00 - 11:45:00

Symptom:

Comment:

Make Appointment

Appointment Booking