Landing Page:

```
import React from "react";
import "../styling/landingPage.css";
import TestimonialsCarousel from "./carousel";
import Login from "./login";
import { useNavigate, useLocation } from "react-router-dom";
import { useState } from "react";
import appointment from "../assets/appointment.png";
import orderMed from "../assets/order medicines.png";
import docVisit from "../assets/docVisit.png";

const MedicalLandingPage = () => {
  const location = useLocation();
  const { userData } = location.state || {};
  const currentHour = new Date().getHours();
  let greeting;

  if (currentHour < 12) {
    greeting = "Good Morning";
  } else if (currentHour < 18) {
    greeting = "Good Afternoon";
  } else {
    greeting = "Good Evening";
  }

  return (
    <div style={{ display: "flex", flexDirection: "column" }}>
      <Header userData={userData} />
      <MainContent userData={userData} greeting={greeting} />
      <Footer />
    </div>
  );
};

const Header = ({ userData }) => {
  const [activeLink, setActiveLink] = useState("services");
  const navigate = useNavigate();

  const handleLinkClick = (link) => {
```

```
  setActiveLink(link);
};

const handleNavigation = () => {
  navigate("/login");
};

return (
  <header
    className="header"
    style={{
      display: "flex",
      justifyContent: "space-between",
      alignItems: "center",
    }}
  >
    <div>
      <h1>eMed</h1>
    </div>
    <div>
      <nav>
        <ul className="nav-links">
          <li>
            <a
              // href="#services"
              className={activeLink === "services" ? "active" : ""}
              onClick={() => handleLinkClick("services")}
            >
              Services
            </a>
          </li>
          <li>
            <a
              // href="#testimonials"
              className={activeLink === "testimonials" ? "active" : ""}
              onClick={() => handleLinkClick("testimonials")}
            >
              Testimonials
            </a>
          </li>
```

```
            <li>
              <a
                // href="#appointment"
                className={activeLink === "appointment" ? "active" : ""}
                onClick={() => handleLinkClick("appointment")}
              >
                Make Appointment
              </a>
            </li>
            <li>
              <a
                // href="#buy-medicine"
                className={activeLink === "buy-medicine" ? "active" : ""}
                onClick={() => handleLinkClick("buy-medicine")}
              >
                Buy Medicine
              </a>
            </li>
          </ul>
        </nav>
      </div>
      <div>
        <button onClick={handleNavigation} className="loginBut">
          {userData?.name ? "Logout" : "Login/Register"}
        </button>
      </div>
    </header>
  );
};

const MainContent = ({ userData, greeting }) => {
  const navigate = useNavigate();

  const handleBuyNowButton = () => {
    navigate("/medicines", { state: { userData } });
  };

  const handleBookNow = () => {
    navigate("/docAppointment", { state: { userData } });
  };
  return (
    <main>
      <section className="hero" style={{ display: "flex", gap: "20px" }}>
        <div
          style={{
            display: "flex",
            flexDirection: "column",
            justifyContent: "space-around",
            gap: "50px",
            marginLeft: "100px",
          }}
        >
          <div>
            <h2>
              {greeting}, {userData?.name}
            </h2>
          </div>
          <div style={{ flex: 1 }}>
            <h2>Your Health, Our Priority</h2>
            <p>Connecting you to the best healthcare services online.</p>
          </div>
        </div>
        <div style={{ flex: 1 }}>
          <img src={docVisit} alt="Doctor Visit" />
        </div>
      </section>
      <section id="services" className="services">
        <h1
          style={{ display: "flex", justifyContent: "center", fontSize:
"2em" }}
        >
          Services
        </h1>
        <div className="service-list">
          <div className="service-item">
            <h3>Online Consultations</h3>
            <p>Consult with top doctors from the comfort of your home.</p>
          </div>
          <div className="service-item">
```

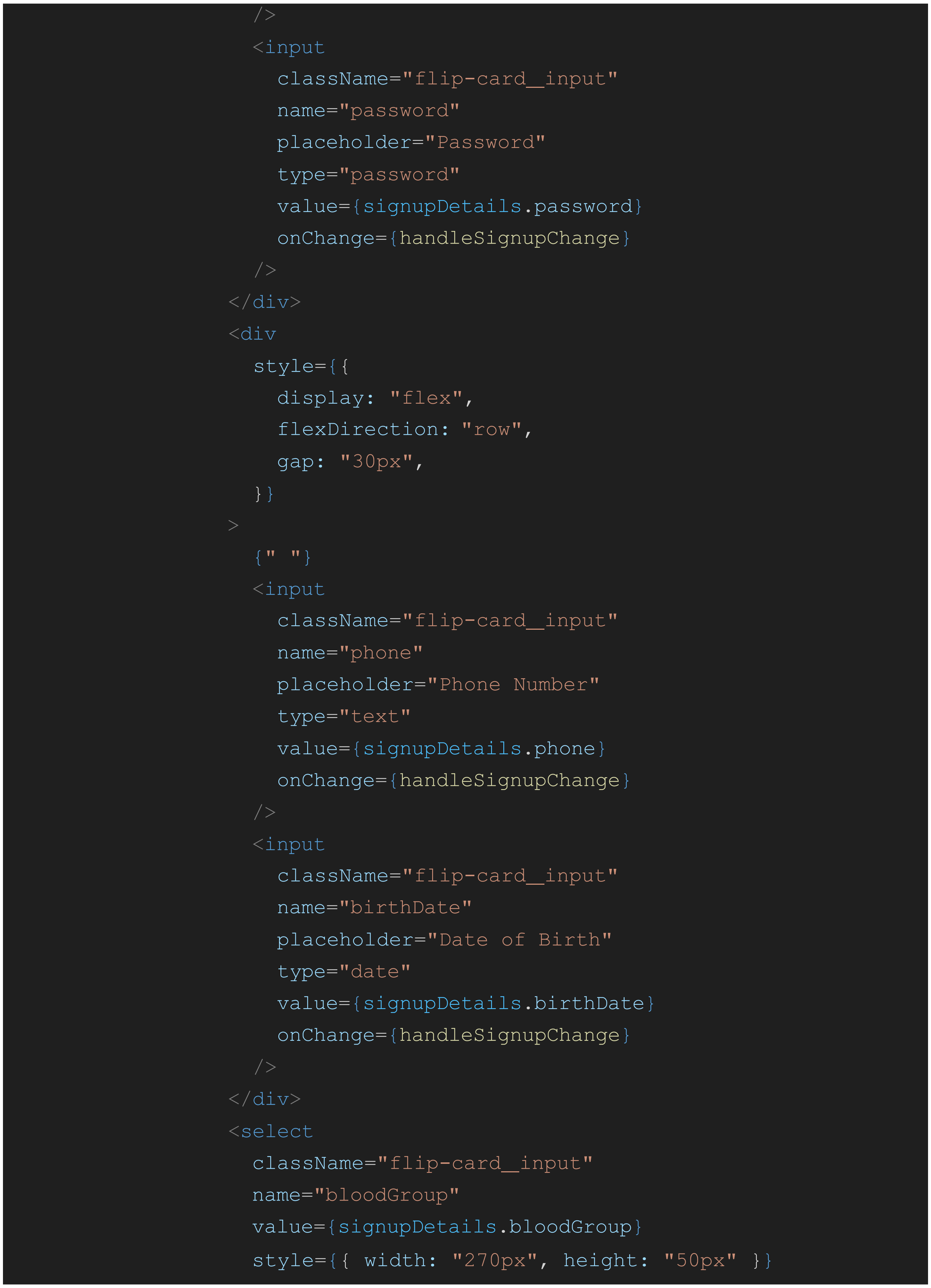
```
            <h3>Prescription Medicines</h3>
            <p>Order your prescribed medicines with easy home
delivery.</p>
          </div>
          <div className="service-item">
            <h3>Health Tracking</h3>
            <p>Track your health and medication adherence
effortlessly.</p>
          </div>
        </div>
      </section>
      <section id="testimonials">
        <TestimonialsCarousel userData={userData} />
      </section>

      <section id="appointment" className="appointment">
        <h1
          style={{ display: "flex", justifyContent: "center", fontSize:
"2em" }}
        >
          Make an Appointment
        </h1>
        <div
          style={{
            height: "400px",
            display: "flex",
            flexDirection: "row",
            justifyContent: "space-around",
            alignItems: "center",
          }}
        >
          <div>
            <button className="appointment-button"
onClick={handleBookNow}>
              Book Now
            </button>
          </div>
          <div>
            <img src={appointment} alt="Appointment" />
          </div>
        </div>
      </section>

      <section id="buy-medicine" className="appointment">
        <h1
          style={{ display: "flex", justifyContent: "center", fontSize:
"2em" }}
        >
          Order Medicines
        </h1>
        <div
          style={{
            height: "400px",
            display: "flex",
            flexDirection: "row",
            justifyContent: "space-around",
            alignItems: "center",
          }}
        >
          <div>
            <img
              style={{ height: "300px" }}
              src={orderMed}
              alt="Order Medicines"
            />
          </div>
          <div>
            <button className="appointment-button"
onClick={handleBuyNowButton}>
              Order Medicines
            </button>
          </div>
        </div>
      </section>
    </main>
  );
};

const Footer = () => {
  return (
    <footer className="footer">
      <p>Contact Us: eMed@gmail.com | Phone: xxx-xxx-xxxx</p>
    </footer>
  );
};

export default MedicalLandingPage;
```

Landing page for users contains the services provided by us, testimonials by users, booking appointments, option for ordering medicines. It also has some contact information.

Login/Signup:

```
import React, { useState } from "react";
import "../styling/login.css";
import axios from "axios";
import { useNavigate } from "react-router-dom";
import MessageScreen from "./messageScreen";

const Login = () => {
  const navigate = useNavigate();

  const [loginEmail, setLoginEmail] = useState("");
  const [loginPassword, setLoginPassword] = useState("");

  // Signup Form State
  const [signupDetails, setSignupDetails] = useState({
    name: "",
    email: "",
    password: "",
    phone: "",
    birthDate: "",
    bloodGroup: "",
    street: "",
    city: "",
    state: "",
    zipCode: "",
    emergencyName: "",
    emergencyPhone: "",
  });

  const [isLoading, setIsLoading] = useState(false);
  const [error, setError] = useState(null);
  const [user, setLoggedUser] = useState(null);
  const [successMessage, setSuccessMessage] = useState("");

  const handleSignupChange = (e) => {
    const { name, value } = e.target;
    setSignupDetails((prevDetails) => ({ ...prevDetails, [name]: value
}));
  };

  const displayMessageAndNavigate = (message, userData) => {
    setSuccessMessage(message);
    setTimeout(() => {
      setSuccessMessage("");
      navigate("/landing", { state: { userData } });
    }, 3000);
  };

  const handleLogin = async (e) => {
    e.preventDefault();
    setIsLoading(true);
    setError(null);

    try {
      const data = await loginApi({
        email: loginEmail,
        password: loginPassword,
      });
      console.log("data", data);

      setLoggedUser(data.user);
      const userData = data.user.user;
      displayMessageAndNavigate("Login Successful!", data.user.user);

      // Set user in state

      // Navigate to landing page
    } catch (err) {
      setError(err.message);
    } finally {
      setIsLoading(false);
    }
  };

  // Handle signup form submission
  const handleSignup = async (e) => {
    e.preventDefault();
    setIsLoading(true);
    setError(null);

    try {
      const data = await signupApi(signupDetails);
      setLoggedUser({ name: signupDetails.name });
      console.log(data, "signupData");
      const userData = signupDetails;

      if (data) {
        navigate("/landing", { state: { userData } });
      }
    } catch (err) {
      setError(err.message);
    } finally {
      setIsLoading(false);
    }
  };

  const loginApi = async (loginData) => {
    try {
      const response = await axios.post(
        "http://localhost:5000/api/login",
        loginData
      );
      console.log(response);

      return response.data;
    } catch (error) {
      throw new Error(error.response?.data?.error || "Login failed.");
    }
  };

  const signupApi = async (signupData) => {
    console.log(signupData);
    try {
      const response = await axios.post(
        "http://localhost:5000/api/signup",
        signupData
      );
      return response.data;
    } catch (error) {
      throw new Error(error.response?.data?.error || "Signup failed.");
    }
  };

  if (successMessage) {
    return <MessageScreen message={successMessage} />;
  }
  return (
    <>
      <div className="wrapper ">
        <div className="card-switch">
          <label className="switch">
            <input type="checkbox" className="toggle" />
            <span className="slider"></span>
            <span className="card-side"></span>
            <div className="flip-card__inner">
              <div className="flip-card__front">
                <div className="title">Log in</div>
                <form className="flip-card__form" onSubmit={handleLogin}>
                  <input
                    className="flip-card__input"
                    name="email"
                    placeholder="Email"
                    type="email"
                    value={loginEmail}
                    onChange={(e) => setLoginEmail(e.target.value)}
                  />
                  <input
```

```
                    className="flip-card__input"
                    name="password"
                    placeholder="Password"
                    type="password"
                    value={loginPassword}
                    onChange={(e) => setLoginPassword(e.target.value)}
                  />
                  <button className="flip-card__btn"
onClick={handleLogin}>
                    Let's go!
                  </button>
                </form>
              </div>

              <div className="flip-card__back">
                <div className="title">Sign up</div>
                <form className="flip-card__form" onSubmit={handleSignup}>
                  <input
                    className="flip-card__input"
                    name="name"
                    placeholder="Name"
                    type="text"
                    value={signupDetails.name}
                    onChange={handleSignupChange}
                  />
                  <div
                    style={{
                      display: "flex",
                      flexDirection: "row",
                      gap: "30px",
                    }}
                  >
                    {" "}
                    <input
                      className="flip-card__input"
                      name="email"
                      placeholder="Email"
                      type="email"
                      value={signupDetails.email}
                      onChange={handleSignupChange}
```



```
  />
  <input
    className="flip-card__input"
    name="password"
    placeholder="Password"
    type="password"
    value={signupDetails.password}
    onChange={handleSignupChange}
  />
</div>
<div
  style={{
    display: "flex",
    flexDirection: "row",
    gap: "30px",
  }}
>
  {" "}
  <input
    className="flip-card__input"
    name="phone"
    placeholder="Phone Number"
    type="text"
    value={signupDetails.phone}
    onChange={handleSignupChange}
  />
  <input
    className="flip-card__input"
    name="birthDate"
    placeholder="Date of Birth"
    type="date"
    value={signupDetails.birthDate}
    onChange={handleSignupChange}
  />
</div>
<select
  className="flip-card__input"
  name="bloodGroup"
  value={signupDetails.bloodGroup}
  style={{ width: "270px", height: "50px" }}
  onChange={handleSignupChange}
>
  <option value="" disabled>
    Select Blood Group
  </option>
  <option>A+</option>
  <option>A-</option>
  <option>B+</option>
  <option>B-</option>
  <option>AB+</option>
  <option>AB-</option>
  <option>O+</option>
  <option>O-</option>
</select>
<div
  style={{
    display: "flex",
    flexDirection: "row",
    gap: "30px",
  }}
>
  {" "}
  <input
    className="flip-card__input"
    name="street"
    placeholder="Street"
    type="text"
    value={signupDetails.street}
    onChange={handleSignupChange}
  />
  <input
    className="flip-card__input"
    name="city"
    placeholder="City"
    type="text"
    value={signupDetails.city}
    onChange={handleSignupChange}
  />
</div>
<div
```

```
    style={{
      display: "flex",
      flexDirection: "row",
      gap: "30px",
    }}
  >
    <input
      className="flip-card__input"
      name="state"
      placeholder="State"
      type="text"
      value={signupDetails.state}
      onChange={handleSignupChange}
    />
    <input
      className="flip-card__input"
      name="zipCode"
      placeholder="Zip Code"
      type="text"
      value={signupDetails.zipCode}
      onChange={handleSignupChange}
    />
  </div>
  <div
    style={{
      display: "flex",
      flexDirection: "row",
      gap: "30px",
    }}
  >
    <input
      className="flip-card__input"
      name="emergencyName"
      placeholder="Emergency Contact Name"
      type="text"
      value={signupDetails.emergencyName}
      onChange={handleSignupChange}
    />
    <input
      className="flip-card__input"
```

```
                          name="emergencyPhone"
                          placeholder="Emergency Contact Phone"
                          type="text"
                          value={signupDetails.emergencyPhone}
                          onChange={handleSignupChange}
                        />
                      </div>
                      <button className="flip-card_btn"
onClick={handleSignup}>
                        Confirm!
                      </button>
                    </form>
                  </div>
                </div>
              </label>
            </div>

            {/* {isLoading && <p>Loading...</p>}
            {error && <p>Error: {error}</p>} */}
          </div>
        </>
  );
};

export default Login;
```

Login page has the signup and login features where user provide their details like email, address phone number, blood group and other emergency details for registration and it also has a page for login which redirects to landing page with the user details on successful login or signup.

Doctor Appointment:

```
import React, { useState, useEffect } from "react";
import { useLocation, useNavigate } from "react-router-dom";
import "../styling/appointment.css";
import {
  TextField,
  Button,
  MenuItem,
  Typography,
  Container,
  Grid,
  InputLabel,
  Select,
  FormControl,
} from "@mui/material";

const DoctorAppointment = () => {
  const location = useLocation();
  const { userData } = location.state || {};
  console.log(userData);
  const [formData, setFormData] = useState({
    name: userData.name,
    email: userData.email,
    date: "",
    time: "",
    doctor: "",
  });
  const [isSubmitted, setIsSubmitted] = useState(false);
  const navigate = useNavigate();

  const doctorsList = [
    "Dr. John Doe - Cardiologist",
    "Dr. Sarah Smith - Dermatologist",
    "Dr. Emma Taylor - Neurologist",
  ];

  const handleChange = (e) => {
    const { name, value } = e.target;
    setFormData((prevData) => ({ ...prevData, [name]: value }));
  };

  const handleSubmit = (e) => {
    e.preventDefault();
    console.log("Appointment booked:", formData);
    setIsSubmitted(true);
  };
  useEffect(() => {
    if (isSubmitted) {
      const timer = setTimeout(() => {
        navigate("/landing", { state: { userData } });
      }, 5000);

      return () => clearTimeout(timer);
    }
  }, [isSubmitted, navigate]);

  return (
    <Container className="appointment-container">
      {/* <Typography className="title">Book a Doctor
Appointment</Typography> */}
      {isSubmitted ? (
        <Typography className="success-message">
          Appointment successfully booked! We'll see you soon.
        </Typography>
      ) : (
        <div>
          <h1> Book your Appointment</h1>
          <form onSubmit={handleSubmit} noValidate>
            <Grid container spacing={2}>
              <Grid item xs={12}>
                <TextField
                  fullWidth
                  required
                  label="Your Name"
                  name="name"
                  value={formData.name}
                  onChange={handleChange}
                />
              </Grid>
              <Grid item xs={12}>
                <TextField
                  fullWidth
                  required
                  label="Email Address"
                  name="email"
                  type="email"
                  value={formData.email}
                  onChange={handleChange}
                />
              </Grid>
              <Grid item xs={6}>
                <TextField
                  fullWidth
                  required
                  name="date"
                  label="Appointment Date"
                  type="date"
                  InputLabelProps={{ shrink: true }}
                  value={formData.date}
                  onChange={handleChange}
                />
              </Grid>
              <Grid item xs={6}>
                <TextField
                  fullWidth
                  required
                  name="time"
                  label="Appointment Time"
                  type="time"
                  InputLabelProps={{ shrink: true }}
                  value={formData.time}
                  onChange={handleChange}
                />
              </Grid>
              <Grid item xs={12}>
                <FormControl fullWidth required>
                  <InputLabel>Select Doctor</InputLabel>
                  <Select
                    name="doctor"
                    value={formData.doctor}
                    onChange={handleChange}
                  >
                    {doctorsList.map((doctor, index) => (
                      <MenuItem key={index} value={doctor}>
                        {doctor}
                      </MenuItem>
                    ))}
                  </Select>
                </FormControl>
              </Grid>
            </Grid>
            <Button
              type="submit"
              fullWidth
              variant="contained"
              className="book-button"
            >
              Book Appointment
            </Button>
          </form>
        </div>
      )}
    </Container>
  );
};

export default DoctorAppointment;
```

After clicking on the book appointment button in the landing page it redirects to the appointment booking page with the user details present

Here users can select the doctors details along with data and time for the appointment and can confirm booking.

Medicines:

```
import React, { useEffect, useState } from "react";
import { useNavigate, useLocation } from "react-router-dom";
import { Pagination } from "@mui/material";
import "../styling/medicines.css";
import output100 from "../assets/output100.json";

const MedicinePage = () => {
  const [medicines, setMedicines] = useState([]);
  const [filteredMedicines, setFilteredMedicines] = useState([]);
  const [cart, setCart] = useState(new Map());
  const [currentPage, setCurrentPage] = useState(1);
  const [searchTerm, setSearchTerm] = useState("");
  const itemsPerPage = 10;
  const navigate = useNavigate();
  const location = useLocation();
  const { cartSaved } = location.state || {};
  const { userData } = location.state || {};

  console.log("user data in medicines", userData);
  useEffect(() => {
    setMedicines(output100);
    setFilteredMedicines(output100);

    const savedCart = localStorage.getItem("cart");
    if (savedCart) setCart(new Map(JSON.parse(savedCart)));

    if (cartSaved) {
      const savedCartMap = new Map(cartSaved.map((item) => [item.id,
item]));
      setCart(savedCartMap);
    }
  }, [cartSaved]);

  useEffect(() => {
    localStorage.setItem("cart",
JSON.stringify(Array.from(cart.entries())));
  }, [cart]);

  const handleAddToCart = (medicine) => {
    setCart((prevCart) => {
      const newCart = new Map(prevCart);
      const currentQty = newCart.get(medicine.id)?.quantity || 0;
      newCart.set(medicine.id, { ...medicine, quantity: currentQty + 1 });
      return newCart;
    });
  };

  const handleRemoveFromCart = (medicine) => {
    setCart((prevCart) => {
      const newCart = new Map(prevCart);
      const currentQty = newCart.get(medicine.id)?.quantity || 0;

      if (currentQty > 1) {
        newCart.set(medicine.id, { ...medicine, quantity: currentQty - 1
});
      } else {
        newCart.delete(medicine.id);
      }
      return newCart;
    });
  };

  const handlePageChange = (event, value) => setCurrentPage(value);

  const handleCheckout = () => {
    const cartArray = Array.from(cart.values());
    navigate("/cart", { state: { cart: cartArray, userData } });
  };

  const handleSearch = (e) => {
    const value = e.target.value.toLowerCase();
    setSearchTerm(value);
    const filtered = medicines.filter((medicine) =>
      medicine.name.toLowerCase().includes(value)
    );
    setFilteredMedicines(filtered);
    setCurrentPage(1);
  };

  const indexOfLastItem = currentPage * itemsPerPage;
  const indexOfFirstItem = indexOfLastItem - itemsPerPage;
  const currentMedicines = filteredMedicines.slice(
    indexOfFirstItem,
    indexOfLastItem
  );

  return (
    <div className="medicine-container">
      <h1>Available Medicines</h1>
      <input
        type="text"
        placeholder="Search for a medicine..."
        value={searchTerm}
        onChange={handleSearch}
        // style={{ marginBottom: "20px", padding: "10px", width: "100%"
}}
        className="search"
      />
      <div className="medicine-list">
        {currentMedicines.map((medicine) => {
          const cartItem = cart.get(medicine.id) || {};
          const quantity = cartItem.quantity || 0;

          return (
            <div key={medicine.id} className="medicine-item">
              <h2>{medicine.name}</h2>
              <p>Price: Rs {medicine.price}</p>
              {quantity > 0 ? (
                <div className="quantity-controls">
                  <button onClick={() => handleRemoveFromCart(medicine)}>
                    -
                  </button>
                  <span>{quantity}</span>
                  <button onClick={() =>
handleAddToCart(medicine)}>+</button>
                </div>
              ) : (
                <button onClick={() => handleAddToCart(medicine)}>
                  Add to Cart
                </button>
              )}
            </div>
          );
        })}
      </div>

      <Pagination
        count={Math.ceil(filteredMedicines.length / itemsPerPage)}
        page={currentPage}
        onChange={handlePageChange}
        color="primary"
        style={{ marginTop: "20px", display: "flex", justifyContent:
"center" }}
        sx={{
          "& .MuiPaginationItem-root": {
            color: "black",
            backgroundColor: "#f0f0f0",
            margin: "0 5px",
          },
          "& .Mui-selected": {
            backgroundColor: "#1976d2",
            color: "white",
          },
          "& .MuiPaginationItem-ellipsis": {
            color: "#888",
          },
        }}
      />

      <button
        onClick={handleCheckout}
        style={{ marginTop: "20px", padding: "10px 20px", cursor:
"pointer" }}
      >
        Go to Cart (
        {Array.from(cart.values()).reduce((a, b) => a + b.quantity, 0)})
      </button>
    </div>
  );
};

export default MedicinePage;
```

This page can be access by clicking on order now button on landing page

This page has a list of medicines. From here users can select the required medicines and can add them to the cart.

Cart:

```
import React, { useState, useEffect } from "react";
import { useLocation, useNavigate } from "react-router-dom";
import "../styling/cart.css";

const CartPage = () => {
  const { state } = useLocation();
  const navigate = useNavigate();
  const [cart, setCart] = useState(state?.cart || []);
  const [deliveryMessage, setDeliveryMessage] = useState("");
  const userData = state?.userData || {};

  const handleBackToMedicine = () => {
    const cartSaved = cart;
    navigate("/medicines", { state: { cartSaved, userData } });
  };

  const handleCheckout = () => {
    const address = userData.address || "your address";
    setDeliveryMessage(
      `Your medicines will be delivered to ${address.street},
${address.city}, ${address.state} shortly!`
    );
    setCart([]);
  };

  useEffect(() => {
    if (deliveryMessage) {
      const timer = setTimeout(() => {
        navigate("/landing", { state: { userData } });
      }, 3000);
      return () => clearTimeout(timer);
    }
  }, [deliveryMessage, navigate, userData]);

  const totalPrice = cart
    .reduce((acc, item) => acc + item.price * item.quantity, 0)
    .toFixed(2);
  return (
    <div className="cart-container">
      <h1>Your Cart</h1>
      {cart.length === 0 ? (
        <p>Your cart is empty.</p>
      ) : (
        <>
          <ul>
            {cart.map((item) => (
              <li className="card-li" key={item.id}>
                <h3>{item.name}</h3>
                <p>Quantity: {item.quantity}</p>
                <p> Rs {item.price * item.quantity}</p>
              </li>
            ))}
          </ul>
          <h3>Total: Rs {totalPrice}</h3>
          <button onClick={handleCheckout}>Proceed to Checkout</button>
        </>
      )}
      {deliveryMessage && <p
className="delivery-message">{deliveryMessage}</p>}
      <button onClick={handleBackToMedicine}>Back to Medicines</button>
    </div>
  );
};

export default CartPage;
```

This is the cart page where users can see all the medicines that they have added to cart by selecting in the medicines page, here users can see the price of individual medicine as well as the total cart price and also have an option for going back to medicines list to add more medicines to cart or to remove few medicines as required.

From here the users can go to the payment gateway which will be implemented in the later phases.

BackendCode:

Server:

```
const express = require("express");
const mongoose = require("mongoose");
const bodyParser = require("body-parser");
const cors = require("cors");
const bcrypt = require("bcrypt");

const app = express();
const PORT = process.env.PORT || 5000;

// Middleware
app.use(bodyParser.json());
app.use(cors());

// MongoDB Connection
mongoose
  .connect(

"mongodb+srv://gnevercodes:seproject@softwareengineeringdb.xli4x.mongodb.n
et/mydatabase?retryWrites=true&w=majority",
    { useNewUrlParser: true, useUnifiedTopology: true }
  )
  .then(() => console.log("Connected to MongoDB successfully!"))
  .catch((error) => console.error("Failed to connect to MongoDB:",
error));

// Event listeners for MongoDB connection
mongoose.connection.on("connected", () =>
  console.log("Mongoose is connected.")
);
mongoose.connection.on("error", (err) => console.error("Mongoose error:",
err));
mongoose.connection.on("disconnected", () =>
  console.log("Mongoose disconnected.")
);

// User Schema and Model
```

```
const userSchema = new mongoose.Schema({
  name: { type: String, required: true },
  email: { type: String, required: true, unique: true, match: /.+\@.+\..+/
},
  password: { type: String, required: true },
  phoneNumber: { type: String, required: true, match: /^\d{10}$/ },
  dateOfBirth: { type: Date, required: true },
  bloodGroup: {
    type: String,
    enum: ["A+", "A-", "B+", "B-", "AB+", "AB-", "O+", "O-"],
    required: true,
  },
  address: {
    street: { type: String, required: true },
    city: { type: String, required: true },
    state: { type: String, required: true },
    zipCode: { type: String, required: true },
  },
  emergencyContact: {
    name: { type: String, required: true },
    phoneNumber: { type: String, required: true, match: /^\d{10}$/ },
  },
  role: { type: String, enum: ["user", "admin"], default: "user" },
  createdAt: { type: Date, default: Date.now },
  updatedAt: { type: Date, default: Date.now },
});

// Middleware to update `updatedAt` before save
userSchema.pre("save", function (next) {
  this.updatedAt = Date.now();
  next();
});

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

// Signup Route
app.post("/api/signup", async (req, res) => {
  try {
    const {
      name,
      email,
      password,
      phone: phoneNumber,
      birthDate: dateOfBirth,
      bloodGroup,
      street,
      city,
      state,
      zipCode,
      emergencyName,
      emergencyPhone,
    } = req.body;

    // Create new user
    const newUser = new User({
      name,
      email,
      password: await bcrypt.hash(password, 10),
      phoneNumber,
      dateOfBirth,
      bloodGroup,
      address: {
        street,
        city,
        state,
        zipCode,
      },
      emergencyContact: {
        name: emergencyName,
        phoneNumber: emergencyPhone,
      },
    });

    await newUser.save();
    res.status(201).json({ message: "User registered successfully!" });
  } catch (error) {
    console.error("Signup error:", error);
    res.status(500).json({ error: "Signup failed." });
  }
});
```

```
app.post("/api/login", async (req, res) => {
  const { email, password } = req.body;

  try {
    console.log("Attempting login for email:", email);

    const user = await User.findOne({ email });
    console.log("User found:", user);

    if (!user) {
      console.log("User not found");
      return res.status(400).json({ error: "Invalid email or password."
});

    }

    // Check password
    const isPasswordValid = await bcrypt.compare(password, user.password);
    console.log("Password valid:", isPasswordValid);

    if (!isPasswordValid) {
      console.log("Password incorrect");
      return res.status(400).json({ error: "Invalid email or password."
});


    }

    // If successful, respond with user information
    res.json({
      message: "Login successful",
      user: { user },
    });
  } catch (error) {
    console.error("Login error:", error);
    res.status(500).json({ error: "Login failed." });
  }
});

// Export the app for testing purposes
module.exports = app;
```

```
// Start Server
if (require.main === module) {
  app.listen(PORT, () => {
    console.log(`Server is running on port ${PORT}`);
  });
}
```

Firstly this code provides the connection to the mongodb that we created using mongoose.

Here we create the user schema which will be stored in the database, which will be used to store user data on successful signup and this data is used for login purpose in the login page in the front end

This also has 2 apis where one is called by signup component in the frontend which upon success helps in storing the user details in the users schema in the database.

Another api is called by login component where email and password is provided to the api and this api checks if there is any user with the provided email and authenticate if the provided credentials match.

Test Cases for SignUp and Login:

```
const request = require("supertest");
const mongoose = require("mongoose");
const app = require("../server");

// beforeAll(async () => {
//   // Connect to the test database before running the tests
//   await mongoose.connect("mongodb://localhost:27017/test_db", {
//     useNewUrlParser: true,
//     useUnifiedTopology: true,
//   });
// });

// afterAll(async () => {
//   // Clean up and close the database connection after all tests
//   await mongoose.connection.dropDatabase(); // Optional: Drop test DB
after tests
//   await mongoose.connection.close();
// });
```

```
describe("POST /api/signup", () => {
  test("should successfully register a new user", async () => {
    const response = await request(app).post("/api/signup").send({
      name: "Test User",
      email: "testuser@example.com",
      password: "TestPassword123",
      phone: "1234567890",
      birthDate: "1990-01-01",
      bloodGroup: "O+",
      street: "123 Main St",
      city: "Test City",
      state: "Test State",
      zipCode: "12345",
      emergencyName: "Emergency Contact",
      emergencyPhone: "0987654321",
    });
    expect(response.statusCode).toBe(201);
    expect(response.body.message).toBe("User registered successfully!");
  });

  test("should fail to register if the email is already in use", async ()
=> {
    const response = await request(app).post("/api/signup").send({
      name: "Test User",
      email: "testuser@example.com",
      password: "TestPassword123",
      phone: "1234567890",
      birthDate: "1990-01-01",
      bloodGroup: "O+",
      street: "123 Main St",
      city: "Test City",
      state: "Test State",
      zipCode: "12345",
      emergencyName: "Emergency Contact",
      emergencyPhone: "0987654321",
    });
    expect(response.statusCode).toBe(500);
    expect(response.body.error).toBe("Signup failed.");
  });
});
```

```
describe("POST /api/login", () => {
  test("should successfully log in a user with correct credentials", async () => {
    const response = await request(app).post("/api/login").send({
      email: "testuser@example.com",
      password: "TestPassword123",
    });
    expect(response.statusCode).toBe(200);
    expect(response.body.message).toBe("Login successful");
  });

  test("should fail to log in a user with incorrect password", async () => {
    const response = await request(app).post("/api/login").send({
      email: "testuser@example.com",
      password: "WrongPassword123",
    });
    expect(response.statusCode).toBe(400);
    expect(response.body.error).toBe("Invalid email or password.");
  });

  test("should fail to log in a user with non-existing email", async () => {
    const response = await request(app).post("/api/login").send({
      email: "nonexistentuser@example.com",
      password: "SomePassword123",
    });
    expect(response.statusCode).toBe(400);
    expect(response.body.error).toBe("Invalid email or password.");
  });
});
```

This file is used for testing the apis in the server.js where various cases are given such as valid signup, valid login, invalid user, duplicate user signup, valid email but invalid password

bloodDonation.js:

```
import React, { useEffect, useState } from "react";
import axios from "axios";
import "../styling/bloodDonation.css";
import { useLocation } from "react-router-dom";

const BloodDonation = () => {
  const [users, setUsers] = useState([]);
  const [filteredUsers, setFilteredUsers] = useState([]);
  const [selectedBloodGroup, setSelectedBloodGroup] = useState("");
  const [flippedCards, setFlippedCards] = useState({});
  const location = useLocation();
  const { userData } = location.state || {};

  console.log(userData);
  useEffect(() => {
    const fetchBloodDonors = async () => {
      try {
        const response = await axios.get(
          "http://localhost:5000/api/bloodDonors"
        );

        // Filter out users with null userId and exclude the current
user
        const validUsers = response.data.filter(
          (user) => user.userId !== null && user.userId._id !==
userData._id
        );

        setUsers(validUsers);
        setFilteredUsers(validUsers); // Initialize filtered users
with all donors
      } catch (error) {
        console.error("Failed to load blood donors:", error);
      }
    };

    fetchBloodDonors();
  }, [userData]);

  console.log(users);

  // Filter users by blood group
  const handleBloodGroupChange = (event) => {
    const bloodGroup = event.target.value;
    setSelectedBloodGroup(bloodGroup);
    setFilteredUsers(
      bloodGroup
        ? users.filter((user) => user.bloodGroup === bloodGroup)
        : users
    );
  };

  // Handle card flip
  const handleRequestClick = (userId) => {
    setFlippedCards((prevFlippedCards) => ({
      ...prevFlippedCards,
      [userId]: !prevFlippedCards[userId],
    }));
  };

  return (
    <div className="blood-donation-container">
      <h2 className="blood-donation-title">Blood Donors</h2>
      <div className="filter-container">
        <label htmlFor="bloodGroup">Filter by Blood Group: </label>
        <select
          id="bloodGroup"
          value={selectedBloodGroup}
          onChange={handleBloodGroupChange}
        >
          <option value="">All</option>
          <option value="A+">A+</option>
          <option value="A-">A-</option>
          <option value="B+">B+</option>
          <option value="B-">B-</option>
          <option value="AB+">AB+</option>
          <option value="AB-">AB-</option>
          <option value="O+">O+</option>
          <option value="O-">O-</option>
        </select>
      </div>

      <div className="donor-cards-container">
        {filteredUsers.length > 0 ? (
          filteredUsers.map((user) => (
            <div className="donor-card" key={user._id}>
              <div
                className={`card-inner ${
                  flippedCards[user._id] ? "flipped" : ""
                }`}
              >
                {/* Front Side */}
                <div className="card-front">
                  <div className="donor-info">
                    <h3>{user.userId?.name}</h3>
                    <p>City: {user.userId?.address?.city}</p>
                    <p>Phone: {user.userId?.phoneNumber}</p>
                  </div>
                  <div className="blood-group-
chip">{user.bloodGroup}</div>
                  <button
                    className="request-button"
                    onClick={() => handleRequestClick(user._id)}
                  >
                    Request
                  </button>
                </div>

                {/* Back Side */}
                <div className="card-back">
                  <h3>Donor Details</h3>
                  <p>Name: {user.userId?.name}</p>
                  <p>Phone: {user.userId?.phoneNumber}</p>
                  <p>Street: {user.userId?.address?.street}</p>
                  <p>City: {user.userId?.address?.city}</p>
                  <p>State: {user.userId?.address?.state}</p>
                  <p className="bgrp">Blood Group:
{user.bloodGroup}</p>
                  <button
                    className="request-button"
                    onClick={() => handleRequestClick(user._id)}
                  >
                    Close
                  </button>
                </div>
              </div>
            </div>
          ))
        ) : (
          <p>No donors found for the selected blood group.</p>
        )}
      </div>
    </div>
  );
};
export default BloodDonation;
```

The BloodDonation component displays and filters blood donor profiles, allowing users to view detailed information by flipping each donor’s card.

docLandingPage.js:

```
import { React, useState, useEffect } from "react";
import "../styling/docLanding.css";
import { useLocation, useNavigate } from "react-router-dom";
import docAppointment from "../assets/docAppointment.png";
import doctors from "../assets/doctors.png";
import docNew from "../assets/docNew.png";
import axios from "axios";

const DoctorLandingPage = () => {
  const navigate = useNavigate();
  const location = useLocation();
  const { doctor } = location.state || {};
  const [appointments, setAppointments] = useState([]);
  const [error, setError] = useState("");

  // Additional state to store counts for appointment statuses
  const [appointmentCounts, setAppointmentCounts] = useState({
    totalAppointments: 0,
    totalScheduled: 0,
    totalCompleted: 0,
    totalCancelled: 0,
  });

  useEffect(() => {
    const fetchAppointments = async () => {
      try {
        const response = await axios.get(

`http://localhost:5000/api/appointments/doctor/${doctor._id}`
        );
        const appointments = response.data;

        // Calculate totals for each status
        const totalScheduled = appointments.filter(
          (app) => app.status === "Scheduled"
        ).length;
        const totalCompleted = appointments.filter(
          (app) => app.status === "completed"
        ).length;
        const totalCancelled = appointments.filter(
          (app) => app.status === "Cancelled"
        ).length;

        setAppointments(appointments);
        setAppointmentCounts({
          totalAppointments: appointments.length,
          totalScheduled,
          totalCompleted,
          totalCancelled,
        });
      } catch (error) {
        setError("Failed to load appointments. Please try again
later.");
      }
    };
    fetchAppointments();
  }, [doctor]);

  const handleViewAppointments = () => {
    navigate(`/docDocAppointments`, { state: { doctor } });
  };

  return (
    <div className="doctor-landing-container">
      <nav className="navbar">
        <div className="logo">
          <h2>eMed</h2>
        </div>
        <ul className="nav-links">
          <li>
            <a href="#appointments">Appointments</a>
          </li>
          <li>
            <a href="#contact">Contact</a>
          </li>
        </ul>
        <button className="btn-login">Logout</button>
      </nav>

      <section className="hero">
        <div className="hero-content">
          <h1>Welcome, Dr {doctor?.name || "Doctor"}</h1>
          <p>
            Thank you for your dedication and tireless service
in providing care
            to your patients. We deeply appreciate your efforts!
          </p>
        </div>
        <div className="hero-image">
          <img src={docNew} alt="Doctor at work" />
        </div>
      </section>

      <section className="metrics-section">
        <h2>Doctor Dashboard</h2>
        <div className="metrics-grid">
          <div className="metric-card">
            <h3>Total Appointments</h3>
            <p>{appointmentCounts.totalAppointments}</p>
          </div>
          <div className="metric-card">
            <h3>Upcoming Appointments</h3>
            <p>{appointmentCounts.totalScheduled}</p>
          </div>
          <div className="metric-card">
            <h3>Treated Patients</h3>
            <p>{appointmentCounts.totalCompleted}</p>
          </div>
          <div className="metric-card">
            <h3>Cancelled Appointments</h3>
            <p>{appointmentCounts.totalCancelled}</p>
          </div>
        </div>
      </section>

      <section id="appointments" className="features-section">
        <h2>Appointments</h2>
        <div
          style={{
            display: "flex",
            flexDirection: "row",
            justifyContent: "center",
            alignItems: "center",
          }}
        >
          <div>
            <p>Manage your appointments and stay organized for
better care.</p>
            <button className="btn-primary"
onClick={handleViewAppointments}>
              View Appointments
            </button>
          </div>
          <div>
            <img src={docAppointment} alt="Manage Appointments"
/>
          </div>
        </div>
      </section>

      <footer id="contact" className="footer">
        <p>&copy; 2024 eMed. All Rights Reserved.</p>
      </footer>
    </div>
  );
};

export default DoctorLandingPage;
```

docLogin.js:

```
import React, { useState } from "react";
import axios from "axios";
import "../styling/docLogin.css";
import { useNavigate } from "react-router-dom";

const DoctorLogin = () => {
  const [loginDetails, setLoginDetails] = useState({
    email: "",
    password: "",
  });
  const navigate = useNavigate();
  const [errorMessage, setErrorMessage] = useState("");
  const [successMessage, setSuccessMessage] = useState("");

  const handleLoginChange = (e) => {
    const { name, value } = e.target;
    setLoginDetails((prevDetails) => ({
      ...prevDetails,
      [name]: value,
    }));
  };

  const handleDocSignUp = () => {
    navigate(`/DocSignup`);
  };

  const handleLogin = async (e) => {
    e.preventDefault();
    try {
      const response = await axios.post(
        "http://localhost:5000/api/doctor/login",
        loginDetails
      );
      setSuccessMessage(response.data.message);
      navigate(`/docLanding`, { state: { doctor:
response.data.doctor } });
      setErrorMessage(""); // Clear error message
    } catch (error) {
      if (error.response && error.response.data) {
        setErrorMessage(error.response.data.error);
      } else {
        setErrorMessage("Doctor login failed.");
      }
    }
  };
```

```
return (
  <div className="login-container">
    <h2>Doctor Login</h2>
    <form className="login-form" onSubmit={handleLogin}>
      <label>Email</label>
      <input
        type="email"
        name="email"
        value={loginDetails.email}
        onChange={handleLoginChange}
        required
      />

      <label>Password</label>
      <input
        type="password"
        name="password"
        value={loginDetails.password}
        onChange={handleLoginChange}
        required
      />

      <button className="login-btn" type="submit">
        Login
      </button>
      <div
        style={{
          display: "flex",
          flexDirection: "row",
          marginTop: "20px",
          justifyContent: "center",
          alignItems: "center",
          gap: "10px",
        }}
      >
        <p>Create an account </p>
        <h3
          className="signup-btnnn"
          style={{ color: "blue" }}
          onClick={handleDocSignUp}
        >
          SignUp
        </h3>
      </div>

      {errorMessage && <p className="error-
message">{errorMessage}</p>}
      </form>
    </div>
  );
};

export default DoctorLogin;
```

docSignUp.js:

```
import React, { useState } from "react";
import axios from "axios";
import "../styling/docSignup.css";
import { useNavigate } from "react-router-dom";

const DoctorSignup = () => {
  const [signupDetails, setSignupDetails] = useState({
    name: "",
    email: "",
    password: "",
    phoneNumber: "",
    specialty: "",
    licenseNumber: "",
    qualifications: "",
    experience: "",
    clinicAddress: "",
    // availability: "",
    bloodDonor: false,
    emergencyName: "",
    emergencyPhone: "",
  });
  const navigate = useNavigate();
  const [errorMessage, setErrorMessage] = useState("");
  const [successMessage, setSuccessMessage] = useState("");

  const handleSignupChange = (e) => {
    const { name, value, type, checked } = e.target;
    setSignupDetails((prevDetails) => ({
      ...prevDetails,
      [name]: type === "checkbox" ? checked : value,
    }));
  };

  const handleSignup = async (e) => {
    e.preventDefault();
    console.log(signupDetails);
    try {
      const response = await axios.post(
        "http://localhost:5000/api/doctor/signup",
        signupDetails
      );
      setSuccessMessage(response.data.message);
      navigate(`/docLanding`, { state: { doctor: signupDetails }
});
      setErrorMessage(""); // Clear error message
    } catch (error) {
      if (error.response && error.response.data) {
        setErrorMessage(error.response.data.error);
      } else {
        setErrorMessage("Doctor signup failed.");
      }
    }
  };

  return (
    <div className="signup-container">
      <h2>Doctor Signup</h2>
      <form className="signup-form" onSubmit={handleSignup}>
        <label>Name</label>
        <input
          type="text"
          name="name"
          value={signupDetails.name}
          onChange={handleSignupChange}
          required
        />

        <label>Email</label>
        <input
          type="email"
          name="email"
          value={signupDetails.email}
          onChange={handleSignupChange}
          required
        />

        <label>Password</label>
        <input
          type="password"
          name="password"
          value={signupDetails.password}
          onChange={handleSignupChange}
          required
```

```
/>

<label>Phone Number</label>
<input
  type="text"
  name="phoneNumber"
  value={signupDetails.phoneNumber}
  onChange={handleSignupChange}
  required
/>

<label>Date of Birth</label>
<input
  type="date"
  name="birthDate"
  value={signupDetails.birthDate}
  onChange={handleSignupChange}
  required
/>

<label>Specialty</label>
<input
  type="text"
  name="specialty"
  value={signupDetails.specialty}
  onChange={handleSignupChange}
  required
/>

<label>License Number</label>
<input
  type="text"
  name="licenseNumber"
  value={signupDetails.licenseNumber}
  onChange={handleSignupChange}
  required
/>

<label>Qualifications</label>
<input
  type="text"
  name="qualifications"
  value={signupDetails.qualifications}
  onChange={handleSignupChange}
  required
/>
```

```
<label>Experience (Years)</label>
<input
  type="number"
  name="experience"
  value={signupDetails.experience}
  onChange={handleSignupChange}
  required
/>

<label>Clinic Address</label>
<input
  type="text"
  name="clinicAddress"
  value={signupDetails.clinicAddress}
  onChange={handleSignupChange}
  required
/>

{/* <label>Availability</label> */}
{/* <input
  type="text"
  name="availability"
  value={signupDetails.availability}
  onChange={handleSignupChange}
  required
/> */}

<div className="checkbox-label">
  <input
    type="checkbox"
    name="bloodDonor"
    checked={signupDetails.bloodDonor}
    onChange={handleSignupChange}
  />
  <label>Sign up for blood donation</label>
</div>

<label>Emergency Contact Name</label>
<input
  type="text"
  name="emergencyName"
  value={signupDetails.emergencyName}
  onChange={handleSignupChange}
  required
/>

<label>Emergency Contact Phone</label>
```

```
        <input
          type="text"
          name="emergencyPhone"
          value={signupDetails.emergencyPhone}
          onChange={handleSignupChange}
          required
        />

        <button className="signup-btn" type="submit">
          Sign Up
        </button>
      </form>
    </div>
  );
};

export default DoctorSignup;
```

This code implements a doctor login, signup, and landing page for an application where doctors can log in, register, and view a dashboard with their appointment stats. The landing page shows metrics for total, scheduled, completed, and canceled appointments, and provides options for managing appointments.

PaymentForm.js:

```
import React, { useState } from "react";
import { CardElement, useStripe, useElements } from "@stripe/react-
stripe-js";
import axios from "axios";
import { useLocation, useNavigate } from "react-router-dom";

const PaymentForm = () => {
  const stripe = useStripe();
  const elements = useElements();
  const navigate = useNavigate();
  const [error, setError] = useState(null);
  const [isLoading, setIsLoading] = useState(false);
  const [successMessage, setSuccessMessage] = useState("");
  const { state } = useLocation();
  const amount = Math.round(state?.totalPrice * 100);
  const userData = state.userData;

  const handlePayment = async (event) => {
    event.preventDefault();
    setIsLoading(true);
    try {
      const {
        data: { clientSecret },
      } = await axios.post(
        "http://localhost:5000/api/create-payment-intent",
        { amount },
        { headers: { "Content-Type": "application/json" } }
      );

      const cardElement = elements.getElement(CardElement);

      if (!clientSecret || !cardElement) {
        setError("Failed to load payment method.");
        setIsLoading(false);
        return;
      }

      const paymentResult = await
stripe.confirmCardPayment(clientSecret, {
        payment_method: {
          card: cardElement,
          billing_details: {
            name: userData?.name,
            email: userData?.email,
          },
        },
      });

      if (paymentResult.error) {
        setError(paymentResult.error.message);
      } else if (paymentResult.paymentIntent.status === "succeeded") {
        setError(null);
        setSuccessMessage(
          `Payment of ${state?.totalPrice} is successful! Redirecting
to your dashboard... \n Your order will be delivered to
${userData?.address?.street}, ${userData?.address?.city}`
        );

        setTimeout(() => {
          navigate("/landing", { state: { userData } });
        }, 6000); // Redirect after 3 seconds
      }
    } catch (error) {
      console.error("Payment error:", error);
      setError("Payment failed. Please try again.");
    }
    setIsLoading(false);
  };

  return (
    <div>
      {successMessage ? (
        <div
          style={{
            color: "green",
            fontSize: "18px",
            textAlign: "center",
            marginTop: "20px",
          }}
        >
          {successMessage}
        </div>
      ) : (
        <div>
          <h2>Complete Payment</h2>
          <form onSubmit={handlePayment}>
            <CardElement />
            {error && (
              <div style={{ color: "red", marginTop: "10px"
}}>{error}</div>
            )}
            <button
              type="submit"
              disabled={!stripe || isLoading}
              style={{ marginTop: "20px" }}
            >
              {isLoading ? "Processing..." : "Pay Now"}
            </button>
          </form>
        </div>
      )}
    </div>
  );
};

export default PaymentForm;
```

This code defines a payment form component using Stripe's API to handle credit card payments, allowing users to enter card details and process the payment. Upon successful payment, a success message with order details is displayed, and the user is redirected to the dashboard.

checkoutConfirmation.js:

```
// CheckoutConfirmation.js
import React from "react";
import { useLocation, useNavigate } from "react-router-dom";
import "../styling/checkoutConfirmation.css"; // Add CSS for
this component

const CheckoutConfirmation = () => {
  const { state } = useLocation();
  const navigate = useNavigate();
  const { cart, userData, totalPrice } = state;

  const handleConfirmOrder = () => {
    alert("Thank you! Your order has been confirmed.");
    navigate("/landing", { state: { userData } });
  };

  return (
    <div className="checkout-confirmation-container">
      <h2>Order Confirmation</h2>
      <p>
        Thank you for shopping with us, {userData.name}. Here is
a summary of
        your order.
      </p>
      <div className="order-summary">
        <ul>
          {cart.map((item) => (
            <li key={item.id}>
              <h3>{item.name}</h3>
              <p>Quantity: {item.quantity}</p>
              <p>Subtotal: Rs {item.price * item.quantity}</p>
            </li>
          ))}
        </ul>
        <h3>Total: Rs {totalPrice}</h3>
      </div>
      <button onClick={handleConfirmOrder}>Confirm
Order</button>
      <button onClick={() => navigate("/cart", { state: { cart,
userData } })}>
        Edit Order
      </button>
    </div>
  );
};

export default CheckoutConfirmation;
```

This code defines a checkout confirmation screen where the user sees a summary of their cart, including item details and total cost, with options to confirm the order or edit it. Upon confirmation, an alert message is shown, and the user is redirected to the landing page.

AdminLogin

```
// AdminLogin.js
import React, { useState } from "react";
import { useNavigate } from "react-router-dom";
import "../styling/adminLogin.css";

const AdminLogin = () => {
  const [username, setUsername] = useState("");
  const [password, setPassword] = useState("");
  const [error, setError] = useState(null);
  const navigate = useNavigate();

  const ADMIN_USERNAME = "adminEmed";
  const ADMIN_PASSWORD = "123456789";

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

    if (username === ADMIN_USERNAME && password === ADMIN_PASSWORD) {
      navigate("/admin");
    } else {
      setError("Invalid credentials. Please try again.");
    }
  };

  return (
    <div className="admin-login-container">
      <h2>Admin Login</h2>
      {error && <p className="error-message">{error}</p>}
      <form onSubmit={handleLogin}>
        <label>
          Username:
          <input
            type="text"
            value={username}
            onChange={(e) => setUsername(e.target.value)}
            required
          />
        </label>
        <label>
          Password:
          <input
            type="password"
            value={password}
            onChange={(e) => setPassword(e.target.value)}
            required
          />
        </label>
        <button type="submit">Login</button>
      </form>
    </div>
  );
};

export default AdminLogin;
```

AdminLandingPage:

```
import React, { useEffect, useState } from "react";
import { useNavigate } from "react-router-dom";
import axios from "axios";
import "../styling/adminLanding.css"; // Assuming you have a CSS file
for styling

const AdminLandingPage = () => {
  const navigate = useNavigate();
  const [metrics, setMetrics] = useState({
    totalDoctors: 0,
    totalPatients: 0,
    totalAppointments: 0,
    totalBloodDonors: 0,
    scheduledAppointments: 0,
    completedAppointments: 0,
    cancelledAppointments: 0,
  });

  useEffect(() => {
    const fetchMetrics = async () => {
      try {
        const response = await axios.get(
          "http://localhost:5000/api/admin/metrics"
        );
        setMetrics(response.data);
      } catch (error) {
        console.error("Failed to load admin metrics:", error);
      }
    };

    fetchMetrics();
  }, []);

  return (
    <div className="admin-landing-container">
      <header className="admin-header">
        <h1>Admin Dashboard</h1>
        <button className="btn-logout" onClick={() => navigate("/")}>
          Logout
        </button>
      </header>

      {/* Metrics Section */}
      <section className="metrics-section">
        <h2>Overview</h2>
        <div className="metrics-grid">
          <div className="metric-card">
            <h3>Total Doctors</h3>
            <p>{metrics.totalDoctors}</p>
          </div>
          <div className="metric-card">
            <h3>Total Patients</h3>
            <p>{metrics.totalPatients}</p>
          </div>
          <div className="metric-card">
            <h3>Total Appointments</h3>
            <p>{metrics.totalAppointments}</p>
          </div>
          <div className="metric-card">
            <h3>Total Blood Donors</h3>
            <p>{metrics.totalBloodDonors}</p>
          </div>
          <div className="metric-card">
            <h3>Scheduled Appointments</h3>
            <p>{metrics.scheduledAppointments}</p>
          </div>
          <div className="metric-card">
            <h3>Completed Appointments</h3>
            <p>{metrics.completedAppointments}</p>
          </div>
          <div className="metric-card">
            <h3>Cancelled Appointments</h3>
            <p>{metrics.cancelledAppointments}</p>
          </div>
        </div>
      </section>

      {/* Management Sections */}
      <section className="management-section">
        <h2>Manage</h2>
        <div className="management-grid">
          <div
            className="management-card"
            onClick={() => navigate("/admin/doctors")}
          >
            <h3>Manage Doctors</h3>
            <p>View and manage all doctors in the system.</p>
          </div>
          <div
            className="management-card"
            onClick={() => navigate("/admin/userList")}
```

```
          >
            <h3>Manage Patients</h3>
            <p>View and manage all patients in the system.</p>
          </div>
          <div
            className="management-card"
            onClick={() => navigate("/admin/appointments")}
          >
            <h3>Manage Appointments</h3>
            <p>Track and manage all appointments.</p>
          </div>
          <div
            className="management-card"
            onClick={() => navigate("/admin/bloodDonors")}
          >
            <h3>Blood Donors</h3>
            <p>View and manage registered blood donors.</p>
          </div>
          <div
            className="management-card"
            onClick={() => navigate("/admin/reports")}
          >
            <h3>Generate Reports</h3>
            <p>Generate reports for appointments and donor
requests.</p>
          </div>
        </div>
      </section>
    </div>
  );
};

export default AdminLandingPage;
```

AdminUsers:

```
import React, { useEffect, useState } from "react";
import axios from "axios";
import "../styling/adminUsers.css"; // Add your CSS file for styling

const AdminUsers = () => {
  const [users, setUsers] = useState([]);
  const [error, setError] = useState(null);

  // Fetch all users from the API
  useEffect(() => {
    const fetchUsers = async () => {
      try {
        const response = await
axios.get("http://localhost:5000/api/users");
        setUsers(response.data);
      } catch (error) {
        console.error("Failed to fetch users:", error);
        setError("Failed to load users. Please try again later.");
      }
    };

    fetchUsers();
  }, []);

  return (
    <div className="admin-users-container">
      <h2 className="admin-users-title">All Registered Users</h2>
      {error && <p className="error-message">{error}</p>}

      <div className="user-cards-container">
        {users.length > 0 ? (
          users.map((user) => (
            <div className="user-card" key={user._id}>
              <h3>{user.name}</h3>
              <p>Email: {user.email}</p>
              <p>Phone: {user.phoneNumber}</p>
              <p>
                Address: {user.address?.city}, {user.address?.state}
              </p>
              <p>Blood Group: {user.bloodGroup || "N/A"}</p>
              <p>
                Emergency Contact: {user.emergencyContact?.name} -{"
"}
                {user.emergencyContact?.phoneNumber}
              </p>
            </div>
          ))
        ) : (
          <p>No users found.</p>
        )}
      </div>
    </div>
  );
};

export default AdminUsers;
```

AdminDoctors:

```
import React, { useEffect, useState } from "react";
import axios from "axios";
import "../styling/adminDoctors.css"; // Assuming you have a CSS file
for styling

const AdminDoctors = () => {
  const [doctors, setDoctors] = useState([]);
  const [error, setError] = useState(null);

  // Fetch all doctors from the API
  useEffect(() => {
    const fetchDoctors = async () => {
      try {
        const response = await
axios.get("http://localhost:5000/api/doctors");
        setDoctors(response.data);
      } catch (error) {
        console.error("Failed to fetch doctors:", error);
        setError("Failed to load doctors. Please try again later.");
      }
    };

    fetchDoctors();
  }, []);

  return (
    <div className="admin-doctors-container">
      <h2 className="admin-doctors-title">All Registered Doctors</h2>
      {error && <p className="error-message">{error}</p>}

      <div className="doctor-cards-container">
        {doctors.length > 0 ? (
          doctors.map((doctor) => (
            <div className="doctor-card" key={doctor._id}>
              <h3>{doctor.name}</h3>
              <p>Email: {doctor.email}</p>
              <p>Specialty: {doctor.specialty || "N/A"}</p>
              <p>Phone: {doctor.phoneNumber}</p>
              <p>Clinic Address: {doctor.clinicAddress}</p>
              <p>Experience: {doctor.experience} years</p>
            </div>
          ))
        ) : (
          <p>No doctors found.</p>
        )}
      </div>
    </div>
  );
};

export default AdminDoctors;
```

The admin can log in and access a dashboard, view and manage metrics like total doctors, patients, appointments, and blood donors, and generate reports. They can oversee all registered users and doctors, track appointment statuses, and handle system-wide data management tasks.

DoctorAppointments:

```
import React, { useEffect, useState } from "react";
import axios from "axios";
import { useLocation } from "react-router-dom";
import { Button, TextField } from "@mui/material";
import DatePicker from "react-datepicker";
import "react-datepicker/dist/react-datepicker.css";
import "../styling/docDoctorAppointments.css";

const DocDoctorAppointments = () => {
  const location = useLocation();
  const { doctor } = location.state || {};
  const [appointments, setAppointments] = useState([]);
  const [error, setError] = useState(null);
  const [filteredAppointments, setFilteredAppointments] =
useState([]);
  const [startDate, setStartDate] = useState(null);
  const [endDate, setEndDate] = useState(null);

  useEffect(() => {
    const fetchAppointments = async () => {
      try {
        const response = await axios.get(

`http://localhost:5000/api/appointments/doctor/${doctor._id}`
        );
        setAppointments(response.data);
        setFilteredAppointments(response.data); // Initialize with all
appointments
      } catch (error) {
        setError("Failed to load appointments. Please try again
later.");
      }
    };

    fetchAppointments();
  }, [doctor]);

  useEffect(() => {
    if (startDate && endDate) {
      const filtered = appointments.filter((appointment) => {
        const appointmentDate = new Date(appointment.date);
        return appointmentDate >= startDate && appointmentDate <=
endDate;
      });
      setFilteredAppointments(filtered);
    } else {
      setFilteredAppointments(appointments);
    }
  }, [startDate, endDate, appointments]);

  const handleCancelAppointment = async (appointmentId) => {
    const cancellationNote = prompt("Please enter a cancellation
note:");
    if (!cancellationNote) return;

    try {
      await axios.put(

`http://localhost:5000/api/appointments/${appointmentId}/cancel`,
        { status: "Cancelled", notes: cancellationNote }
      );

      setAppointments((prevAppointments) =>
        prevAppointments.map((appointment) =>
          appointment._id === appointmentId
            ? { ...appointment, status: "Cancelled", notes:
cancellationNote }
            : appointment
        )
      );
    } catch (error) {
      setError("Failed to cancel appointment. Please try again.");
    }
  };

  const scheduledAppointments = filteredAppointments.filter(
    (appointment) => appointment.status === "Scheduled"
  );
  const completedAppointments = filteredAppointments.filter(
    (appointment) => appointment.status === "completed"
  );
  const cancelledAppointments = filteredAppointments.filter(
    (appointment) => appointment.status === "Cancelled"
  );

  return (
    <div className="appointments-container">
      <h2 className="appointments-title">Doctor's Appointments</h2>
      {error && <p className="error-message">{error}</p>}

      {/* Date filter section */}
      <div className="date-filter">
        <DatePicker
          selected={startDate}
          onChange={(date) => setStartDate(date)}
          selectsStart
          startDate={startDate}
          endDate={endDate}
          placeholderText="Start Date"
          customInput={<TextField label="Start Date"
variant="outlined" />}
        />
        <DatePicker
          selected={endDate}
          onChange={(date) => setEndDate(date)}
          selectsEnd
          startDate={startDate}
          endDate={endDate}
          minDate={startDate}
          placeholderText="End Date"
          customInput={<TextField label="End Date" variant="outlined"
/>}
        />
        <Button
          variant="contained"
          onClick={() => {
            setStartDate(null);
            setEndDate(null);
          }}
        >
          Clear Filter
        </Button>
      </div>

      {/* Appointments sections */}
      <div className="appointments-section">
        <h3 className="appointment-section-title">Scheduled</h3>
        <div className="appointment-section">
          {scheduledAppointments.length > 0 ? (
            <div className="appointment-cards-container">
              {scheduledAppointments.map((appointment) => (
                <div className="appointment-card"
key={appointment._id}>
                  <div className="appointment-date">
                    {new Date(appointment.date).toLocaleDateString()}
                  </div>
                  <div className="appointment-info">
                    <p>Time: {appointment.time}</p>
                    <p>Patient: {appointment.patientName}</p>
                    <p>Notes: {appointment.notes || "No notes
available"}</p>
                  </div>
                  <div className="status-chip status-
scheduled">Scheduled</div>
                  <button
                    className="cancel-button"
                    onClick={() =>
handleCancelAppointment(appointment._id)}
                  >
                    Cancel Appointment
                  </button>
                </div>
              ))}
            </div>
          ) : (
            <p>No scheduled appointments.</p>
          ))}
        </div>
      </div>

      {/* Completed and Cancelled sections */}
      <div className="appointments-section">
        <h3 className="appointment-section-title">Completed</h3>
        {completedAppointments.length > 0 ? (
          <div className="appointment-cards-container">
            {completedAppointments.map((appointment) => (
              <div className="appointment-card" key={appointment._id}>
                <div className="appointment-date">
                  {new Date(appointment.date).toLocaleDateString()}
                </div>
                <div className="appointment-info">
                  <p>Time: {appointment.time}</p>
                  <p>Patient: {appointment.patientName}</p>
                  <p>Notes: {appointment.notes || "No notes
available"}</p>
                </div>
                <div className="status-chip status-
completed">Completed</div>
              </div>
            ))}
          </div>
        ) : (
          <p>No completed appointments.</p>
        )}
      </div>

      <div className="appointments-section">
        <h3 className="appointment-section-title">Cancelled</h3>
        {cancelledAppointments.length > 0 ? (
          <div className="appointment-cards-container">
            {cancelledAppointments.map((appointment) => (
              <div className="appointment-card" key={appointment._id}>
                <div className="appointment-date">
                  {new Date(appointment.date).toLocaleDateString()}
                </div>
                <div className="appointment-info">
                  <p>Time: {appointment.time}</p>
                  <p>Patient: {appointment.patientName}</p>
                  <p>Notes: {appointment.notes || "No notes
available"}</p>
                </div>
                <div className="status-chip status-
cancelled">Cancelled</div>
              </div>
            ))}
          </div>
        ) : (
          <p>No cancelled appointments.</p>
      )}
    </div>
  </div>
 );
};

export default DocDoctorAppointments;
```

This page displays doctor's appointments, allowing them to filter appointments by date and view them by status as scheduled or completed or canceled. This also allows doctors to cancel scheduled appointments, with the option to add a cancellation note.

RoleSelection:

```
import React from "react";
import "../styling/roleSelection.css";
import { useNavigate } from "react-router-dom";
import { Button } from "@mui/material";
const RoleSelection = () => {
  const navigate = useNavigate();
  const handlePatientClick = () => {
    navigate(`/PatientLanding`);
  };

  const handleDocClick = () => {
    navigate(`/docLogin`);
  };

  const handleAdminClick = () => {
    navigate(`/adminLogin`);
  };
  return (
    <div className="container">
      <div className="left-half left">
        <button className="doctor-btn" onClick={handleDocClick}>
          Doctor
        </button>
      </div>
      <div>
        <button className="admin-btn" onClick={handleAdminClick}>
          Admin
        </button>
      </div>
      <div className="right-half right">
        <button className="patient-btn" onClick={handlePatientClick}>
          Patient
        </button>
      </div>
    </div>
  );
};

export default RoleSelection;
```

This page allows users to choose their role either as "Doctor", "Admin", or "Patient". Based on the selection, the user is then navigated to the respective login page.

BACKEND:

scheduler.js:

```
const cron = require("node-cron");
const Appointment = require("../backend/models/appointments");
const mongoose = require("mongoose");

// Function to update appointment status
const updateAppointmentStatus = async () => {
  const currentDate = new Date();

  try {
    // Find appointments to be updated
    const appointmentsToUpdate = await Appointment.find({
      date: { $lt: currentDate },
      status: "Scheduled",
    });
    console.log("Appointments to update:", appointmentsToUpdate);

    // Update the status of past appointments
    const updatedAppointments = await Appointment.updateMany(
      { date: { $lt: currentDate }, status: "Scheduled" },
      { $set: { status: "Completed" } }
    );

    console.log(
      `Updated ${updatedAppointments.modifiedCount} appointments to
'Completed'`
    );
  } catch (error) {
    console.error("Error updating appointment statuses:", error);
  }
};

// Schedule the job to run daily at midnight
cron.schedule("0 0 * * *", updateAppointmentStatus);

module.exports = updateAppointmentStatus;
```

This code schedules a job to run daily at midnight, using `node-cron`, to update the status of past appointments in a MongoDB database from "Scheduled" to "Completed" if the appointment date is before the current date. It fetches and logs the appointments to be updated, then logs the count of successfully updated appointments or errors, if any occur.

appointments.js:

Models:

```
const mongoose = require("mongoose");

const appointmentSchema = new mongoose.Schema({
  patientId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: "User",
    required: true,
  },
  doctorId: {
    type: mongoose.Schema.Types.ObjectId,
    ref: "Doctor",
    required: true,
  },
  date: {
    type: Date,
    required: true,
  },
  time: {
    type: String,
    required: true,
  },
  status: {
    type: String,
    enum: ["Scheduled", "Completed", "Cancelled"],
    default: "Scheduled",
  },
  notes: {
    type: String,
    default: "",
  },
  createdAt: {
    type: Date,
    default: Date.now,
  },
});
```

```
const Appointment = mongoose.model("Appointment", appointmentSchema);

module.exports = Appointment;
```

This code defines a Mongoose schema and model for an `Appointment` in a MongoDB database, with fields for patient and doctor IDs, date, time, status, and notes. The `status` field has options for "Scheduled," "Completed," and "Cancelled," with "Scheduled" as the default, and each appointment is timestamped with a `createdAt` date.

Routes:

```
const express = require("express");

const Appointment = require("../models/appointments");
const Doctor = require("../models/doc");
const User = require("../models/user");

const router = express.Router();

router.post("/api/appointments", async (req, res) => {
  try {
    const { patientId, doctorId, date, time, notes } = req.body;

    // Check if the user (patient) exists
    const patient = await User.findById(patientId);
    if (!patient) {
      return res.status(404).json({ error: "Patient not found" });
    }

    // Check if the doctor exists
    const doctor = await Doctor.findById(doctorId);
    if (!doctor) {
      return res.status(404).json({ error: "Doctor not found" });
    }

    // Create a new appointment
    const newAppointment = new Appointment({
      patientId,
      doctorId,
      date,
      time,
      notes,
    });

    await newAppointment.save();

    res.status(201).json({
      message: "Appointment successfully created",
      appointment: newAppointment,
    });
  } catch (error) {
    console.error("Appointment creation error:", error);
    res.status(500).json({ error: "Failed to create appointment" });
  }
});
router.get("/api/appointments/user/:userId", async (req, res) => {
  const { userId } = req.params;

  try {
    // Find appointments based on the user ID
    const appointments = await Appointment.find({ patientId: userId
}).populate(
      "doctorId",
      "name specialty"
    );

    if (!appointments.length) {
      return res
        .status(404)
        .json({ error: "No appointments found for this user." });
    }

    // Send appointment details including doctor name and specialty
    res.json(
      appointments.map((appointment) => ({
        _id: appointment._id,
        date: appointment.date,
        time: appointment.time,
        notes: appointment.notes,
        doctorName: appointment.doctorId.name,
        doctorSpecialty: appointment.doctorId.specialty,
        status: appointment.status,
      }))
    );
  } catch (error) {
    console.error("Error fetching appointments:", error);
    res
      .status(500)
      .json({ error: "An error occurred while fetching appointments."
});
  }
});

router.get("/api/appointments/doctor/:doctorId", async (req, res) => {
  const { doctorId } = req.params;

  try {
    // Find appointments based on the doctor ID
    const appointments = await Appointment.find({ doctorId
}).populate(
      "patientId",
      "name email"
    );

    if (!appointments.length) {
      return res
        .status(404)
        .json({ error: "No appointments found for this doctor." });
    }

    // Send appointment details including patient name and email
    res.json(
      appointments.map((appointment) => ({
        _id: appointment._id,
        date: appointment.date,
        time: appointment.time,
        notes: appointment.notes,
        patientName: appointment.patientId.name,
        patientEmail: appointment.patientId.email,
        status: appointment.status,
      }))
    );
  } catch (error) {
    console.error("Error fetching appointments:", error);
    res
      .status(500)
      .json({ error: "An error occurred while fetching appointments."
});
  }
});

router.put("/api/appointments/:appointmentId/cancel", async (req, res)
=> {
  const { appointmentId } = req.params;
  const { status, notes } = req.body;

  try {
    // Check if the appointment exists
    const appointment = await Appointment.findById(appointmentId);
    if (!appointment) {
      return res.status(404).json({ error: "Appointment not found."
});
    }

    // Update the status and notes
    appointment.status = status || "Cancelled"; // Default to
"Cancelled"
    if (notes) {
      appointment.notes = notes;
    }
    // Save the updated appointment
    await appointment.save();

    res.json({
      message: "Appointment updated successfully.",
      appointment: {
        _id: appointment._id,
        date: appointment.date,
        time: appointment.time,
        status: appointment.status,
        notes: appointment.notes,
      },
    });
  } catch (error) {
    console.error("Error updating appointment:", error);
    res.status(500).json({ error: "Failed to update appointment." });
  }
});

module.exports = router;
```

This code defines API routes for creating, retrieving, and updating appointments in a healthcare system. The `POST /api/appointments` route creates a new appointment by checking if the patient and doctor exist, while the `GET /api/appointments/user/:userId` and `GET /api/appointments/doctor/:doctorId` routes retrieve appointments for a specific user or doctor, respectively, including related details like the doctor's name or patient's contact information; the `PUT /api/appointments/:appointmentId/cancel` route allows updating the appointment status to "Cancelled" with optional notes.

bloodDonor.js:

MODELS:

```
const mongoose = require("mongoose");
const bloodDonorSchema = new mongoose.Schema({
  userId: { type: mongoose.Schema.Types.ObjectId, ref: "User",
required: true },
  bloodGroup: {
    type: String,
    enum: ["A+", "A-", "B+", "B-", "AB+", "AB-", "O+", "O-"],
    required: true,
  },
  lastDonationDate: { type: Date },
  availabilityStatus: { type: Boolean, default: true },
});

const BloodDonor = mongoose.model("BloodDonor", bloodDonorSchema);
module.exports = BloodDonor;
```

This code creates a Mongoose schema and model for a `BloodDonor` document, with fields for the donor's user ID, blood group, last donation date, and availability status. The `bloodGroup` field is restricted to common blood types, and `availabilityStatus` defaults to `true`, indicating the donor is available.

ROUTES:

```
const express = require("express");
const BloodDonor = require("../models/bloodDonor");
const router = express.Router();

// Get all blood donors or filter by blood group
router.get("/api/bloodDonors", async (req, res) => {
  try {
    const { bloodGroup } = req.query; // Optional query parameter
    const filter = bloodGroup ? { bloodGroup } : {};
    const bloodDonors = await BloodDonor.find(filter).populate(
      "userId",
      "name address phoneNumber"
    );
    res.status(200).json(bloodDonors);
  } catch (error) {
    console.error("Error fetching blood donors:", error);
    res.status(500).json({ error: "Failed to fetch blood donors." });
  }
});

// Add a new blood donor
router.post("/api/bloodDonors", async (req, res) => {
  try {
    const { userId, bloodGroup, lastDonationDate, availabilityStatus }
=
      req.body;
    const newBloodDonor = new BloodDonor({
      userId,
      bloodGroup,
      lastDonationDate,
      availabilityStatus,
    });
    await newBloodDonor.save();
    res.status(201).json(newBloodDonor);
  } catch (error) {
    console.error("Error adding blood donor:", error);
    res.status(500).json({ error: "Failed to add blood donor." });
  }
});

// Update a blood donor's availability or last donation date
router.put("/api/bloodDonors/:id", async (req, res) => {
  try {
    const { id } = req.params;
    const { lastDonationDate, availabilityStatus } = req.body;
    const updatedDonor = await BloodDonor.findByIdAndUpdate(
      id,
      { lastDonationDate, availabilityStatus },
      { new: true }
    );
    if (!updatedDonor)
      return res.status(404).json({ error: "Blood donor not found."
});
    res.status(200).json(updatedDonor);
  } catch (error) {
    console.error("Error updating blood donor:", error);
    res.status(500).json({ error: "Failed to update blood donor." });
  }
});

module.exports = router;
```

This code defines routes for managing blood donors. The `GET /api/bloodDonors` route retrieves all blood donors or filters them by blood group, while the `POST /api/bloodDonors` route adds a new blood donor, and the `PUT /api/bloodDonors/:id` route updates a blood donor's availability status or last donation date.

user.js:

```
const mongoose = require("mongoose");

const userSchema = new mongoose.Schema({
  name: { type: String, required: true },
  email: { type: String, required: true, unique: true, match:
/.+\@.+\..+/ },
  password: { type: String, required: true },
  phoneNumber: { type: String, required: true, match: /^\d{10}$/ },
  dateOfBirth: { type: Date, required: true },
  bloodGroup: {
    type: String,
    enum: ["A+", "A-", "B+", "B-", "AB+", "AB-", "O+", "O-"],
    required: true,
  },
  address: {
    street: { type: String, required: true },
    city: { type: String, required: true },
    state: { type: String, required: true },
    zipCode: { type: String, required: true },
  },
  emergencyContact: {
    name: { type: String, required: true },
    phoneNumber: { type: String, required: true, match: /^\d{10}$/ },
  },
  isDonor: { type: Boolean, default: false }, // Field for blood
donation option
  createdAt: { type: Date, default: Date.now },
  updatedAt: { type: Date, default: Date.now },
});

const User = mongoose.model("User", userSchema);
module.exports = User;
```

This code defines a Mongoose schema and model for a `User` document, which includes fields like name, email, password, phone number, date of birth, blood group, address, emergency contact, and donor status. The schema enforces validation, such as unique and formatted emails, 10-digit phone numbers, and a specific set of values for blood group, with timestamps for account creation and updates.

ROUTES:

```
const express = require("express");
const bcrypt = require("bcrypt");
const User = require("../models/user");

const router = express.Router(); // Use express.Router()

// Signup Route
const BloodDonor = require("../models/bloodDonor");

router.post("/api/signup", async (req, res) => {
  try {
    const {
      name,
      email,
      password,
      phone: phoneNumber,
      birthDate: dateOfBirth,
      bloodGroup,
      street,
      city,
      state,
      zipCode,
      emergencyName,
      emergencyPhone,
      isDonor, // Added field
    } = req.body;

    // Create new user
    const newUser = new User({
      name,
      email,
      password: await bcrypt.hash(password, 10),
      phoneNumber,
      dateOfBirth,
      bloodGroup,
      address: {
        street,
        city,
        state,
        zipCode,
      },
      emergencyContact: {
        name: emergencyName,
        phoneNumber: emergencyPhone,
      },
      isDonor,
    });

    await newUser.save();

    // If the user opts to be a blood donor
    if (isDonor) {
      const newDonor = new BloodDonor({
        userId: newUser._id,
        bloodGroup: newUser.bloodGroup,
      });
      await newDonor.save();
    }

    res.status(201).json({ message: "User registered successfully!"
});
  } catch (error) {
    console.error("Signup error:", error);
    res.status(500).json({ error: "Signup failed." });
  }
});
router.get("/api/users", async (req, res) => {
  try {
    // Fetch all users from the database
    const users = await User.find({});

    // Return the list of users
    res.status(200).json(users);
  } catch (error) {
    console.error("Error fetching users:", error);
    res.status(500).json({ error: "Failed to fetch users." });
  }
});
// Login Route
router.post("/api/login", async (req, res) => {
  const { email, password } = req.body;

  try {
    console.log("Attempting login for email:", email);
    console.log("Entered password:", await bcrypt.hash(password, 10));
// Log the plain text password

    const user = await User.findOne({ email });
    console.log("User found:", user);

    if (!user) {
      console.log("User not found");
      return res.status(400).json({ error: "Invalid email or
password." });
    }

    // Check password
    const isPasswordValid = await bcrypt.compare(password,
user.password);
    console.log("Stored hashed password:", user.password); // Log
hashed password from DB
    console.log("Password valid:", isPasswordValid);

    if (!isPasswordValid) {
      console.log("Password incorrect");
      return res.status(400).json({ error: "Incorrect Password" });
    }

    // If successful, respond with user information
    res.json({
      message: "Login successful",
      user: { user },
    });
  } catch (error) {
    console.error("Login error:", error);
    res.status(500).json({ error: "Login failed." });
  }
});

module.exports = router; // Fix the module export
```

This code defines routes for user registration, login, and fetching user information. The `POST /api/signup` route handles user registration, encrypts the password using bcrypt, and optionally registers the user as a blood donor, while the `GET /api/users` route retrieves and returns all users from the database, and the `POST /api/login` route authenticates a user by comparing the entered password with the stored hashed password.

admin.js

Routes:

```
// backend/routes/adminRoutes.js
const express = require("express");
const Appointment = require("../models/appointments");
const Doctor = require("../models/doc");
const User = require("../models/user");
const BloodDonor = require("../models/bloodDonor");

const router = express.Router();

// GET /api/admin/metrics
// Fetch total counts for doctors, patients, appointments, blood
donors, and appointment statuses
router.get("/api/admin/metrics", async (req, res) => {
  try {
    const totalDoctors = await Doctor.countDocuments();
    const totalPatients = await User.countDocuments();
    const totalAppointments = await Appointment.countDocuments();
    const totalBloodDonors = await BloodDonor.countDocuments();

    const scheduledAppointments = await Appointment.countDocuments({
      status: "Scheduled",
    });
    const completedAppointments = await Appointment.countDocuments({
      status: "completed",
    });
    const cancelledAppointments = await Appointment.countDocuments({
      status: "Cancelled",
    });

    res.json({
      totalDoctors,
      totalPatients,
      totalAppointments,
      totalBloodDonors,
      scheduledAppointments,
      completedAppointments,
      cancelledAppointments,
    });
  } catch (error) {
    console.error("Error fetching metrics:", error);
    res.status(500).json({ message: "Error fetching metrics data." });
  }
});

module.exports = router;
```

This code defines a `GET /api/admin/metrics` route for fetching various metrics related to the system, such as the total count of doctors, patients, appointments, blood donors, and the status breakdown of appointments (scheduled, completed, and cancelled). The route returns this data in JSON format for administrative purposes.