HEALTHCARE MANAGEMENT

A report submitted for the course of

APPLICATION DEVELOPMENT_CLOUD COMPUTING

III B. Tech I Semester

by

M. NAVYA	**– (2211IT010069)**
N.SUMAN	**– (2211IT010078)**
P.KAVYA	**– (2211IT010084)**
R.MADHU BABU	**– (2211IT010093)**
T.V.SAI KRISHNA	**– (2211IT010109)**

Under the esteemed guidance of

Dr. P.Anitha
(Associate professor)

MALLA REDDY UNIVERSITY

DEPARTMENT OF INFORMATION TECHNOLOGY

Malla Reddy University
Maisammaguda, Dulapally, Hyderabad, Telangana 500100
www.mallareddyuniversity.ac.in

2024-25

MALLA REDDY UNIVERSITY

(Telangana State Private Universities Act No.13 of 2020 and G.O.Ms.No.14, Higher Education (UE) Department)

INFORMATION TECHNOLOGY

CERTIFICATE

This is to certify that this bonafide record of the Application Development entitled **Healthcare Management** submitted by **Ms.M.Navya (2211IT010069), Mr.N.Suman (2211IT010078) Ms.P.Kavya (2211IT010084), Mr.R.Madhu Babu (2211IT010093), Mr.T.V.Sai Krishna (2211IT010109)** of **III**-year **I** semester to the Malla Reddy University, Hyderabad. This bonafide record of work carried out by us under the guidance of our supervision. The contents of this report, in full or in parts, have not been submitted to any other Organization for the award of any Degree.

Internal Guide:
Dr. P.Anitha
(Associate professor)

Head of the Department
(GUIDE_NAME)
(DESIGNATION)

External Examiner

Date:30/11/2024

MALLA REDDY UNIVERSITY
(Telangana State Private Universities Act No.13 of 2020 and G.O.Ms.No.14, Higher Education (UE) Department)

INFORMATION TECHNOLOGY

DECLARATION

We certify that

a. The work contained in this report is original and has been done by us under the guidance of our supervisor.

b. The work has not been submitted to any other Organization for any degree.

c. We have followed the guidelines provided by the Organization in preparing the report.

d. We have conformed to the norms and guidelines given in the Ethical Code of Conduct of the Organization.

e. Whenever we have used materials (data, theoretical analysis, figures, and text) from other sources, we have given due credit to them by citing them in the text of the report and giving their details in the references.

M. NAVYA	**– (2211IT010069)**
N.SUMAN	**– (2211IT010078)**
P.KAVYA	**– (2211IT010084)**
R.MADHU BABU	**– (2211IT010093)**
T.V.SAI KRISHNA	**– (2211IT010109)**

ACKNOWLEDGEMENT

We would like take this opportunity to acknowledge everyone who has helped us in every stage of this project.

We ineffably indebted **Dr. P.Anitha, Associate Professor, Dept. of IT** for conscientious guidance and encouragement to accomplish this Application

We extremely thankful to **Mr. V. Rupesh, Assistant Professor, App Development Convener, Dept. of IT, Malla Reddy University** for his continuous support.

We wish to convey our sincere thanks to **Dr. A.V.L.N. Sujith, Head of Department, Dept. of IT, Malla Reddy University** for his continuous support.

We wish to convey our extended gratitude to **Prof. Dr. K. Harikrishna, Dean-SoE, Malla Reddy University** for his continuous support.

We would like to express our special gratitude and thanks to **Dr. VSK Reddy, Vice Chancellor, Malla Reddy University** for his continuous support.

We are also grateful to all administrative staff and the technical staff of Information Technology Department, people who provided us the useful and necessary information needed for this project.

INFORMATION TECHNOLOGY

ABSTRACT

Healthcare Management uses online servers to store and manage patient information and healthcare services. This approach makes it easier for doctors and healthcare providers to access patient records from anywhere, improving the speed and quality of care. It also allows healthcare professionals to share information and collaborate more effectively, leading to better patient outcomes. Using cloud services reduces the need for expensive hardware and maintenance, saving money for healthcare organizations. Additionally, cloud systems can easily be expanded to handle more data or users as needed. However, protecting patient information is crucial, so strong security measures are necessary. Healthcare providers must also follow laws and regulations to ensure data is handled properly, and reliable internet access is essential for these systems to work effectively. Combining cloud- based healthcare with advanced technologies like artificial intelligence can further improve care by predicting health issues early and personalizing treatments.

MALLA REDDY UNIVERSITY

(Telangana State Private Universities Act No.13 of 2020 and G.O.Ms.No.14, Higher Education (UE) Department)

INFORMATION TECHNOLOGY

CONTENTS

LIST OF FIGURES

Figure [number]	Title	Page

LIST OF ABBREVIATIONS

CHAPTER-1

Introduction:

Our project Healthcare Management system includes registration of patients, storing their details into the system, and also booking their appointments with doctors.our software has the facility to give a unique id for every patient and stores the details of every patient and the staff automatically. user can search availability of a doctor and the details of a patient using the id. the Healthcare Management System can be entered using a username and password. it is accessible either by an administrator or receptionist.it is having mainly two modules. one is at Administration Level and other one is of user I.e. of patients and doctors.The Application maintains authentication in order to access the application. Aministrator task includes managing doctors information, patient's information.The Patient modules include checking appointments, prescription. user can also pay doctor's Fee online.

1.1 Context and Significance:

Healthcare management involves overseeing the operations, financials, and strategic direction of healthcare organizations to ensure the efficient delivery of patient care. Key areas include strategic planning, financial management, human resources, quality improvement, compliance with regulations, and integrating new technologies. Healthcare managers are responsible for optimizing resource use, ensuring regulatory compliance, and fostering a culture of patient safety and high-quality care. They also manage the workforce, focusing on recruitment, training, and retention of healthcare professionals, as well as addressing employee satisfaction and burnout.

The significance of healthcare management lies in its ability to improve patient outcomes, reduce costs, and maintain a balance between quality care and financial sustainability. By effectively managing resources, implementing innovative solutions, and adapting to changes in healthcare policy and technology, healthcare managers play a crucial role in enhancing both patient experiences and organizational performance. Additionally, their leadership ensures that healthcare organizations comply with laws and regulations, mitigate risks, and contribute positively to public health. Ultimately, healthcare management is vital in ensuring that healthcare systems are efficient, accessible, and capable of meeting the growing demands of diverse populations.

1.2 Objectives:

- Enhance Patient Care: Ensure high-quality, safe, and effective care for patients by implementing best practices and standards.
- Improving Patient Outcomes: Ensuring that patients receive high-quality care that leads to better health outcomes and enhances overall well-being.
- Improving Patient Access: Ensure that healthcare services are available and accessible to all patients, addressing barriers related to cost, location, and other factors.
- Facilitating Innovation: Encouraging the adoption of new technologies, treatments, and processes that can improve care and operational efficiency.
-
- Enhancing Patient Experience: Improving the overall experience of patients, including aspects like communication, comfort, and respect, to foster patient satisfaction and engagement.

1.3 Key Features:

1. Remote Accessibility:
 -Access patient records and healthcare services from anywhere.
 - Improves the speed and quality of care by enabling quick access to information.

2. Collaboration:
 - Allows healthcare professionals to share information and work together more effectively.
 - Leads to better patient outcomes through improved teamwork.

3. Cost-Efficiency:
 - Reduces the need for expensive hardware and ongoing maintenance.
 - Saves money for healthcare organizations.

4. Scalability:
 - Easily expand systems to handle more data or users as needed.
 - Accommodates growing healthcare demands without significant infrastructure changes.

5. Security:
 - Protecting patient information is crucial.
 - Strong security measures such as encryption, access controls, and regular audits are necessary.

6. Compliance:

- Ensures data handling follows laws and regulations, like HIPAA in the United States or GDPR in Europe.
- Helps avoid legal issues and maintain patient trust.

7. Internet Dependence:
- Requires reliable internet access to function effectively.
- Downtime or connectivity issues can impact access to patient data and services.

8. Integration with AI:
- Combines cloud-based systems with advanced technologies like artificial intelligence.
- Improves care by predicting health issues early and personalizing treatments.

CHAPTER-2

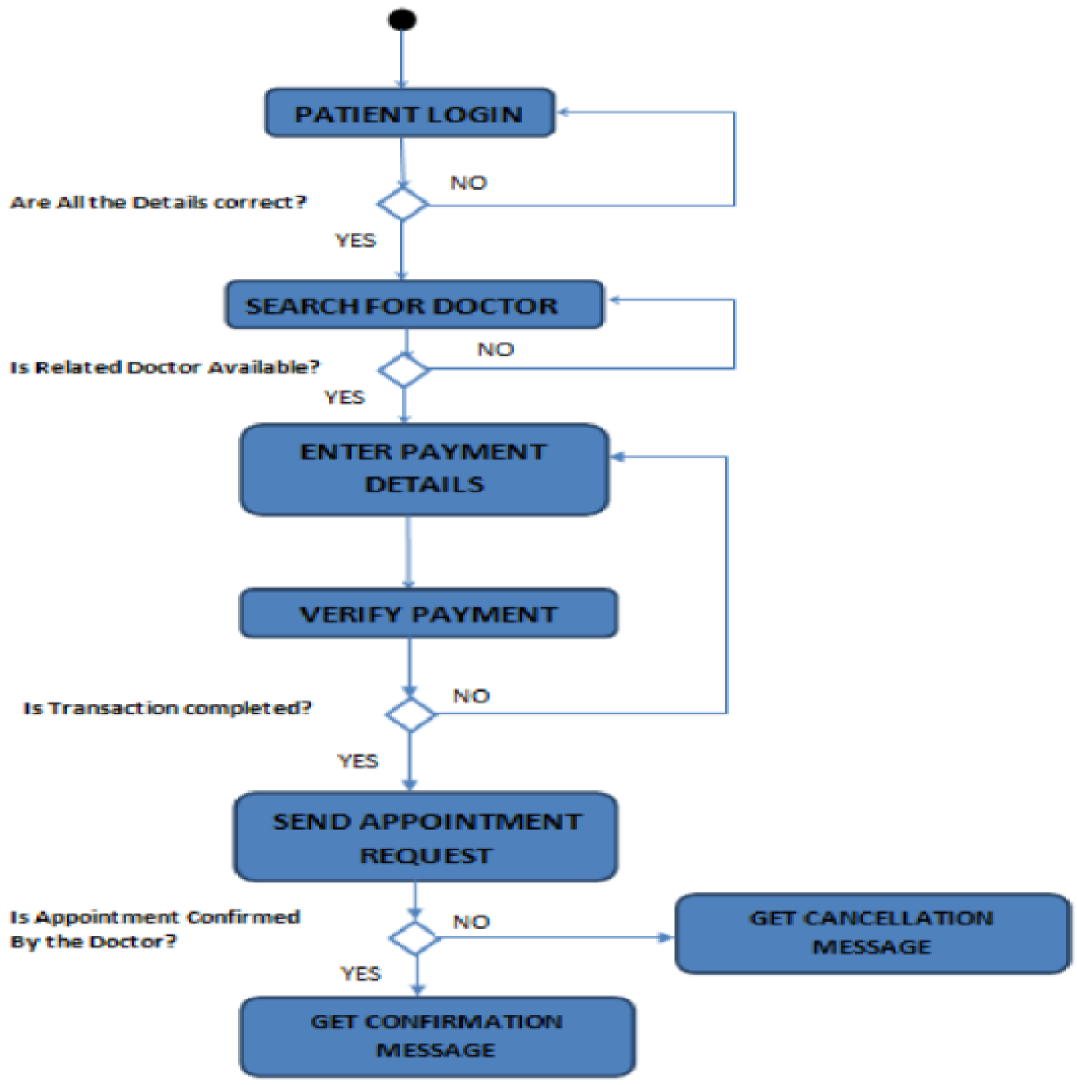
Review of Relevant Literature:

The literature on healthcare management emphasizes the critical role of leadership, strategic planning, and operational efficiency in enhancing healthcare delivery. Studies have highlighted the importance of healthcare administrators in navigating complex healthcare systems, balancing financial constraints with the need for high-quality patient care. Research on strategic management in healthcare underscores the integration of data analytics, evidence-based practices, and technology to streamline operations, reduce costs, and improve patient outcomes. Key themes also include workforce management, where literature examines how leadership practices, staff satisfaction, and employee retention directly impact patient care quality and organizational performance. Moreover, the shift toward patient-centered care models and the increasing focus on interprofessional collaboration are recurrent themes that reflect evolving healthcare priorities.

The growing body of literature also addresses the challenges of healthcare management, such as regulatory compliance, healthcare policy changes, and the adoption of new technologies. Studies have explored the impact of healthcare reforms, such as the Affordable Care Act, on organizational operations and the financial health of healthcare systems. Additionally, research on health information technology (HIT) and electronic health records (EHR) systems shows their transformative role in improving data management, reducing errors, and enhancing decision-making. However, scholars also note barriers to successful technology implementation, including resistance to change and the high costs of integration. Overall, the literature reveals a dynamic field where healthcare managers must continually adapt to new challenges while focusing on delivering quality care and improving system efficiency.

CHAPTER-3

Methodology:



3.1 Problem Definition:

In healthcare management, the problem definition revolves around identifying and addressing challenges that impact the efficiency, quality, and accessibility of healthcare services. One key problem is the *rising cost of healthcare, which strains both healthcare organizations and patients. Managing costs while maintaining high-quality care is a central challenge for administrators, as they must balance financial sustainability with patient satisfaction and clinical outcomes. Another significant issue is **resource allocation*: healthcare managers must optimize the use of limited resources, including staff, equipment, and facilities, to meet growing patient demand without compromising service quality.

Additionally, workforce shortages and burnout among healthcare professionals present ongoing difficulties in healthcare management. These challenges lead to high turnover rates, lower employee morale, and potential risks to patient care. Another pressing problem is *patient safety and quality of care, with healthcare systems striving to reduce errors, enhance safety protocols, and ensure effective treatment outcomes. The **integration of technology* in healthcare, such as electronic health records (EHR) and telemedicine, while offering great potential for improvement, also presents problems of high initial costs, technological resistance, and data security concerns. Overall, healthcare management faces the complex task of addressing these interconnected issues while adapting to changes in healthcare policy, technological advancements, and evolving patient needs.

3.2 Objectives of Application:

Improving Operational Efficiency: Applications like Electronic Health Records (EHR), Hospital Information Systems (HIS), and practice management software streamline administrative tasks, reduce paperwork, and improve coordination between departments. This leads to faster processing, reduced wait times for patients, and more efficient use of resources.

Enhancing Patient Care Quality and Safety: Healthcare management applications aim to reduce medical errors, ensure better clinical decision-making, and monitor patient outcomes in real-time. Tools such as clinical decision support systems (CDSS) and patient safety software help healthcare professionals make informed decisions, prevent adverse events, and improve patient care delivery.

Optimizing Resource Allocation: Applications like workforce management tools and resource scheduling software allow healthcare organizations to allocate staff, equipment, and facilities effectively. This ensures that resources are used efficiently and that patients receive timely care, while also minimizing costs.

Supporting Financial Management: Healthcare management applications, such as revenue cycle management (RCM) tools, help organizations manage billing, coding, and insurance claims. These applications improve cash flow, ensure compliance with regulatory standards, and reduce billing errors, which directly impacts the financial health of healthcare institutions.

Facilitating Data-Driven Decision Making: With the integration of big data analytics and artificial intelligence, healthcare management applications provide actionable insights to administrators. These tools allow for better forecasting, performance measurement, and decision-making, which can lead to improved operational strategies and patient outcomes.

Ensuring Compliance and Regulatory Adherence: Healthcare management applications help organizations comply with evolving healthcare regulations, such as HIPAA, ACA, and other regional health policies. These tools ensure that patient data is securely stored, privacy is maintained, and healthcare providers meet necessary standards and regulations.

Enhancing Communication and Collaboration: Applications like telemedicine platforms, patient portals, and secure messaging systems facilitate communication among healthcare teams, between providers and patients, and among patients themselves. These systems improve care coordination, enhance patient engagement, and support interdisciplinary collaboration in treatment planning.

3.1 System Design:

Patient Care Systems: These systems include Electronic Health Records (EHR), which store all patient information digitally, making it easy for doctors and nurses to access and update patient details quickly. This helps improve the quality of care and reduces errors. Other tools like clinical decision support systems give healthcare providers suggestions based on patient data to help them make better decisions.

equipment, hospital beds, and staff schedules. For example, workforce management tools ensure that the right number of doctors and nurses are available at the right time, while inventory management systems track medical supplies, making sure the hospital doesn't run out of essential items.

Resource and Staff Management: Systems also help manage healthcare resources like medical

Financial and Billing Systems: Healthcare management systems also help in managing finances, from billing patients to processing insurance claims. These systems ensure that hospitals are paid for the services they provide, track expenses, and help in reducing errors in billing.

Telemedicine and Monitoring: With telemedicine systems, patients can have virtual appointments with doctors, especially useful in remote areas. Remote monitoring tools allow healthcare providers to keep track of patients' health conditions from a

distance, ensuring continuous care.

Data Security: Since healthcare deals with sensitive personal information, system design also focuses on securing data, ensuring that only authorized personnel can access patient records, and that all data is protected from cyber threats.

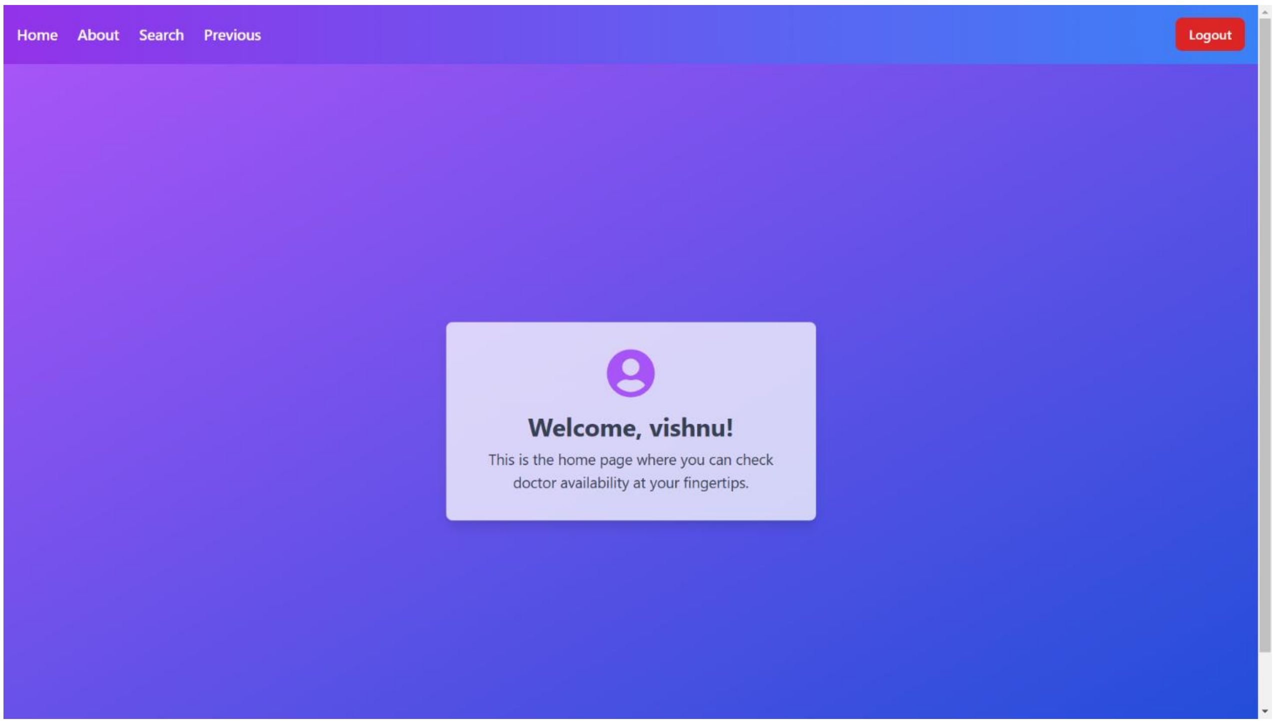
3.4 User Interface:

In healthcare management, the *user interface (UI)* is crucial for ensuring that healthcare professionals, patients, and administrators can efficiently interact with digital systems and tools. For medical staff, a well-designed UI in systems such as Electronic Health Records (EHR) and Clinical Decision Support Systems (CDSS) enables quick access to patient data, medical histories, and test results. The interface should be intuitive, allowing healthcare providers to easily input data, search for patient information, and make informed decisions using clear and organized displays of clinical information. In administrative systems, such as scheduling and patient management, the UI should streamline processes like appointment bookings, billing, and patient flow management, reducing wait times and improving operational efficiency.

For patients, user-friendly UI design is equally important, especially in tools such as telemedicine platforms and mobile health applications. Patients need simple, easy-to-navigate interfaces that allow them to schedule appointments, access their medical information, and communicate with healthcare providers. The UI must ensure secure logins and protect sensitive health data while being accessible to individuals with varying levels of technological expertise. In addition to ease of use, healthcare management UIs must comply with strict security and privacy regulations, ensuring that only authorized personnel have access to sensitive patient information. Ultimately, an effective UI in healthcare management enhances user experience, reduces errors, and contributes to improved patient care and operational efficiency.

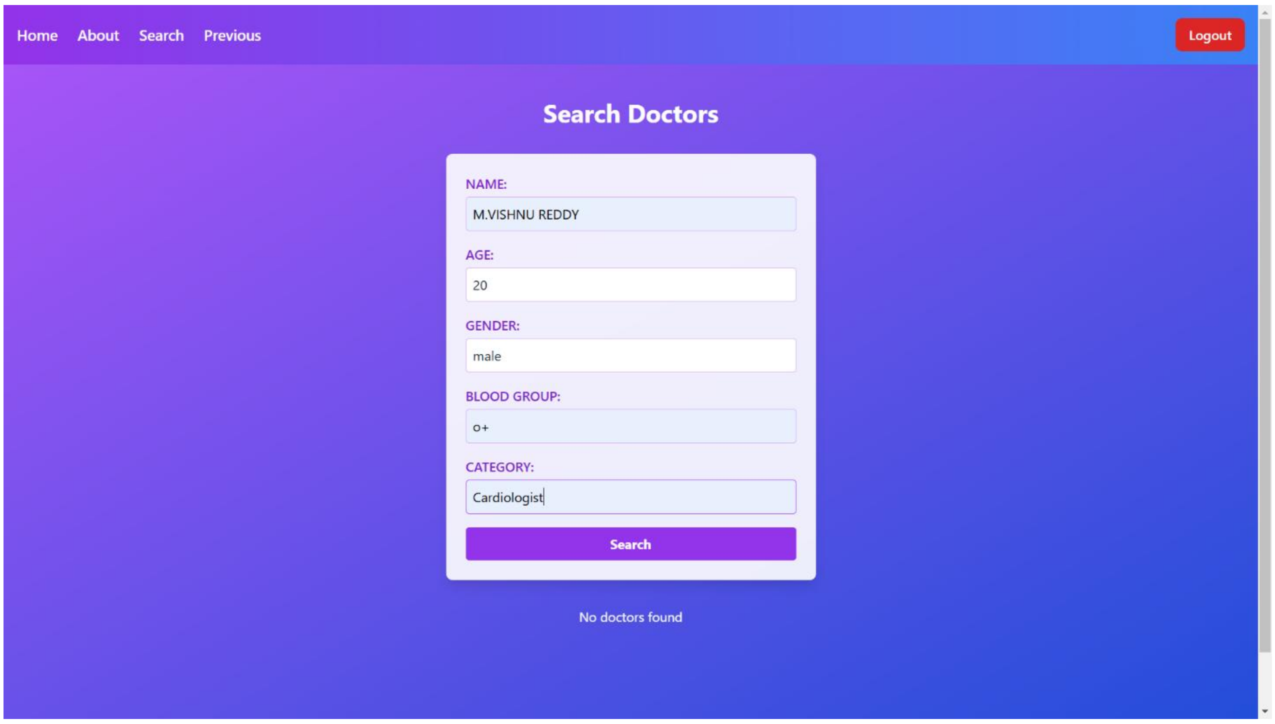
CHAPTER-4

Results:



Welcome, vishnu!

This is the home page where you can check doctor availability at your fingertips.



Search Doctors

NAME: M.VISHNU REDDY

AGE: 20

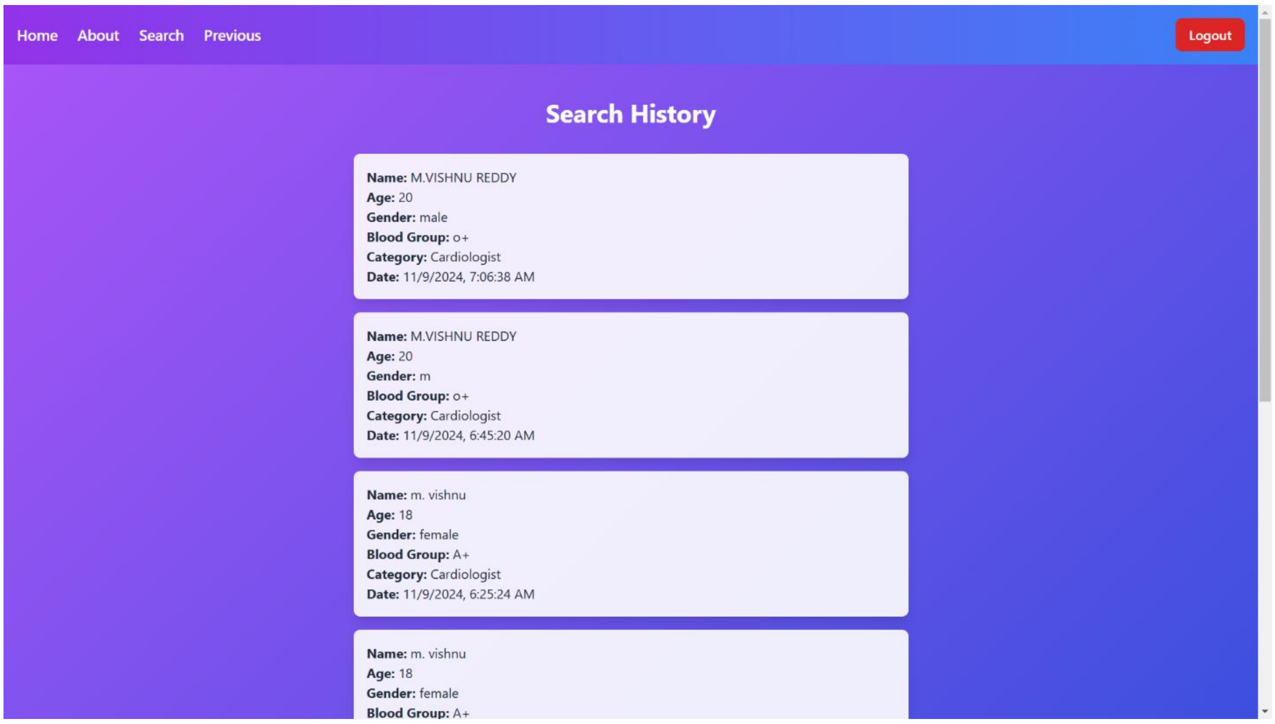
GENDER: male

BLOOD GROUP: o+

CATEGORY: Cardiologist

Search

No doctors found



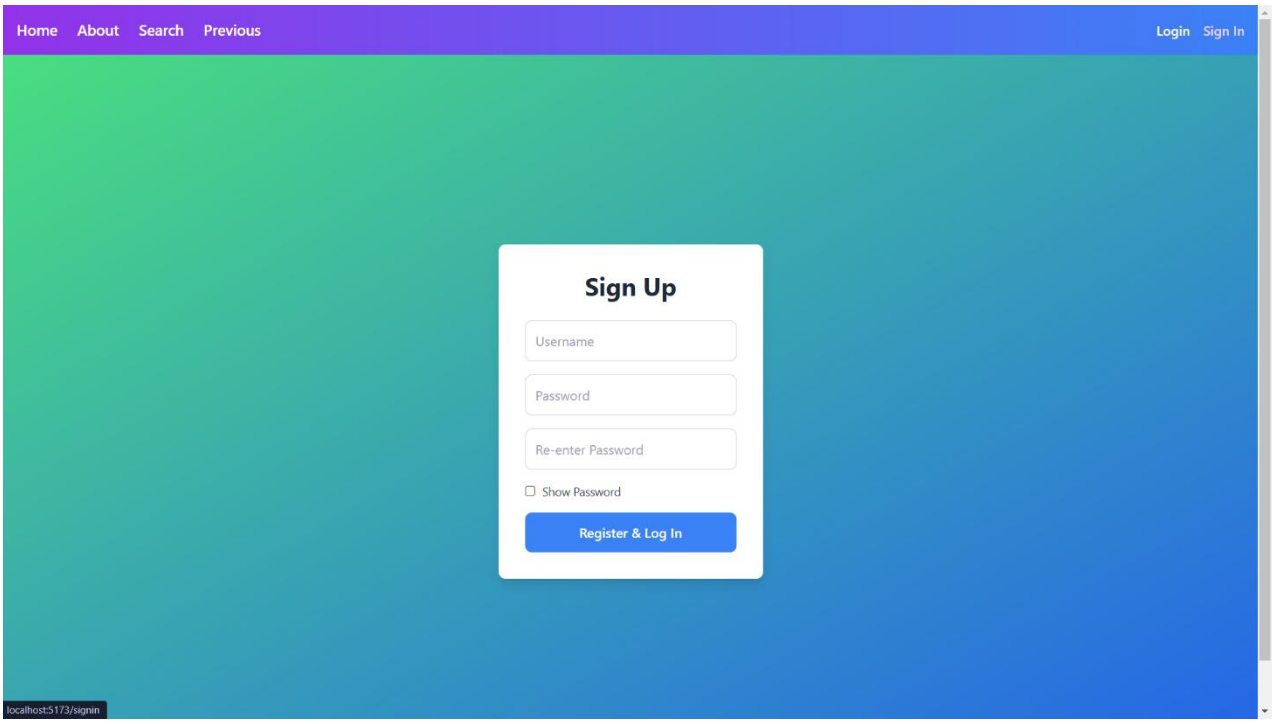
Search History

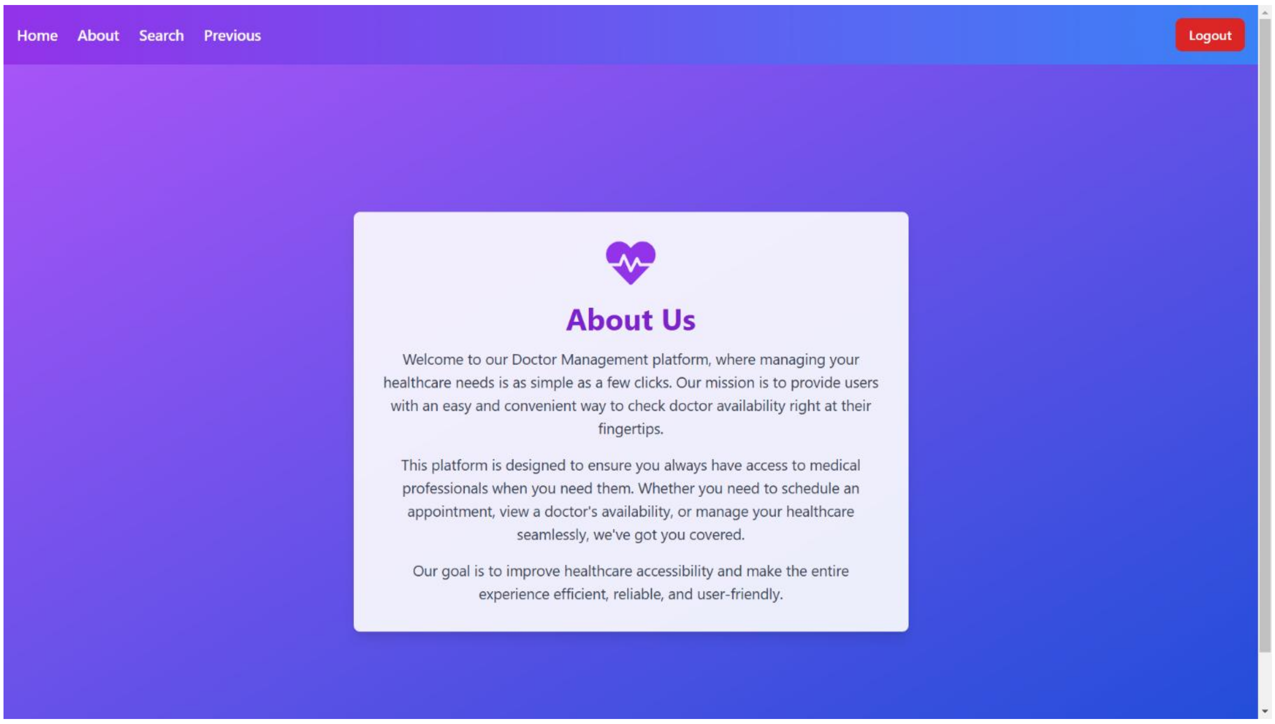
Name: M.VISHNU REDDY
Age: 20
Gender: male
Blood Group: o+
Category: Cardiologist
Date: 11/9/2024, 7:06:38 AM

Name: M.VISHNU REDDY
Age: 20
Gender: m
Blood Group: o+
Category: Cardiologist
Date: 11/9/2024, 6:45:20 AM

Name: m. vishnu
Age: 18
Gender: female
Blood Group: A+
Category: Cardiologist
Date: 11/9/2024, 6:25:24 AM

Name: m. vishnu
Age: 18
Gender: female
Blood Group: A+



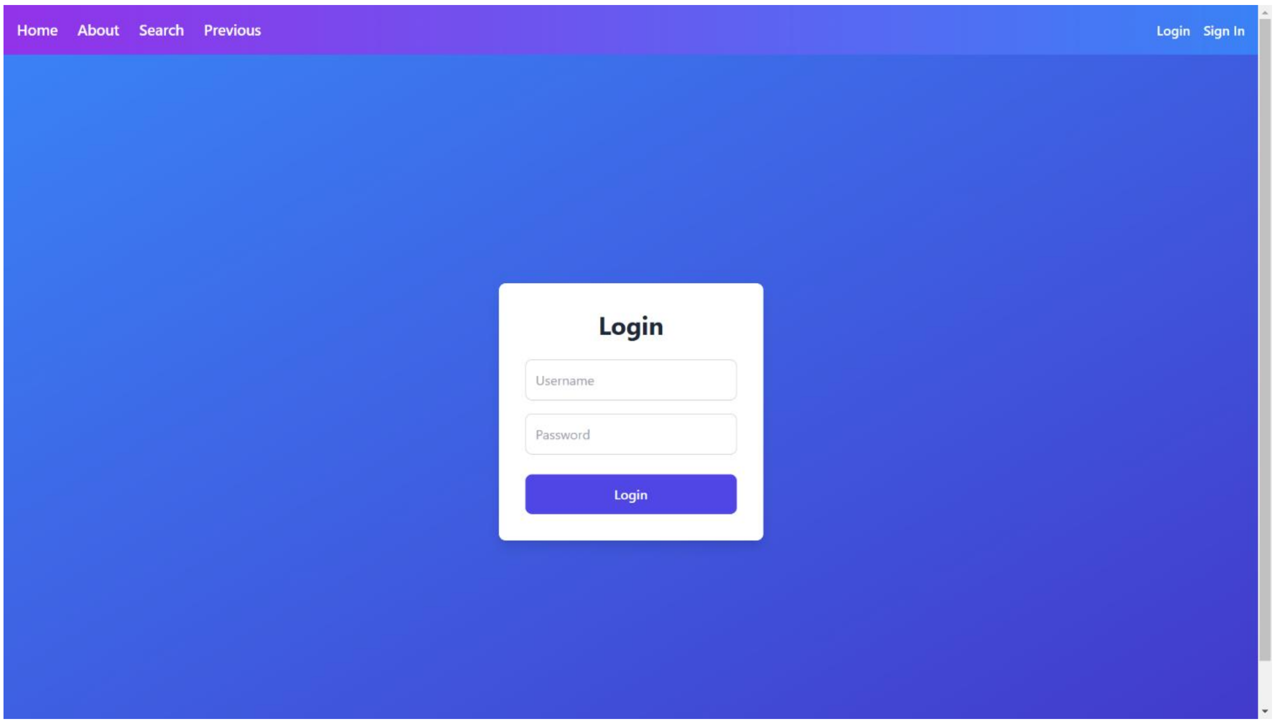


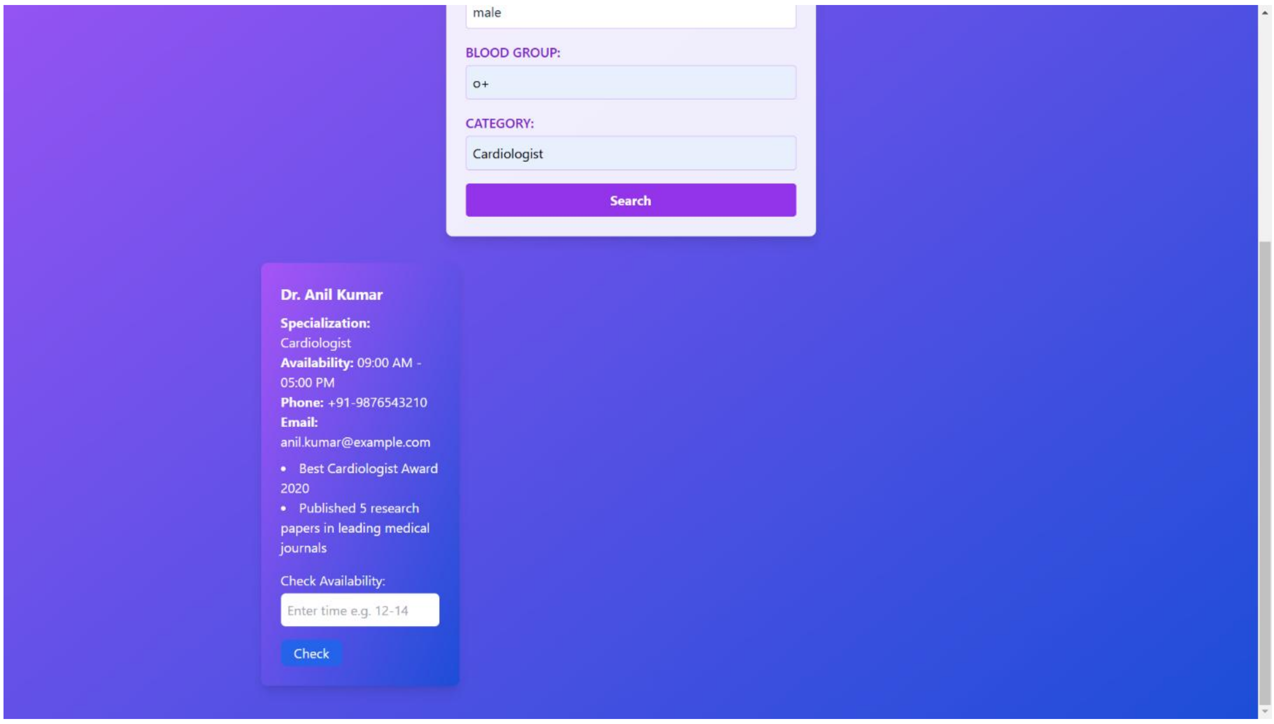
About Us

Welcome to our Doctor Management platform, where managing your healthcare needs is as simple as a few clicks. Our mission is to provide users with an easy and convenient way to check doctor availability right at their fingertips.

This platform is designed to ensure you always have access to medical professionals when you need them. Whether you need to schedule an appointment, view a doctor's availability, or manage your healthcare seamlessly, we've got you covered.

Our goal is to improve healthcare accessibility and make the entire experience efficient, reliable, and user-friendly.




male

BLOOD GROUP:
o+

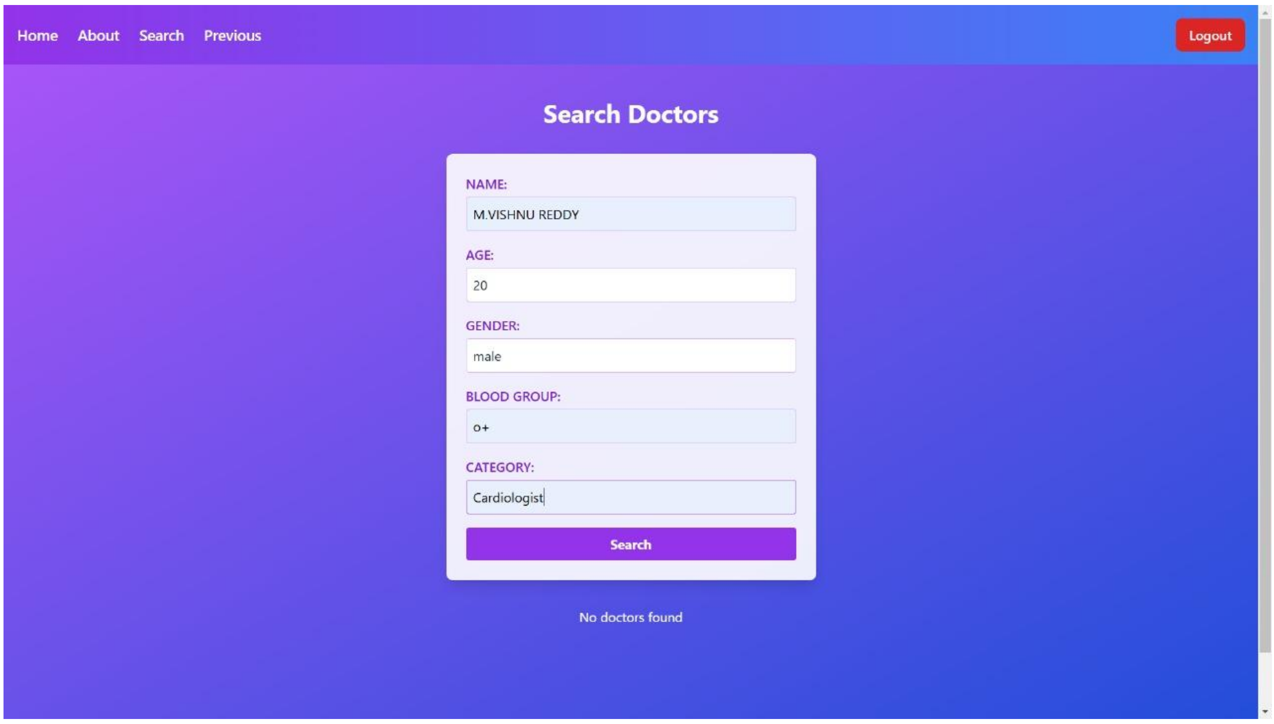
CATEGORY:
Cardiologist

Search

Dr. Anil Kumar
Specialization: Cardiologist
Availability: 09:00 AM - 05:00 PM
Phone: +91-9876543210
Email: anil.kumar@example.com

- Best Cardiologist Award 2020
- Published 5 research papers in leading medical journals

Check Availability:
Enter time e.g. 12-14
Check


Home About Search Previous Logout

Search Doctors

NAME:
M.VISHNU REDDY

AGE:
20

GENDER:
male

BLOOD GROUP:
o+

CATEGORY:
Cardiologist

Search

No doctors found

Discussions:

Healthcare management is the backbone of efficient healthcare delivery, e ncompassing the administration, organization, and coordination of healthc are services to ensure optimal patient care. Effective healthcare managem ent involves strategic planning, resource allocation, and maintaining high standards of care,integrating advanced technology to streamline processes , reduce costs, and improve patient outcomes. from managing patient reco rds to optimizing staff workflows and ensuring regulatory compliance, he althcare management aims to create a seamless healthcare experience that is accessible, affordable, and of Highquality. By leveraging datadriven in sights and patientcentric approaches, healthcare managers can significantl y enhance the overall performance of healthcare systems.

CHAPTER-5

Conclusion:

Healthcare management is the backbone of effective healthcare systems, overseeing the complex web of administrative tasks, policies, and strategies that ensure high-quality, patient-centered care. Healthcare Management is essential for delivering high-quality patient care, optimizing resources, and ensuring the overall success of healthcare organizations. Since we are entering details of the patient's electronically in the "Healthcare Management" data will be secured. Using this application we can retrieve patient history with a single click. Thus processing information will be faster. It guarantee accurate maintenance of patient details. It easily reduces the book keeping task and thus reduces the human effort and increases accuracy speed. It's the process of keeping healthcare systems effective, efficient, and focused on providing quality care to patients. Healthcare managers play a critical role in shaping the future of healthcare delivery.

Future Scope:

The future scope of healthcare management is vast and promising, with o ngoing advancements in technology poised to revolutionize the field. As t elemedicine becomes more prevalent, healthcare Management will need t o integrate virtual consultations and remote monitoring seamlessly. The ri se of artificial intelligence and machine learning will enable predictive an alytics, improving patient outcomes by anticipating healthcare needs and personalizing treatments. Blockchain technology could enhance data secu rity and interoperability, ensuring that patient records are both accessible and protected. Additionally, with a growing emphasis on preventive care and wellness, healthcare management will increasingly focus on holistic a pproaches to patient care, emphasizing early intervention and lifestyle ma nagement. In short, the future of healthcare management is geared toward creating more efficient, patientcentric, and technologically advanced heal thcare systems.

REFERENCES

https://www.researchgate.net/publication/377557804_Health_Care_Management_System
www.smilehealthdigital.com

APPENDIX

Source Code:

LOGIN CODE:

```
import { useState } from 'react';
import { useUser } from '../context/UserContext';

const Login = () => {
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');
  const { loginUser } = useUser();
  const [errorMessage, setErrorMessage] = useState('');

  const handleLogin = async (e) => {
    e.preventDefault();

    // Client-side validation
    if (!username || !password) {
      setErrorMessage('Both fields are required.');
      return;
    }

    setErrorMessage(''); // Clear previous error message
    await loginUser(username, password);
  };

  return (
    <div className="flex items-center justify-center min-h-screen bg-gradient-to-br from-blue-500 to-indigo-700">
      <form onSubmit={handleLogin} className="bg-white p-8 rounded-lg shadow-lg w-80">
        <h1 className="text-3xl font-bold mb-6 text-center text-gray-800">Login</h1>
        {errorMessage && <p className="text-red-600 text-sm text-center mb-4">{errorMessage}</p>}

        <input
          type="text"
          placeholder="Username"
          value={username}
          onChange={(e) => setUsername(e.target.value)}
          className="border border-gray-300 p-3 mb-4 w-full rounded-lg focus:outline-none focus:ring-2 focus:ring-indigo-
400"
        />
        <input
          type="password"
          placeholder="Password"
          value={password}
          onChange={(e) => setPassword(e.target.value)}
          className="border border-gray-300 p-3 mb-6 w-full rounded-lg focus:outline-none focus:ring-2 focus:ring-indigo-
400"
        />

        <button
          type="submit"
          className="bg-indigo-600 text-white py-3 rounded-lg w-full font-semibold hover:bg-indigo-700 transition-colors"
        >
          Login
        </button>
      </form>
    </div>
  );
};

export default Login;
```

SIGNUP CODE:

```
import { useState } from 'react';
import { useUser } from '../context/UserContext';

const SignIn = () => {
   const [username, setUsername] = useState('');
   const [password, setPassword] = useState('');
   const [reEnterPassword, setReEnterPassword] = useState('');
   const [showPassword, setShowPassword] = useState(false);
   const { registerUser } = useUser();
   const [errorMessage, setErrorMessage] = useState('');

   const handleSignIn = async (e) => {
      e.preventDefault();

      // Client-side validation
      if (!username || !password || !reEnterPassword) {
         setErrorMessage('All fields are required.');
         return;
      }

      if (password !== reEnterPassword) {
         setErrorMessage('Passwords do not match.');
         return;
      }

      setErrorMessage(''); // Clear previous error message
      await registerUser(username, password);
   };

   return (
      <div className="flex items-center justify-center min-h-screen bg-gradient-to-br from-green-400 to-
blue-600">
         <form onSubmit={handleSignIn} className="bg-white p-8 rounded-lg shadow-lg w-80">
            <h1 className="text-3xl font-bold mb-6 text-center text-gray-800">Sign Up</h1>
            {errorMessage && <p className="text-red-600 text-sm text-center mb-
4">{errorMessage}</p>}

            <input
               type="text"
               placeholder="Username"
               value={username}
               onChange={(e) => setUsername(e.target.value)}
               className="border border-gray-300 p-3 mb-4 w-full rounded-lg focus:outline-none
focus:ring-2 focus:ring-blue-400"
            />
            <input
               type={showPassword ? "text" : "password"}
               placeholder="Password"
            value={password}
            onChange={(e) => setPassword(e.target.value)}
            className="border border-gray-300 p-3 mb-4 w-full rounded-lg focus:outline-none
focus:ring-2 focus:ring-blue-400"
          />
          <input
            type={showPassword ? "text" : "password"}
            placeholder="Re-enter Password"
            value={reEnterPassword}
            onChange={(e) => setReEnterPassword(e.target.value)}
            className="border border-gray-300 p-3 mb-4 w-full rounded-lg focus:outline-none
focus:ring-2 focus:ring-blue-400"
          />

          <div className="flex items-center mb-4">
            <input
              type="checkbox"
              id="showPassword"
              checked={showPassword}
              onChange={() => setShowPassword(!showPassword)}
              className="mr-2"
            />
            <label htmlFor="showPassword" className="text-sm text-gray-700">Show
Password</label>
          </div>

          <button
            type="submit"
            className="bg-blue-500 text-white py-3 rounded-lg w-full font-semibold hover:bg-blue-600
transition-colors"
          >
            Register & Log In
          </button>
        </form>
      </div>
  );
};

export default SignIn;
```

HOME CODE:

```
import { useUser } from '../context/UserContext'; // Adjust the path as needed
import { FaUserCircle } from 'react-icons/fa';

const Home = () => {
  const { user } = useUser();
  return (
    <div className="flex flex-col items-center justify-center h-screen bg-gradient-to-br from-purple-500
to-blue-700 text-gray-700">
      <div className="p-8 bg-opacity-75 bg-white rounded-lg shadow-lg text-center max-w-md">
        <FaUserCircle className="text-6xl text-purple-500 mx-auto mb-4" />
        <h1 className="text-3xl font-bold mb-2">Welcome, {user?.username || 'Guest'}!</h1>
        <p className="text-lg">This is the home page where you can check doctor availability at your
fingertips.</p>
      </div>
    </div>
    );
};

export default Home;
```

NAVBAR CODE:

```
import { Link } from 'react-router-dom';
import { useUser } from '../context/UserContext';

const Navbar = () => {
   const { user, logoutUser } = useUser();

   return (
      <nav className="bg-gradient-to-r from-purple-600 to-blue-500 text-white flex justify-between items-
center p-4 shadow-md">
         {/* Left Side: Brand/Logo or Components */}
         <div className="flex items-center space-x-6 font-semibold text-lg">
            <Link to="/home" className="hover:text-purple-200 transition duration-200">Home</Link>
            <Link to="/about" className="hover:text-purple-200 transition duration-200">About</Link>
            <Link to="/search" className="hover:text-purple-200 transition duration-200">Search</Link>
            <Link to="/history" className="hover:text-purple-200 transition duration-
200">Previous</Link>
         </div>

         {/* Right Side: User Actions */}
         <div className="flex items-center space-x-4">
            {user ? (
               <>
                  <button
                     onClick={logoutUser}
                     className="bg-red-600 hover:bg-red-700 text-white font-semibold py-2 px-4 rounded-lg
transition duration-200"
                  >
                     Logout
                  </button>
               </>
            ) : (
               <>
                  <Link to="/login" className="hover:text-purple-200 transition duration-200 font-
medium">Login</Link>
                  <Link to="/signin" className="hover:text-purple-200 transition duration-200 font-
medium">Sign In</Link>
               </>
            )}
         </div>
      </nav>
   );
};

export default Navbar;
```

ABOUTUS CODE:

```
import { FaHeartbeat } from 'react-icons/fa';

const About = () => (
  <div className="flex flex-col items-center justify-center min-h-screen bg-gradient-to-br from-purple-
500 to-blue-700 text-gray-800">
    <div className="p-8 bg-opacity-90 bg-white rounded-lg shadow-lg text-center max-w-2xl">
      <FaHeartbeat className="text-6xl text-purple-600 mx-auto mb-4" />
      <h1 className="text-4xl font-bold mb-4 text-purple-700">About Us</h1>
      <p className="text-lg mb-4 text-gray-700">
        Welcome to our Doctor Management platform, where managing your healthcare needs is as
simple as a few clicks. Our mission is to provide users with an easy and convenient way to check doctor
availability right at their fingertips.
      </p>
      <p className="text-lg mb-4 text-gray-700">
        This platform is designed to ensure you always have access to medical professionals when you
need them. Whether you need to schedule an appointment, view a doctor's availability, or manage your
healthcare seamlessly, we've got you covered.
      </p>
      <p className="text-lg text-gray-700">
        Our goal is to improve healthcare accessibility and make the entire experience efficient, reliable,
and user-friendly.
      </p>
    </div>
  </div>
);

export default About;
```

DOCTOR CARD CODE:

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

const DoctorCard = ({ doctor }) => {
  const [usertime, setUsertime] = useState('');
  const [availabilityMessage, setAvailabilityMessage] = useState('');

  const apiUrl = import.meta.env.VITE_API_URL || 'http://localhost:5000';

  const handleCheckAvailability = async () => {
    try {
      const response = await axios.get(`${apiUrl}/user/checkavailability`, {
        params: {
          usertime,
          doctorid: doctor._id
        },
        withCredentials: true
      });

      setAvailabilityMessage(response.data.message);
    } catch (err) {
      setAvailabilityMessage(err.response?.data?.message || 'Error checking availability');
    }
  };

  return (
    <div className="bg-gradient-to-br from-purple-500 to-blue-700 text-white rounded-lg p-6 w-60
shadow-lg">
      <h3 className="text-lg font-bold mb-2">{doctor.name}</h3>
      <p><strong>Specialization:</strong> {doctor.specialization}</p>
```

```
          <p><strong>Availability:</strong> {doctor.availability_time}</p>
          <p><strong>Phone:</strong> {doctor.contact_details.phone}</p>
          <p><strong>Email:</strong> {doctor.contact_details.email}</p>

          {doctor.achievements.length > 0 && (
            <ul className="mt-2 list-disc list-inside">
              {doctor.achievements.map((achievement, index) => (
                <li key={index}>{achievement}</li>
              ))}
            </ul>
          )}

          <div className="mt-4">
            <label className="block mb-2">
              Check Availability:
              <input
                type="text"
                value={usertime}
                onChange={(e) => setUsertime(e.target.value)}
                placeholder="Enter time e.g. 12-14"
                className="mt-1 w-full p-2 text-gray-900 rounded-md"
              />
            </label>
            <button
              onClick={handleCheckAvailability}
              className="mt-2 bg-blue-600 hover:bg-blue-700 text-white py-1 px-4 rounded-lg"
            >
              Check
            </button>
          </div>

          {availabilityMessage && <p className="mt-2 text-yellow-200">{availabilityMessage}</p>}
        </div>
    );
};

export default DoctorCard;
```

SEARCH DOCTORS CODE:

```
import React, { useState } from 'react';
import axios from 'axios';
import DoctorCard from './DoctorCard';

const SearchDoctors = () => {
    const [searchCriteria, setSearchCriteria] = useState({
        name: '',
        age: '',
        gender: '',
        blood_group: '',
        category: ''
    });
    const [doctors, setDoctors] = useState([]);
    const [loading, setLoading] = useState(false);
    const [error, setError] = useState('');

    const apiUrl = import.meta.env.VITE_API_URL || 'http://localhost:5000';
```

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

  const handleSearch = async () => {
    setLoading(true);
    setError('');

    try {
      const response = await axios.post(`${apiUrl}/user/search`, searchCriteria, { withCredentials: true
});
      setDoctors(response.data.data || []);
    } catch (err) {
      setError(err.response?.data?.message || 'Error searching for doctors');
    } finally {
      setLoading(false);
    }
  };

  return (
    <div className="flex flex-col items-center min-h-screen bg-gradient-to-br from-purple-500 to-blue-
700 text-gray-800 py-10 px-4">
      <h1 className="text-3xl font-bold text-white mb-8">Search Doctors</h1>
      <div className="bg-white bg-opacity-90 p-6 rounded-lg shadow-lg max-w-md w-full space-y-4
mb-8">
        {['name', 'age', 'gender', 'blood_group', 'category'].map((field) => (
          <label key={field} className="block">
            <span className="text-purple-700 font-semibold">{field.replace('_', '
').toUpperCase()}:</span>
            <input
              type={field === 'age' ? 'number' : 'text'}
              name={field}
              value={searchCriteria[field]}
              onChange={handleInputChange}
              placeholder={`Enter ${field.replace('_', ' ')}`}
              className="mt-1 block w-full p-2 border border-purple-300 rounded focus:outline-none
focus:border-purple-500"
            />
          </label>
        ))}
        <button
          onClick={handleSearch}
          disabled={loading}
          className="w-full py-2 px-4 mt-4 text-white font-bold bg-purple-600 rounded hover:bg-
purple-700 focus:outline-none"
        >
          {loading ? 'Searching...' : 'Search'}
        </button>
      </div>

      {error && <p className="text-red-500 mb-4">{error}</p>}

      <div className="w-full max-w-4xl">
        {doctors.length > 0 ? (
          <div className="grid gap-4 grid-cols-1 sm:grid-cols-2 lg:grid-cols-3">
```

```
            {doctors.map((doctor) => (
              <DoctorCard key={doctor._id} doctor={doctor} />
            ))}
          </div>
        ) : (
          !loading && <p className="text-center text-white">No doctors found</p>
        )}
      </div>
    </div>
  );
};

export default SearchDoctors;
```

HISTORY CODE:

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

const History = () => {
  const [history, setHistory] = useState([]);
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState('');

  const apiUrl = import.meta.env.VITE_API_URL || 'http://localhost:5000';

  useEffect(() => {
    const fetchHistory = async () => {
      setLoading(true);
      setError('');

      try {
        const response = await axios.get(`${apiUrl}/user/history`, { withCredentials: true });
        setHistory(response.data.data || []);
      } catch (err) {
        setError(err.response?.data?.message || 'Error fetching search history');
      } finally {
        setLoading(false);
      }
    };

    fetchHistory();
  }, [apiUrl]);

  return (
    <div className="flex flex-col items-center min-h-screen bg-gradient-to-br from-purple-500 to-blue-
700 py-10 px-4">
      <h1 className="text-3xl font-bold text-white mb-8">Search History</h1>

      {loading && <p className="text-white">Loading...</p>}
      {error && <p className="text-red-500 mb-4">{error}</p>}

      {history.length > 0 ? (
        <ul className="w-full max-w-2xl space-y-4">
          {history.map((record, index) => (
            <li key={index} className="bg-white bg-opacity-90 p-4 rounded-lg shadow-lg">
```

```
              <p className="text-gray-800"><strong>Name:</strong>
{record.searchCriteria.name}</p>
              <p className="text-gray-800"><strong>Age:</strong> {record.searchCriteria.age}</p>
              <p className="text-gray-800"><strong>Gender:</strong>
{record.searchCriteria.gender}</p>
              <p className="text-gray-800"><strong>Blood Group:</strong>
{record.searchCriteria.blood_group}</p>
              <p className="text-gray-800"><strong>Category:</strong>
{record.searchCriteria.category}</p>
              <p className="text-gray-800"><strong>Date:</strong> {new
Date(record.createdAt).toLocaleString()}</p>
            </li>
          ))}
        </ul>
      ) : (
        !loading && <p className="text-white">No search history found</p>
      )}
    </div>
  );
};

export default History;
```

LOGOUT BUTTON CODE:

```
import React, { useContext } from 'react';
import { AuthContext } from '../context/AuthContext';

const LogoutButton = () => {
  const { logout } = useContext(AuthContext);

  return (
    <button
      onClick={logout}
      className="bg-gradient-to-r from-red-500 to-red-600 text-white font-semibold py-2 px-6 rounded-
full shadow-lg hover:from-red-600 hover:to-red-700 transition duration-200"
    >
      Logout
    </button>
  );
};

export default LogoutButton;
```

APP.JS:

```
import { BrowserRouter as Router, Routes, Route, Navigate, useNavigate } from 'react-router-dom';
import { UserProvider, useUser } from './context/UserContext'; // Import useUser here
import Home from './components/Home';
import About from './components/About';
import SignIn from './components/SignIn';
import Login from './components/Login';
import Navbar from './components/Navbar';
import SearchDoctors from './components/SearchDoctors';
import History from './components/History';
```

```
const ProtectedRoute = ({ children }) => {
  const { user } = useUser(); // Now useUser is defined here
  return user ? children : <Navigate to="/login" replace />;
};

const AppWrapper = () => {
  const navigate = useNavigate(); // Get navigate from useNavigate here

  return (
    <UserProvider navigate={navigate}>
      <Navbar />
      <Routes>
        <Route path="/login" element={<Login />} />
        <Route path="/signin" element={<SignIn />} />
        <Route path="/home" element={<ProtectedRoute><Home /></ProtectedRoute>} />
        <Route path="/about" element={<ProtectedRoute><About /></ProtectedRoute>} />
        <Route path="/search" element={<ProtectedRoute><SearchDoctors /></ProtectedRoute>} />
        <Route path="/history" element={<ProtectedRoute><History /></ProtectedRoute>} />
        <Route path="*" element={<RedirectBasedOnUser />} />
      </Routes>
    </UserProvider>
  );
};

// Component for redirecting based on user state
const RedirectBasedOnUser = () => {
  const { user } = useUser();
  return <Navigate to={user ? "/home" : "/login"} replace />;
};

function App() {
  return (
    <Router>
      <AppWrapper />
    </Router>
  );
}

export default App;
```

USER CONTEXT.JS:

```
import { createContext, useState, useEffect, useContext, useCallback } from 'react';
import { useNavigate } from 'react-router-dom';
import axios from 'axios';

axios.defaults.baseURL = import.meta.env.VITE_API_URL;
axios.defaults.withCredentials = true; // Ensures cookies are included for server-side sessions

const UserContext = createContext();

export const UserProvider = ({ children }) => {
    const [user, setUser] = useState(() => {
        // Load user from localStorage if available
        const savedUser = localStorage.getItem('user');
        return savedUser ? JSON.parse(savedUser) : null;
```

```
});
const [loading, setLoading] = useState(false); // Loading state
const navigate = useNavigate();

// Register user
const registerUser = async (username, password) => {
   setLoading(true);
   try {
      const response = await axios.post('/user/register', { username, password });
      if (response.data.success) {
         const userData = response.data.data;
         setUser(userData);
         localStorage.setItem('user', JSON.stringify(userData)); // Store user data in localStorage
         navigate('/home');
      }
   } catch (error) {
      console.error("Registration error:", error);
      alert("Registration failed. Please try again.");
   } finally {
      setLoading(false);
   }
};

// Login user
const loginUser = async (username, password) => {
   setLoading(true);
   try {
      const response = await axios.post('/user/login', { username, password });
      if (response.data.success) {
         const userData = response.data.data;
         setUser(userData);
         localStorage.setItem('user', JSON.stringify(userData)); // Store user data in localStorage
         navigate('/home');
      }
   } catch (error) {
      console.error("Login error:", error);
      alert("Login failed. Please check your credentials and try again.");
   } finally {
      setLoading(false);
   }
};

// Logout user
const logoutUser = useCallback(async () => {
   try {
      await axios.get('/user/logout');
      setUser(null);
      localStorage.removeItem('user'); // Clear user data from localStorage
      navigate('/login');
   } catch (error) {
      console.error("Logout error:", error);
      alert("Logout failed. Please try again.");
   }
}, [navigate]);

useEffect(() => {
   const verifySession = async () => {
      setLoading(true);
      try {
        const response = await axios.get('/user/session');
        if (response.data.isLoggedIn) {
          const userData = response.data.data;
          setUser(userData);
          localStorage.setItem('user', JSON.stringify(userData)); // Sync user data with localStorage
          navigate('/home'); // Redirect to home if session is valid
        } else {
          setUser(null);
          localStorage.removeItem('user'); // Clear user from localStorage if session is invalid
          navigate('/login'); // Redirect to login if session is invalid
        }
      } catch (error) {
        console.error("Session verification error:", error);
        setUser(null);
        localStorage.removeItem('user'); // Clear user from localStorage on error
        navigate('/login'); // Redirect to login on error
      } finally {
        setLoading(false);
      }
    };

    // Run session verification on component mount
    verifySession();
  }, []); // Verify session once when the component mounts

  return (
    <UserContext.Provider value={{ user, registerUser, loginUser, logoutUser, loading }}>
      {children}
    </UserContext.Provider>
  );
};

export const useUser = () => {
  return useContext(UserContext);
};
```

SERVER:
USER.CONTROLLER.JS:

```
import { ApiResponse } from '../utils/ApiResponse.js';
import { ApiError } from '../utils/ApiError.js';
import User from '../models/user.model.js';
import bcrypt from 'bcrypt';
import Doctor from '../models/doctor.model.js';
import SearchHistory from '../models/history.model.js'; // Import SearchHistory model

// Middleware to check if the user is logged in
const isLoggedIn = (req, res, next) => {
  if (req.isAuthenticated()) {
    return next(); // If authenticated, proceed to the next middleware/controller
  }
  return next(new ApiError(401, 'You must be logged in to access this resource.'));
};

// Function to validate user data
```

```
const validateUserData = (username, password) => {
   const errors = [];

   if (!username || typeof username !== 'string' || username.trim() === '') {
      errors.push('Username is required.');
   }
   if (!password || typeof password !== 'string' || password.length < 6) {
      errors.push('Password must be at least 6 characters long.');
   }

   return errors;
};

// Register user and log them in
const registerUser = async (req, res, next) => {
   try {
      const { username, password } = req.body;

      // Validate user data
      const validationErrors = validateUserData(username, password);
      if (validationErrors.length > 0) {
         return next(new ApiError(400, 'Validation Error', validationErrors));
      }

      // Check if the user already exists
      const existingUser = await User.findOne({ username });
      if (existingUser) {
         return next(new ApiError(409, 'Username is already taken.'));
      }

      // Hash the password and create the user
      const hashedPassword = await bcrypt.hash(password, 10);
      const user = await User.create({ username, password: hashedPassword });

      // Log in the user immediately after registration
      req.login(user, (err) => {
         if (err) {
            return next(new ApiError(500, 'Error during login after registration.', [err.message]));
         }
         res.status(201).json(new ApiResponse(201, user, 'User registered and logged in successfully'));
      });
   } catch (error) {
      return next(new ApiError(500, 'An error occurred during registration.', [error.message]));
   }
};

// Login user
const loginUser = (req, res) => {
   res.status(200).json(new ApiResponse(200, req.user, 'Logged in successfully', { message: 'Welcome
back!' }));
};

// Logout user
const logoutUser = (req, res, next) => {
   req.logout((err) => {
      if (err) {
         return next(new ApiError(500, 'An error occurred while logging out.'));
```

```
        }
        res.status(200).json(new ApiResponse(200, null, 'Logged out successfully', { message: 'See you next
time!' }));
    });
};

// Verify session
const verifySession = (req, res) => {
    if (req.isAuthenticated()) {
        res.status(200).json(new ApiResponse(200, req.user, 'Session is valid', { isLoggedIn: true }));
    } else {
        res.status(401).json(new ApiResponse(401, null, 'Session is not valid', { isLoggedIn: false }));
    }
};

// Search for doctors by category and save in history
const searchDoctors = async (req, res, next) => {
    const { name, age, gender, blood_group, category } = req.body; // Assuming user's details are stored in
req.user

    // Build the search query for doctors
    let query = {};
    if (category) query.specialization = category;

    try {
        // Find doctors that match the category
        const doctors = await Doctor.find(query);

        // Save search criteria in history
        if (req.isAuthenticated()) {
            const userId = req.user._id;
            const searchHistory = new SearchHistory({
                userId,
                searchCriteria: {
                    name,
                    age,
                    gender,
                    blood_group,
                    category
                }
            });
            await searchHistory.save();
        }

        // Return matched doctors
        if (doctors.length > 0) {
            res.status(200).json(new ApiResponse(200, doctors, 'Doctors found'));
        } else {
            res.status(404).json(new ApiResponse(404, null, 'No doctors found matching the criteria'));
        }
    } catch (error) {
        return next(new ApiError(500, 'Error searching for doctors', [error.message]));
    }
};

// Fetch search history for the logged-in user
const getSearchHistory = async (req, res, next) => {
  try {
    const userId = req.user._id;
    const history = await SearchHistory.find({ userId }).sort({ createdAt: -1 });

    res.status(200).json(new ApiResponse(200, history, 'Search history retrieved successfully'));
  } catch (error) {
    return next(new ApiError(500, 'Error fetching search history', [error.message]));
  }
};

// Function to check doctor's availability
const checkDoctorAvailability = async (req, res, next) => {
  const { usertime, doctorid } = req.query;

  try {
    const doctor = await Doctor.findById(doctorid);
    if (!doctor) {
      return next(new ApiError(404, 'Doctor not found'));
    }

    const [startTime, endTime] = usertime.split('-').map(time => parseInt(time));

    // Assuming availability_time is in the format "HH:MM AM/PM - HH:MM AM/PM"
    const [availabilityStart, availabilityEnd] = doctor.availability_time.split(' - ');
    const [availabilityStartHour, availabilityStartMinute] = parseTime(availabilityStart);
    const [availabilityEndHour, availabilityEndMinute] = parseTime(availabilityEnd);

    // Check if the usertime falls within the doctor's availability
    const isAvailable =
      (startTime >= availabilityStartHour && endTime <= availabilityEndHour) &&
      (startTime < endTime); // Ensuring the provided time range is valid

    if (isAvailable) {
      res.status(200).json(new ApiResponse(200, null, 'Doctor is available'));
    } else {
      res.status(200).json(new ApiResponse(200, null, 'Doctor is not available'));
    }
  } catch (error) {
    return next(new ApiError(500, 'Error checking doctor availability', [error.message]));
  }
};

// Helper function to parse time in "HH:MM AM/PM" format
const parseTime = (time) => {
  const [hours, minutesPart] = time.split(':');
  const [minutes, period] = minutesPart.split(' ');
  let hour = parseInt(hours);
  const minute = parseInt(minutes);

  if (period === 'PM' && hour !== 12) {
    hour += 12;
  } else if (period === 'AM' && hour === 12) {
    hour = 0;
  }

  return [hour, minute];
};
```

```
export { registerUser, loginUser, logoutUser, verifySession, isLoggedIn, searchDoctors, getSearchHistory,
checkDoctorAvailability };
```

MULTER.JS:

```
import multer from "multer";
import path from "path";
import fs from "fs";

// Ensure the temp directory exists
const tempDir = "./public/temp";
if (!fs.existsSync(tempDir)) {
  fs.mkdirSync(tempDir, { recursive: true });
}

const storage = multer.diskStorage({
  destination: function (req, file, cb) {
    cb(null, tempDir); // Use pre-defined temp directory
  },
  filename: function (req, file, cb) {
    // Add a timestamp to prevent overwriting files with the same name
    const uniqueSuffix = Date.now() + '-' + Math.round(Math.random() * 1E9);
    cb(null, `${uniqueSuffix}-${file.originalname}`);
  }
});

// Optional: Add a file filter to limit file types (e.g., images only)
const fileFilter = (req, file, cb) => {
  const allowedTypes = /jpeg|jpg|png|pdf|docx/; // Specify allowed file types
  const mimeType = allowedTypes.test(file.mimetype);
  const extname = allowedTypes.test(path.extname(file.originalname).toLowerCase());

  if (mimeType && extname) {
    return cb(null, true);
  }
  cb(new Error("Unsupported file type."), false);
};

// Configure Multer with additional options
const upload = multer({
  storage,
  limits: { fileSize: 1024 * 1024 * 5 }, // Set file size limit to 5MB
  fileFilter
});

export { upload };
```

ISAUTHENTICATED.JS:

```
// middlewares/auth.middleware.js
const isAuthenticated = (req, res, next) => {
  if (req.isAuthenticated()) {
    return next(); // User is authenticated, proceed to the next middleware/route handler
    }
    return res.status(401).json({
        success: false,
        message: 'Unauthorized access. Please log in.',
    });
};

export default isAuthenticated;
```

MODELS:
DOCTOR.MODEL.JS:

```
// doctor.model.js
import mongoose from 'mongoose';

const doctorSchema = new mongoose.Schema({
    name: String,
    age: Number,
    gender: String,
    specialization: String,
    availability_time: String,
    contact_details: {
        phone: String,
        email: String
    },
    achievements: [String]
});

const Doctor = mongoose.model('Doctor', doctorSchema);

export default Doctor;
```

FILE.MODEL.JS:

```
import mongoose from 'mongoose';

const fileSchema = new mongoose.Schema({
    user: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
    originalName: { type: String, required: true },
    filePath: { type: String, required: true },
    createdAt: { type: Date, default: Date.now }
});

const File = mongoose.model('File', fileSchema);

export default File;
```

HISTORY.MODEL.JS:

```
// models/searchHistory.model.js
import mongoose from 'mongoose';

const searchHistorySchema = new mongoose.Schema({
    userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
    searchCriteria: {
        name: String,
        age: Number,
        gender: String,
        blood_group: String,
        category: String
    },
    createdAt: { type: Date, default: Date.now }
});

const SearchHistory = mongoose.model('SearchHistory', searchHistorySchema);

export default SearchHistory;
```

USER.MODEL.JS:

```
import mongoose from 'mongoose';

const userSchema = new mongoose.Schema({
    username: { type: String, required: true, unique: true },
    password: { type: String, required: true }
});

const User = mongoose.model('User', userSchema);

export default User;
```

USERS:
USER.ROUTES.JS:

```
import express from 'express';
import passport from 'passport';
import { registerUser, loginUser, logoutUser, verifySession, isLoggedIn, searchDoctors, getSearchHistory,
checkDoctorAvailability } from '../controllers/user.controller.js';
import asyncHandler from '../utils/asyncHandler.js';

const router = express.Router();

// Register user route
router.post('/register', asyncHandler(registerUser));

// Login user route
router.post('/login', passport.authenticate('local', { session: true }), asyncHandler(loginUser));

// Logout user route (only accessible if logged in)
router.get('/logout', isLoggedIn, asyncHandler(logoutUser));

// Verify session route
router.get('/session', asyncHandler(verifySession));

// Search doctors route
router.post('/search', isLoggedIn, asyncHandler(searchDoctors));

// Get search history route
router.get('/history', isLoggedIn, asyncHandler(getSearchHistory));

// Check doctor availability route
router.get('/checkavailability', isLoggedIn, asyncHandler(checkDoctorAvailability));

export default router;
```

APP.JS:

```
import express from 'express';
import session from 'express-session';
import passport from 'passport';
import LocalStrategy from 'passport-local';
import bcrypt from 'bcrypt';
import cors from 'cors';
import cookieParser from 'cookie-parser';
import User from './models/user.model.js';
import dotenv from 'dotenv';
import mongoose from 'mongoose';
import Doctor from './models/doctor.model.js';

dotenv.config();

const app = express();

// Middleware
app.use(express.json());
app.use(express.urlencoded({ extended: true }));
app.use(cookieParser());
app.use(cors({
  origin: 'http://localhost:5173', // your frontend URL
  credentials: true, // Allow credentials
}));

app.use(session({
  secret: process.env.SESSION_SECRET || 'secret',
  resave: false,
  saveUninitialized: false,
  cookie: {
    httpOnly: true,
    secure: false, // Change to true for production with HTTPS
    maxAge: 1000 * 60 * 60 * 24,
  },
}));

// Passport.js initialization
app.use(passport.initialize());
app.use(passport.session());

// Passport Local strategy
passport.use(new LocalStrategy.Strategy(
  async (username, password, done) => {
    try {
      const user = await User.findOne({ username });
      if (!user) {
        return done(null, false, { message: 'Incorrect username' });
      }

      const isMatch = await bcrypt.compare(password, user.password);
      if (!isMatch) {
        return done(null, false, { message: 'Incorrect password' });
      }
```

```
            return done(null, user);
        } catch (error) {
            return done(error);
        }
    }
));

// Passport serialization and deserialization
passport.serializeUser((user, done) => done(null, user.id));
passport.deserializeUser(async (id, done) => {
    try {
        const user = await User.findById(id);
        done(null, user);
    } catch (error) {
        done(error, null);
    }
});

// Import and use user routes
app.get("/", (_, res) => {
    res.send("server connected");
});

import userRoutes from './routes/user.route.js';
app.use('/api/v1/user', userRoutes);

// Add route to insert doctors' data
app.get('/api/v1/doctors', async (req, res) => {
    try {
        const doctors = await Doctor.find();
        res.status(200).json(doctors);
    } catch (error) {
        res.status(500).json({ message: 'Error fetching doctors data', error });
    }
});


export default app;
```

INDEX.JS:

```
import mongoose from 'mongoose';
import app from './app.js';

const PORT = process.env.PORT || 3000;

const connectDb = async () => {
    try {
        const mongoUri = process.env.MONGO_URI;

        // Ensure the MONGO_URI variable is properly defined
        if (!mongoUri) {
            throw new Error("MONGO_URI is not defined in the environment variables.");
        }

        // Connect to MongoDB
        await mongoose.connect(`${mongoUri}`);
        console.log('Connected to MongoDB Atlas');

        // Start the server
        app.listen(PORT, () => {
            console.log(`Server is running on port ${PORT}`);
        });
    } catch (error) {
        console.error('Error connecting to MongoDB Atlas:', error);
    }
}

connectDb();
```