NavigationBar.jsx

```
import React, { useContext, useState } from 'react';
import { Navbar, Nav, Container } from 'react-bootstrap';
import { NavLink, useNavigate } from 'react-router-dom';
import { LinkContainer } from 'react-router-bootstrap';
import { AuthContext } from '../contexts/AuthContext';
import logo from '../assets/logo.png';
import ProfileModal from './ProfileModal';

const NavigationBar = () => {
  const { user, logout } = useContext(AuthContext);
  const navigate = useNavigate();
  const [showProfileModal, setShowProfileModal] = useState(false);

  const handleLogout = () => {
    logout();
    navigate('/login');
  };

  return (
    <>
      <Navbar bg="light" expand="lg">
        <Container>
          <LinkContainer to="/">
            <Navbar.Brand className="d-flex align-items-center gap-2">
              <img
                src={logo}
                alt="LM Ltd Logo"
                width="80"
                height="80"
                className="rounded-circle"
              />
              LM Ltd
            </Navbar.Brand>
          </LinkContainer>

          <Navbar.Toggle aria-controls="main-navbar" />
          <Navbar.Collapse id="main-navbar">
            <Nav className="ms-auto">
              <Nav.Link as={NavLink} to="/">Home</Nav.Link>
              <Nav.Link as={NavLink} to="/services">Services</Nav.Link>

              {!user ? (
```

```
            <>
              <Nav.Link as={NavLink} to="/login">Login</Nav.Link>
              <Nav.Link as={NavLink} to="/register">Register</Nav.Link>
              <Nav.Link as={NavLink} to="/admin-login">Admin</Nav.Link>
            </>
          ) : (
            <>
              <Navbar.Text className="me-2">Hello, {user.name}</Navbar.Text>

              {user.role === 'admin' && (
                <Nav.Link as={NavLink} to="/admin">Dashboard</Nav.Link>
              )}
              {user.role === 'user' && (
                <Nav.Link as={NavLink} to="/dashboard">Dashboard</Nav.Link>
              )}

              <Nav.Link onClick={() => setShowProfileModal(true)} className="text-primary">
                Profile
              </Nav.Link>
              <Nav.Link onClick={handleLogout} className="text-danger">
                Logout
              </Nav.Link>
            </>
          )}
        </Nav>
      </Navbar.Collapse>
    </Container>
  </Navbar>

  {/* Profile Modal */}
  {user && (
    <ProfileModal
      show={showProfileModal}
      onHide={() => setShowProfileModal(false)}
    />
  )}
 </>
 );
};

export default NavigationBar;
```

PrivateRoute.jsx

```
import React, { useContext } from "react";
import { Navigate } from "react-router-dom";
import { AuthContext } from "../contexts/AuthContext";

function PrivateRoute({ children, adminOnly = false }) {
  const { user } = useContext(AuthContext);

  if (!user) return <Navigate to="/login" />;
  if (adminOnly && user.role !== "admin") return <Navigate to="/" />;

  return children;
}

export default PrivateRoute;
```

UserDashboard.jsx

```
import React, { useEffect, useState, useContext } from "react";
import axios from "axios";
import { AuthContext } from "../contexts/AuthContext";
import { Container, Table, Spinner, Alert, Badge } from "react-bootstrap";

function UserDashboard() {
  const { user } = useContext(AuthContext);
  const [requests, setRequests] = useState([]);
  const [scheduledRequests, setScheduledRequests] = useState([]);
  const [pastRequests, setPastRequests] = useState([]);
  const [schedules, setSchedules] = useState([]);
  const [scheduledServices, setScheduledServices] = useState([]);
  const [pastServices, setPastServices] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState("");

  useEffect(() => {
    const fetchData = async () => {
      try {
        const [reqRes, schedRes] = await Promise.all([
          axios.get("/api/user/requests", {
            headers: { Authorization: `Bearer ${user?.token}` },
          }),
          axios.get("/api/user/schedules", {
            headers: { Authorization: `Bearer ${user?.token}` },
          }),
```

```
      ]);

      setRequests(reqRes.data.requests || []);
      setScheduledRequests(reqRes.data.scheduled || []);
      setPastRequests(reqRes.data.past || []);
      setSchedules(schedRes.data.schedules || []);
      setScheduledServices(schedRes.data.scheduled || []);
      setPastServices(schedRes.data.past || []);
    } catch (err) {
      console.error("Dashboard error:", err);
      setError("Failed to load your service data.");
    } finally {
      setLoading(false);
    }
  };

  if (user?.token) fetchData();
}, [user]);

const renderStatusBadge = (status) => {
  const variantMap = {
    pending: "warning",
    confirmed: "primary",
    completed: "success",
    cancelled: "danger",
    expired: "secondary",
  };
  const variant = variantMap[status?.toLowerCase()] || "dark";
  return <Badge bg={variant} className="text-capitalize">{status}</Badge>;
};

const renderTable = (title, data, dateField = "createdAt") => (
  <>
    <h5 className="mt-4 mb-3">{title}</h5>
    {data.length === 0 ? (
      <Alert variant="info">No {title.toLowerCase()} available.</Alert>
    ) : (
      <Table striped bordered hover responsive>
        <thead>
          <tr>
            <th>ID</th>
            <th>Service</th>
            <th>Status</th>
            <th>{dateField === "scheduledFor" ? "Scheduled Date" : "Requested On"}</th>
```

```
          </tr>
        </thead>
        <tbody>
          {data.map((item, index) => (
            <tr key={item._id}>
              <td>{index + 1}</td>
              <td>{item.serviceName || item.serviceTitle}</td>
              <td>{renderStatusBadge(item.status)}</td>
              <td>{new Date(item[dateField]).toLocaleDateString()}</td>
            </tr>
          ))}
        </tbody>
      </Table>
    )}
  </>
);

return (
  <>
    <Container style={{ padding: "2rem" }}>
      <h2 className="mb-4 text-center">Welcome, {user?.name}</h2>
      <p>Email: {user?.email}</p>
      <p>Role: {user?.role}</p>
      <hr />
      <h4 className="mb-3 text-center">Your Service Overview</h4>

      {loading ? (
        <Spinner animation="border" />
      ) : error ? (
        <Alert variant="danger">{error}</Alert>
      ) : (
        <>
          {renderTable("Requested Services", requests)}
          {renderTable("Scheduled Requests", scheduledRequests, "scheduledFor")}
          {renderTable("Past Requests", pastRequests, "scheduledFor")}
          {renderTable("Scheduled Services", scheduledServices, "scheduledFor")}
          {renderTable("Past Services", pastServices, "scheduledFor")}
        </>
      )}
      <hr />
    </Container>
    <footer className="text-center py-2">
      <small>&copy; {new Date().getFullYear()} LM Ltd. All rights reserved.</small>
    </footer>
```

```
    </>
  );
}

export default UserDashboard;
```

App.jsx

```
import React from "react";
import { Routes, Route } from "react-router-dom";
import NavigationBar from "./components/NavigationBar";
import ServicesPromo from "./components/ServicesPromo";
import UserDashboard from "./pages/UserDashboard";
import AdminDashboard from "./pages/AdminDashboard";
import AdminRoute from "./components/AdminRoute";
import PrivateRoute from "./components/PrivateRoute";
import AdminLogin from "./pages/AdminLogin";
import LearnMore from "./pages/LearnMore";
import WhoWeAre from "./pages/WhoWeAre";
import Contact from "./pages/Contact";
import Home from "./pages/Home";
import Services from "./pages/Services";
import Register from "./pages/Register";
import Profile from "./pages/Profile";
import Login from "./pages/Login";

function App() {
  return (
    <>
      <NavigationBar />
      <ServicesPromo />
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/services" element={<Services />} />
        <Route path="/register" element={<Register />} />
        <Route path="/login" element={<Login />} />
        <Route path="/profile" element={<Profile />} />
        <Route path="/learn-more" element={<LearnMore />} />
        <Route path="/who-we-are" element={<WhoWeAre />} />
        <Route path="/contact" element={<Contact />} />
        <Route path="/admin-login" element={<AdminLogin />} />
        {/* Protected User Dashboard */}
        <Route
          path="/dashboard"
```

```
          element={
            <PrivateRoute>
              <UserDashboard />
            </PrivateRoute>
          }
        />
        <Route
          path="/admin"
          element={
            <AdminRoute>
              <AdminDashboard />
            </AdminRoute>
          }
        />
      </Routes>
    </>
  );
}

export default App;
```

authMiddlewere.js

```
const jwt = require('jsonwebtoken');
const User = require('../models/User');

const protect = async (req, res, next) => {
  const auth = req.headers.authorization;
  if (!auth || !auth.startsWith('Bearer ')) {
    return res.status(401).json({ error: 'Not authorized.' });
  }

  try {
    const token = auth.split(' ')[1];
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    req.user = await User.findById(decoded.id).select('-password');
    next();
  } catch (err) {
    console.error('Auth error:', err);
    res.status(401).json({ error: 'Token failed.' });
  }
};

module.exports = { protect };
```

model/ServiceRequest.js

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

const ServiceRequestSchema = new mongoose.Schema({
  serviceName:      { type: String, required: true }, // unified naming
  fullName:         { type: String, required: true },
  serviceType:      { type: String, required: true },
  details:          { type: String, required: true },
  paid:             { type: Boolean, default: false },
  stripeSessionId:  { type: String },
  scheduledFor:     { type: Date }, // optional if scheduling is added later
  status:           { type: String, default: 'pending' },
  user:             { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
  createdAt:        { type: Date, default: Date.now }
});

module.exports = mongoose.model('ServiceRequest', ServiceRequestSchema);
```

model/ServiceSchedule.js

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

const ServiceScheduleSchema = new mongoose.Schema({
  serviceName:   { type: String, required: true }, // unified naming
  fullName:      { type: String, required: true },
  serviceType:   { type: String, required: true },
  date:          { type: String }, // optional display
  time:          { type: String }, // optional display
  scheduledFor:  { type: Date },   // canonical scheduling field
  status:        { type: String, default: 'pending' },
  user:          { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
  createdAt:     { type: Date, default: Date.now }
});

module.exports = mongoose.model('ServiceSchedule', ServiceScheduleSchema);
```

routes/requests.js

```
const express = require('express');
const router = express.Router();
const ServiceRequest = require('../models/ServiceRequest');
const { protect } = require('../middleware/authMiddleware');
```

```
// Create a new service request
router.post('/', protect, async (req, res) => {
  try {
    const { serviceTitle, fullName, serviceType, details } = req.body;

    const request = await ServiceRequest.create({
      serviceTitle,
      fullName,
      serviceType,
      details,
      user: req.user._id, // ✅ associate with user
      paid: false
    });

    res.status(201).json(request);
  } catch (err) {
    console.error('POST /api/requests error:', err);
    res.status(500).json({ error: 'Failed to create service request.' });
  }
});

// List all requests for the logged-in user
router.get('/', protect, async (req, res) => {
  try {
    const list = await ServiceRequest.find({ user: req.user._id }).sort('-createdAt');
    res.json(list);
  } catch (err) {
    console.error('GET /api/requests error:', err);
    res.status(500).json({ error: 'Failed to fetch requests.' });
  }
});

router.get('/:id', async (req, res) => {
  try {
    const reqDoc = await ServiceRequest.findById(req.params.id);
    if (!reqDoc) return res.status(404).json({ error: 'Request not found.' });
    res.json(reqDoc);
  } catch (err) {
    console.error(`GET /api/requests/${req.params.id} error:`, err);
    res.status(500).json({ error: 'Failed to fetch request.' });
  }
});

router.patch('/:id', async (req, res) => {
```

```
  try {
    const updated = await ServiceRequest.findByIdAndUpdate(
      req.params.id,
      req.body,
      { new: true, runValidators: true }
    );
    if (!updated) return res.status(404).json({ error: 'Request not found.' });
    res.json(updated);
  } catch (err) {
    console.error(`PATCH /api/requests/${req.params.id} error:`, err);
    res.status(500).json({ error: 'Failed to update request.' });
  }
});

router.delete('/:id', async (req, res) => {
  try {
    const deleted = await ServiceRequest.findByIdAndDelete(req.params.id);
    if (!deleted) return res.status(404).json({ error: 'Request not found.' });
    res.json({ deleted: true });
  } catch (err) {
    console.error(`DELETE /api/requests/${req.params.id} error:`, err);
    res.status(500).json({ error: 'Failed to delete request.' });
  }
});

module.exports = router;
```

router/schedule.js

```
const express = require("express");
const router = express.Router();
const ServiceSchedule = require("../models/ServiceSchedule");
const { protect } = require("../middleware/authMiddleware");

// Create a new schedule
router.post("/", protect, async (req, res) => {
  try {
    const sched = await ServiceSchedule.create({
      ...req.body,
      user: req.user._id,
    });
    res.json(sched);
  } catch (err) {
    console.error("POST /api/schedules error:", err);
```

```
    res.status(500).json({ error: "Failed to create schedule." });
  }
});

// List all schedules for the logged-in user
router.get("/", protect, async (req, res) => {
  try {
    const list = await ServiceSchedule.find({ user: req.user._id }).sort("-createdAt");
    res.json(list);
  } catch (err) {
    console.error("GET /api/schedules error:", err);
    res.status(500).json({ error: "Failed to fetch schedules." });
  }
});

// Update
router.patch("/:id", protect, async (req, res) => {
  const updated = await ServiceSchedule.findByIdAndUpdate(
    req.params.id,
    req.body,
    { new: true }
  );
  res.json(updated);
});

// Delete
router.delete("/:id", protect, async (req, res) => {
  await ServiceSchedule.findByIdAndDelete(req.params.id);
  res.json({ deleted: true });
});

module.exports = router;
```

router/userRoutes.js

```
const express = require('express');
const router = express.Router();
const { signup, login, logout } = require('../controllers/authController');
const { updateProfile } = require('../controllers/userController');
const { protect } = require('../middleware/authMiddleware');
const ServiceRequest = require('../models/ServiceRequest');
const ServiceSchedule = require('../models/ServiceSchedule');

// Auth routes
```

```
router.post('/signup', signup);
router.post('/login', login);
router.post('/logout', logout);
router.put('/profile', protect, updateProfile);

// GET /api/user/requests
router.get('/requests', protect, async (req, res) => {
  try {
    const allRequests = await ServiceRequest.find({ user: req.user._id });
    const now = new Date();
    const scheduled = allRequests.filter(r => r.scheduledFor && new Date(r.scheduledFor) >
now);
    const past = allRequests.filter(r => r.scheduledFor && new Date(r.scheduledFor) <= now);
    res.json({ requests: allRequests, scheduled, past });
  } catch (err) {
    console.error("Error fetching user requests:", err);
    res.status(500).json({ message: "Server error" });
  }
});

// GET /api/user/schedules
router.get('/schedules', protect, async (req, res) => {
  try {
    const allSchedules = await ServiceSchedule.find({ user: req.user._id });
    const now = new Date();
    const scheduled = allSchedules.filter(r => r.scheduledFor && new Date(r.scheduledFor) >
now);
    const past = allSchedules.filter(r => r.scheduledFor && new Date(r.scheduledFor) <= now);
    res.json({ schedules: allSchedules, scheduled, past });
  } catch (err) {
    console.error("Error fetching user schedules:", err);
    res.status(500).json({ message: "Server error" });
  }
});

router.get('/ping', (req, res) => {
  res.send('User route is working');
});

module.exports = router;
```

server.js

```
require('dotenv').config();
```

```
const express = require('express');
const mongoose = require('mongoose');
const bodyParser = require('body-parser');
const cors = require('cors');
const path = require('path');

// Import routes
const authRoutes = require('./routes/authRoutes');
const serviceRoutes = require('./routes/serviceRoutes');
const userRoutes = require('./routes/userRoutes');
const requestRoutes = require('./routes/requests');
const scheduleRoutes = require('./routes/schedules');
const shareRoutes = require('./routes/shares');
const testimonialsRoute = require('./routes/testimonials');
const cardsRoutes = require('./routes/cardsRoutes');
const adminRoutes = require('./routes/adminRoutes');

// Instantiate the Express application
const app = express();

// Stripe webhook route must be mounted BEFORE body parsers
app.use('/api/requests/webhook', express.raw({ type: 'application/json' }), requestRoutes);

// Middlewares
app.use(cors({ origin: 'http://localhost:3000', credentials: true }));
app.use(express.json());
app.use(bodyParser.json());

// Mount API routes
app.use('/api/auth', authRoutes);
app.use('/api/user', userRoutes);
app.use('/api/cards', cardsRoutes);
app.use('/api/admin', adminRoutes);
app.use('/api/services', serviceRoutes);
app.use('/api/requests', requestRoutes); // regular request routes
app.use('/api/schedules', scheduleRoutes);
app.use('/api/shares', shareRoutes);
app.use('/api/testimonials', testimonialsRoute);

// Connect to MongoDB and start server
mongoose.connect(process.env.MONGO_URI, {
  useNewUrlParser: true,
})
.then(() => {
```

```
  console.log('✅ MongoDB connected');
  app.listen(process.env.PORT, () =>
    console.log(`🚀 Server listening on port ${process.env.PORT}`)
  );
})
.catch(err => {
  console.error('❌ MongoDB connection error:', err);
});
```