# Project Code Compilation

📌 Total Code Files: 100

## 1. Eslint.Config

📂 File Path: C:\Users\Admin\React\sai\project(hemo\_connect)\eslint.config.js

import js from '@eslint/js';  
import globals from 'globals';  
import reactHooks from 'eslint-plugin-react-hooks';  
import reactRefresh from 'eslint-plugin-react-refresh';  
  
export default [  
 { ignores: ['dist'] },  
 {  
 extends: [js.configs.recommended],  
 files: ['\*\*/\*.{js,jsx}'],  
 languageOptions: {  
 ecmaVersion: 2020,  
 globals: globals.browser,  
 },  
 plugins: {  
 'react-hooks': reactHooks,  
 'react-refresh': reactRefresh,  
 },  
 rules: {  
 ...reactHooks.configs.recommended.rules,  
 'react-refresh/only-export-components': [  
 'warn',  
 { allowConstantExport: true },  
 ],  
 },  
 },  
];

================================================================================

## 2. Index

📂 File Path: C:\Users\Admin\React\sai\project(hemo\_connect)\index.html

<!doctype html>  
<html lang="en">  
 <head>  
 <meta charset="UTF-8" />  
 <link rel="icon" type="image/svg+xml" href="/vite.svg" />  
 <meta name="viewport" content="width=device-width, initial-scale=1.0" />  
 <title>HEMO CONNECT - Blood Donation Platform</title>  
 </head>  
 <body>  
 <div id="root"></div>  
 <script type="module" src="/src/main.jsx"></script>  
 </body>  
</html>

================================================================================

## 3. Postcss.Config

📂 File Path: C:\Users\Admin\React\sai\project(hemo\_connect)\postcss.config.js

export default {  
 plugins: {  
 tailwindcss: {},  
 autoprefixer: {},  
 },  
};

================================================================================

## 4. Tailwind.Config

📂 File Path: C:\Users\Admin\React\sai\project(hemo\_connect)\tailwind.config.js

/\*\* @type {import('tailwindcss').Config} \*/  
export default {  
 content: ['./index.html', './src/\*\*/\*.{js,ts,jsx,tsx}'],  
 theme: {  
 extend: {},  
 },  
 plugins: [],  
};

================================================================================

## 5. Vite.Config

📂 File Path: C:\Users\Admin\React\sai\project(hemo\_connect)\vite.config.js

import { defineConfig } from 'vite';  
import react from '@vitejs/plugin-react';  
  
export default defineConfig({  
 plugins: [react()],  
 server: {  
 proxy: {  
 '/api': {  
 target: 'http://localhost:5000',  
 changeOrigin: true,  
 secure: false,  
 },  
 },  
 },  
 build: {  
 chunkSizeWarningLimit: 1000, // Increase limit to suppress warning (optional)  
 rollupOptions: {  
 output: {  
 manualChunks(id) {  
 if (id.includes('node\_modules')) {  
 return id  
 .toString()  
 .split('node\_modules/')[1]  
 .split('/')[0]  
 .toString();  
 }  
 },  
 },  
 },  
 },  
});

================================================================================

## 6. Index

📂 File Path: C:\Users\Admin\React\sai\project(hemo\_connect)\server\index.js

import express from 'express';  
import cors from 'cors';  
import http from 'http';  
import { Server } from 'socket.io';  
import mongoose from 'mongoose';  
import dotenv from 'dotenv';  
import authRoutes from './routes/auth.js';  
import userRoutes from './routes/users.js';  
import bloodRequestRoutes from './routes/blood-requests.js';  
import donorBloodRoutes from './routes/donor-blood.js';  
import adminRoutes from './routes/admin.js';  
import contactRoutes from './routes/contact.js';  
import collectionRoutes from './routes/collections.js';  
import dynamicModelRoutes from './routes/dynamic-models.js';  
import { authenticateSocket } from './middleware/auth.js';  
  
const app = express();  
const server = http.createServer(app);  
const io = new Server(server, {  
 cors: {  
 origin: "\*",  
 methods: ["GET", "POST"]  
 }  
});  
  
// Middleware  
app.use(cors());  
app.use(express.json());  
  
// Routes  
app.use('/api/auth', authRoutes);  
app.use('/api/users', userRoutes);  
app.use('/api/blood-requests', bloodRequestRoutes);  
app.use('/api/donor-blood', donorBloodRoutes);  
app.use('/api/admin', adminRoutes);  
app.use('/api/contact', contactRoutes);  
app.use('/api/collections', collectionRoutes);  
app.use('/api/dynamic-models', dynamicModelRoutes);  
import notificationRoutes from './routes/notifications.js';  
app.use('/api/notifications', notificationRoutes);  
  
// Health check  
app.get('/api/health', (req, res) => {  
 res.json({ status: 'OK', timestamp: new Date().toISOString() });  
});  
  
// MongoDB Connection  
const connectDB = async () => {  
 try {  
 await mongoose.connect('mongodb://localhost:27017/sai\_new');  
 console.log('Database connected successfully');  
 await ensureCollections();  
 } catch (error) {  
 console.error('Database connection failed:', error);  
 process.exit(1);  
 }  
};  
  
// Ensure required collections exist using the collection manager  
const ensureCollections = async () => {  
 const { createMultipleCollections } = await import('./utils/collectionManager.js');  
 const required = [  
 'users',   
 'bloodrequests',   
 'notifications',   
 'contacts',  
 'contactmessages',  
 'donorblood'  
 ];  
   
 try {  
 const results = await createMultipleCollections(required);  
 const successCount = Object.values(results).filter(r => r.success).length;  
 console.log(`✅ Ensured ${successCount}/${required.length} required collections exist`);  
 } catch (error) {  
 console.warn('Could not ensure all required collections:', error?.message || error);  
 }  
};  
  
// Socket.IO Connection  
io.use(authenticateSocket);  
io.on('connection', (socket) => {  
 console.log('User connected:', socket.userId);  
   
 // Join user to their personal room for notifications  
 socket.join(`user\_${socket.userId}`);  
   
 socket.on('disconnect', () => {  
 console.log('User disconnected:', socket.userId);  
 });  
});  
  
// Error handling middleware  
app.use((error, req, res, next) => {  
 console.error('Server Error:', error);  
 res.status(500).json({  
 message: 'Internal server error',  
 error: process.env.NODE\_ENV === 'development' ? error.message : undefined  
 });  
});  
  
// 404 handler  
app.use('\*', (req, res) => {  
 res.status(404).json({ message: 'Route not found' });  
});  
  
const PORT = process.env.PORT || 5000;  
  
// Start server  
connectDB().then(() => {  
 server.listen(PORT, () => {  
 console.log(`Server running on port ${PORT}`);  
 });  
});  
  
// Make io available to routes  
app.set('io', io);  
  
export default app;

================================================================================

## 7. Auth

📂 File Path: C:\Users\Admin\React\sai\project(hemo\_connect)\server\middleware\auth.js

import jwt from 'jsonwebtoken';  
import User from '../models/User.js';  
  
const JWT\_SECRET = process.env.JWT\_SECRET || 'your-secret-key';  
  
const authenticateToken = async (req, res, next) => {  
 const authHeader = req.headers['authorization'];  
 const token = authHeader && authHeader.split(' ')[1];  
 if (!token) {  
 console.log('No token provided');  
 return res.status(401).json({ message: 'Access token required' });  
 }  
 jwt.verify(token, JWT\_SECRET, async (err, decoded) => {  
 if (err) {  
 console.log('JWT verification error:', err);  
 return res.status(403).json({ message: 'Invalid or expired token' });  
 }  
 try {  
 console.log('Decoded JWT:', decoded);  
 let userId = decoded.userId;  
 if (typeof userId !== 'string') userId = String(userId);  
 // Validate ObjectId format  
 if (!userId.match(/^[a-fA-F0-9]{24}$/)) {  
 console.error('Invalid userId in JWT:', userId);  
 return res.status(400).json({ message: 'Invalid user ID in token' });  
 }  
 const user = await User.findById(userId);  
 console.log('Authenticated user lookup for:', userId, 'Result:', user);  
 if (!user) {  
 return res.status(404).json({ message: 'User not found', userId });  
 }  
 req.user = user;  
 next();  
 } catch (error) {  
 console.log('Error during user lookup:', error);  
 return res.status(500).json({ message: 'Server error' });  
 }  
 });  
};  
  
const authenticateSocket = async (socket, next) => {  
 const token = socket.handshake.auth.token;  
 if (!token) {  
 return next(new Error('Authentication error'));  
 }  
 jwt.verify(token, JWT\_SECRET, async (err, decoded) => {  
 if (err) {  
 return next(new Error('Authentication error'));  
 }  
 try {  
 const user = await User.findById(decoded.userId);  
 if (!user) {  
 return next(new Error('User not found'));  
 }  
 socket.userId = user.\_id;  
 socket.user = user;  
 next();  
 } catch (error) {  
 return next(new Error('Server error'));  
 }  
 });  
};  
  
const requireRole = (role) => {  
 return (req, res, next) => {  
 if (req.user.role !== role) {  
 return res.status(403).json({   
 message: `Access denied. ${role} role required.`   
 });  
 }  
 next();  
 };  
};  
  
const requireRoles = (roles) => {  
 return (req, res, next) => {  
 if (!roles.includes(req.user.role)) {  
 return res.status(403).json({   
 message: `Access denied. One of these roles required: ${roles.join(', ')}`   
 });  
 }  
 next();  
 };  
};  
  
export {  
 authenticateToken,  
 authenticateSocket,  
 requireRole,  
 requireRoles,  
 JWT\_SECRET,  
};

================================================================================

## 8. Bloodrequest

📂 File Path: C:\Users\Admin\React\sai\project(hemo\_connect)\server\models\BloodRequest.js

import mongoose from 'mongoose';  
import { ensureCollection } from '../utils/collectionManager.js';  
  
const donorResponseSchema = new mongoose.Schema({  
 donorId: {   
 type: mongoose.Schema.Types.ObjectId,   
 ref: 'User',   
 required: true   
 },  
 donorName: { type: String, required: true },  
 donorPhone: { type: String },  
 donorEmail: { type: String },  
 responseType: {   
 type: String,   
 enum: ['accept', 'decline', 'maybe'],   
 required: true   
 },  
 responseMessage: { type: String, maxlength: 500 },  
 availabilityDate: { type: Date },  
 location: { type: String },  
 timestamp: { type: Date, default: Date.now },  
 isConfirmed: { type: Boolean, default: false }  
});  
  
const bloodRequestSchema = new mongoose.Schema({  
 // Requester information  
 requesterId: {   
 type: mongoose.Schema.Types.ObjectId,   
 ref: 'User',   
 required: true   
 },  
 requesterName: { type: String, required: true },  
 requesterPhone: { type: String, required: true },  
 requesterEmail: { type: String },  
   
 // Blood requirement details  
 bloodGroup: {   
 type: String,   
 required: true,  
 enum: ['A+', 'A-', 'B+', 'B-', 'AB+', 'AB-', 'O+', 'O-']  
 },  
 units: {   
 type: Number,   
 required: true,  
 min: 1,  
 max: 10  
 },  
 urgencyLevel: {   
 type: String,   
 required: true,  
 enum: ['low', 'medium', 'high', 'critical'],  
 default: 'medium'  
 },  
   
 // Medical information  
 hospital: { type: String, required: true },  
 hospitalAddress: { type: String },  
 doctorName: { type: String },  
 doctorContact: { type: String },  
 medicalCondition: { type: String },  
 prescriptionRequired: { type: Boolean, default: false },  
   
 // Location and timing  
 location: { type: String, required: true },  
 requiredByDate: { type: Date, required: true },  
 preferredTimeSlots: [{  
 date: Date,  
 startTime: String,  
 endTime: String  
 }],  
   
 // Additional information  
 additionalNotes: {   
 type: String,   
 maxlength: 1000   
 },  
 specialRequirements: { type: String },  
   
 // Status and tracking  
 status: {   
 type: String,   
 enum: ['pending', 'matched', 'confirmed', 'in\_progress', 'fulfilled', 'cancelled', 'expired'],  
 default: 'pending'  
 },  
 priority: {  
 type: String,  
 enum: ['low', 'medium', 'high', 'urgent'],  
 default: 'medium'  
 },  
   
 // Donor responses  
 responses: [donorResponseSchema],  
 matchedDonors: [{  
 donorId: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },  
 confirmedAt: { type: Date },  
 donationDate: { type: Date },  
 status: {   
 type: String,   
 enum: ['matched', 'confirmed', 'donated', 'cancelled']   
 }  
 }],  
   
 // Administrative  
 isVerified: { type: Boolean, default: false },  
 verifiedBy: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },  
 verifiedAt: { type: Date },  
 isFlagged: { type: Boolean, default: false },  
 flagReason: { type: String },  
 flaggedBy: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },  
 flaggedAt: { type: Date },  
   
 // Analytics  
 viewCount: { type: Number, default: 0 },  
 responseCount: { type: Number, default: 0 },  
 shareCount: { type: Number, default: 0 },  
   
 // Timestamps  
 createdAt: { type: Date, default: Date.now },  
 updatedAt: { type: Date, default: Date.now },  
 expiresAt: { type: Date }  
}, {  
 timestamps: true,  
 collection: 'bloodrequests'  
});  
  
// Indexes for better query performance  
bloodRequestSchema.index({ requesterId: 1 });  
bloodRequestSchema.index({ bloodGroup: 1 });  
bloodRequestSchema.index({ location: 1 });  
bloodRequestSchema.index({ status: 1 });  
bloodRequestSchema.index({ urgencyLevel: 1 });  
bloodRequestSchema.index({ requiredByDate: 1 });  
bloodRequestSchema.index({ createdAt: -1 });  
bloodRequestSchema.index({ bloodGroup: 1, location: 1, status: 1 });  
bloodRequestSchema.index({ expiresAt: 1 }, { expireAfterSeconds: 0 });  
  
// Pre-save middleware  
bloodRequestSchema.pre('save', function(next) {  
 this.updatedAt = new Date();  
   
 // Set expiration date if not set (default 30 days)  
 if (!this.expiresAt) {  
 this.expiresAt = new Date(Date.now() + 30 \* 24 \* 60 \* 60 \* 1000);  
 }  
   
 // Update response count  
 this.responseCount = this.responses.length;  
   
 next();  
});  
  
// Static methods for querying  
bloodRequestSchema.statics.findActive = function() {  
 return this.find({   
 status: { $in: ['pending', 'matched', 'confirmed'] },  
 expiresAt: { $gt: new Date() }  
 });  
};  
  
bloodRequestSchema.statics.findByBloodGroup = function(bloodGroup) {  
 return this.find({ bloodGroup, status: 'pending' });  
};  
  
bloodRequestSchema.statics.findByLocation = function(location) {  
 return this.find({   
 location: { $regex: location, $options: 'i' },  
 status: 'pending'  
 });  
};  
  
bloodRequestSchema.statics.findUrgent = function() {  
 return this.find({   
 urgencyLevel: { $in: ['high', 'critical'] },  
 status: { $in: ['pending', 'matched'] }  
 });  
};  
  
bloodRequestSchema.statics.findExpired = function() {  
 return this.find({   
 expiresAt: { $lt: new Date() },  
 status: { $in: ['pending', 'matched'] }  
 });  
};  
  
// Instance methods  
bloodRequestSchema.methods.addResponse = function(donorId, responseType, responseMessage = '') {  
 const response = {  
 donorId,  
 responseType,  
 responseMessage,  
 timestamp: new Date()  
 };  
   
 this.responses.push(response);  
   
 if (responseType === 'accept') {  
 this.status = 'matched';  
 }  
   
 return this.save();  
};  
  
bloodRequestSchema.methods.confirmDonor = function(donorId, donationDate) {  
 const matchedDonor = this.matchedDonors.find(d => d.donorId.toString() === donorId.toString());  
 if (matchedDonor) {  
 matchedDonor.status = 'confirmed';  
 matchedDonor.confirmedAt = new Date();  
 matchedDonor.donationDate = donationDate;  
 this.status = 'confirmed';  
 }  
 return this.save();  
};  
  
bloodRequestSchema.methods.markAsFulfilled = function() {  
 this.status = 'fulfilled';  
 return this.save();  
};  
  
bloodRequestSchema.methods.cancel = function(reason = '') {  
 this.status = 'cancelled';  
 this.additionalNotes = this.additionalNotes + `\nCancelled: ${reason}`;  
 return this.save();  
};  
  
// Ensure collection exists before creating model  
const initBloodRequestModel = async () => {  
 await ensureCollection('bloodrequests');  
 return mongoose.model('BloodRequest', bloodRequestSchema);  
};  
  
// Create model immediately for backward compatibility  
const BloodRequest = mongoose.model('BloodRequest', bloodRequestSchema);  
  
export default BloodRequest;  
export { initBloodRequestModel };

================================================================================

## 9. Contact

📂 File Path: C:\Users\Admin\React\sai\project(hemo\_connect)\server\models\Contact.js

import mongoose from 'mongoose';  
  
const contactSchema = new mongoose.Schema({  
 name: { type: String, required: true },  
 email: { type: String, required: true },  
 subject: { type: String, required: true },  
 message: { type: String, required: true },  
 timestamp: { type: Date, default: Date.now },  
});  
  
const Contact = mongoose.model('Contact', contactSchema);  
export default Contact;

================================================================================

## 10. Contactmessage

📂 File Path: C:\Users\Admin\React\sai\project(hemo\_connect)\server\models\ContactMessage.js

import mongoose from 'mongoose';  
import { ensureCollection } from '../utils/collectionManager.js';  
  
const contactMessageSchema = new mongoose.Schema({  
 // Basic contact information  
 name: {   
 type: String,   
 required: true,  
 trim: true,  
 maxlength: 100  
 },  
 email: {   
 type: String,   
 required: true,  
 trim: true,  
 lowercase: true,  
 match: [/^\w+([.-]?\w+)\*@\w+([.-]?\w+)\*(\.\w{2,3})+$/, 'Please enter a valid email']  
 },  
 phone: {  
 type: String,  
 trim: true,  
 match: [/^[\+]?[1-9][\d]{0,15}$/, 'Please enter a valid phone number']  
 },  
   
 // Message details  
 subject: {   
 type: String,   
 required: true,  
 trim: true,  
 maxlength: 200  
 },  
 message: {   
 type: String,   
 required: true,  
 trim: true,  
 maxlength: 2000  
 },  
   
 // Message categorization  
 category: {  
 type: String,  
 enum: ['general', 'support', 'feedback', 'complaint', 'suggestion', 'partnership', 'media'],  
 default: 'general'  
 },  
 priority: {  
 type: String,  
 enum: ['low', 'medium', 'high', 'urgent'],  
 default: 'medium'  
 },  
   
 // Status tracking  
 status: {  
 type: String,  
 enum: ['new', 'in\_progress', 'responded', 'resolved', 'closed'],  
 default: 'new'  
 },  
   
 // Response tracking  
 response: {  
 message: String,  
 respondedBy: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },  
 respondedAt: Date  
 },  
   
 // Additional metadata  
 source: {  
 type: String,  
 enum: ['website', 'mobile\_app', 'api', 'admin\_panel'],  
 default: 'website'  
 },  
 userAgent: String,  
 ipAddress: String,  
   
 // Timestamps  
 createdAt: { type: Date, default: Date.now },  
 updatedAt: { type: Date, default: Date.now }  
}, {  
 timestamps: true,  
 collection: 'contactmessages'  
});  
  
// Indexes for better query performance  
contactMessageSchema.index({ email: 1 });  
contactMessageSchema.index({ status: 1 });  
contactMessageSchema.index({ category: 1 });  
contactMessageSchema.index({ priority: 1 });  
contactMessageSchema.index({ createdAt: -1 });  
contactMessageSchema.index({ status: 1, priority: 1 });  
  
// Pre-save middleware  
contactMessageSchema.pre('save', function(next) {  
 this.updatedAt = new Date();  
 next();  
});  
  
// Static methods for querying  
contactMessageSchema.statics.findByStatus = function(status) {  
 return this.find({ status });  
};  
  
contactMessageSchema.statics.findByCategory = function(category) {  
 return this.find({ category });  
};  
  
contactMessageSchema.statics.findByPriority = function(priority) {  
 return this.find({ priority });  
};  
  
contactMessageSchema.statics.findUnresolved = function() {  
 return this.find({ status: { $in: ['new', 'in\_progress'] } });  
};  
  
contactMessageSchema.statics.findByDateRange = function(startDate, endDate) {  
 return this.find({  
 createdAt: {  
 $gte: startDate,  
 $lte: endDate  
 }  
 });  
};  
  
// Instance methods  
contactMessageSchema.methods.markAsResponded = function(responseMessage, respondedBy) {  
 this.status = 'responded';  
 this.response = {  
 message: responseMessage,  
 respondedBy: respondedBy,  
 respondedAt: new Date()  
 };  
 return this.save();  
};  
  
contactMessageSchema.methods.markAsResolved = function() {  
 this.status = 'resolved';  
 return this.save();  
};  
  
// Ensure collection exists before creating model  
const initContactMessageModel = async () => {  
 await ensureCollection('contactmessages');  
 return mongoose.model('ContactMessage', contactMessageSchema);  
};  
  
// Create model immediately for backward compatibility  
const ContactMessage = mongoose.model('ContactMessage', contactMessageSchema);  
  
export default ContactMessage;  
export { initContactMessageModel };

================================================================================

## 11. Donorblood

📂 File Path: C:\Users\Admin\React\sai\project(hemo\_connect)\server\models\DonorBlood.js

import mongoose from 'mongoose';  
import { ensureCollection } from '../utils/collectionManager.js';  
  
const donationRecordSchema = new mongoose.Schema({  
 donationId: { type: String, required: true, unique: true },  
 donationDate: { type: Date, required: true },  
 bloodBank: { type: String, required: true },  
 bloodBankAddress: { type: String },  
 unitsDonated: { type: Number, required: true, min: 1, max: 2 },  
 bloodType: {   
 type: String,   
 required: true,  
 enum: ['A+', 'A-', 'B+', 'B-', 'AB+', 'AB-', 'O+', 'O-']  
 },  
 hemoglobinLevel: { type: Number },  
 bloodPressure: { type: String },  
 weight: { type: Number },  
 temperature: { type: Number },  
 medicalOfficer: { type: String },  
 certificateNumber: { type: String },  
 nextEligibleDate: { type: Date },  
 status: {  
 type: String,  
 enum: ['completed', 'rejected', 'deferred'],  
 default: 'completed'  
 },  
 rejectionReason: { type: String },  
 notes: { type: String }  
});  
  
const donorBloodSchema = new mongoose.Schema({  
 // Donor information  
 donorId: {   
 type: mongoose.Schema.Types.ObjectId,   
 ref: 'User',   
 required: true,  
 unique: true  
 },  
 donorName: { type: String, required: true },  
 donorEmail: { type: String, required: true },  
 donorPhone: { type: String, required: true },  
   
 // Blood information  
 bloodGroup: {   
 type: String,   
 required: true,  
 enum: ['A+', 'A-', 'B+', 'B-', 'AB+', 'AB-', 'O+', 'O-']  
 },  
 rhFactor: {   
 type: String,   
 enum: ['positive', 'negative'],  
 required: true  
 },  
   
 // Donation eligibility  
 isEligible: { type: Boolean, default: true },  
 eligibilityReason: { type: String },  
 lastDonationDate: { type: Date },  
 nextEligibleDate: { type: Date },  
 totalDonations: { type: Number, default: 0 },  
 totalUnitsDonated: { type: Number, default: 0 },  
   
 // Medical information  
 medicalHistory: {  
 hasDiabetes: { type: Boolean, default: false },  
 hasHypertension: { type: Boolean, default: false },  
 hasHeartDisease: { type: Boolean, default: false },  
 hasHepatitis: { type: Boolean, default: false },  
 hasHIV: { type: Boolean, default: false },  
 hasMalaria: { type: Boolean, default: false },  
 hasTuberculosis: { type: Boolean, default: false },  
 hasCancer: { type: Boolean, default: false },  
 hasEpilepsy: { type: Boolean, default: false },  
 hasBleedingDisorder: { type: Boolean, default: false },  
 isPregnant: { type: Boolean, default: false },  
 isBreastfeeding: { type: Boolean, default: false },  
 recentSurgery: { type: Boolean, default: false },  
 recentTattoo: { type: Boolean, default: false },  
 recentTravel: { type: Boolean, default: false },  
 medications: [{   
 name: String,   
 dosage: String,   
 startDate: Date,  
 endDate: Date  
 }],  
 allergies: [String],  
 otherConditions: { type: String }  
 },  
   
 // Physical information  
 physicalInfo: {  
 age: { type: Number, min: 18, max: 65 },  
 weight: { type: Number, min: 45, max: 200 },  
 height: { type: Number, min: 140, max: 220 },  
 gender: {   
 type: String,   
 enum: ['male', 'female', 'other'],  
 required: true  
 }  
 },  
   
 // Availability and preferences  
 availability: {  
 isAvailable: { type: Boolean, default: true },  
 preferredLocations: [String],  
 preferredTimeSlots: [{  
 day: { type: String, enum: ['monday', 'tuesday', 'wednesday', 'thursday', 'friday', 'saturday', 'sunday'] },  
 startTime: String,  
 endTime: String  
 }],  
 maxDistance: { type: Number, default: 50 }, // in kilometers  
 emergencyOnly: { type: Boolean, default: false }  
 },  
   
 // Donation records  
 donationHistory: [donationRecordSchema],  
   
 // Verification and certification  
 isVerified: { type: Boolean, default: false },  
 verifiedBy: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },  
 verifiedAt: { type: Date },  
 certificationNumber: { type: String },  
 certificationExpiry: { type: Date },  
   
 // Statistics  
 stats: {  
 totalRequests: { type: Number, default: 0 },  
 acceptedRequests: { type: Number, default: 0 },  
 declinedRequests: { type: Number, default: 0 },  
 responseRate: { type: Number, default: 0 },  
 averageResponseTime: { type: Number, default: 0 }, // in minutes  
 lastActiveDate: { type: Date, default: Date.now }  
 },  
   
 // Status and flags  
 status: {  
 type: String,  
 enum: ['active', 'inactive', 'suspended', 'banned'],  
 default: 'active'  
 },  
 isFlagged: { type: Boolean, default: false },  
 flagReason: { type: String },  
 flaggedBy: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },  
 flaggedAt: { type: Date },  
   
 // Timestamps  
 createdAt: { type: Date, default: Date.now },  
 updatedAt: { type: Date, default: Date.now }  
}, {  
 timestamps: true,  
 collection: 'donorblood'  
});  
  
// Indexes for better query performance  
// Note: donorId index is automatically created due to unique: true  
donorBloodSchema.index({ bloodGroup: 1 });  
donorBloodSchema.index({ isEligible: 1 });  
donorBloodSchema.index({ 'availability.isAvailable': 1 });  
donorBloodSchema.index({ nextEligibleDate: 1 });  
donorBloodSchema.index({ status: 1 });  
donorBloodSchema.index({ 'availability.preferredLocations': 1 });  
donorBloodSchema.index({ bloodGroup: 1, 'availability.isAvailable': 1, isEligible: 1 });  
  
// Pre-save middleware  
donorBloodSchema.pre('save', function(next) {  
 this.updatedAt = new Date();  
   
 // Calculate next eligible date (56 days from last donation)  
 if (this.lastDonationDate) {  
 const nextDate = new Date(this.lastDonationDate);  
 nextDate.setDate(nextDate.getDate() + 56);  
 this.nextEligibleDate = nextDate;  
   
 // Check if eligible based on last donation  
 this.isEligible = new Date() >= nextDate;  
 }  
   
 // Update total donations count  
 this.totalDonations = this.donationHistory.length;  
 this.totalUnitsDonated = this.donationHistory.reduce((total, donation) => {  
 return total + (donation.status === 'completed' ? donation.unitsDonated : 0);  
 }, 0);  
   
 // Calculate response rate  
 if (this.stats.totalRequests > 0) {  
 this.stats.responseRate = (this.stats.acceptedRequests / this.stats.totalRequests) \* 100;  
 }  
   
 next();  
});  
  
// Static methods for querying  
donorBloodSchema.statics.findEligibleDonors = function(bloodGroup, location = null) {  
 const query = {  
 bloodGroup,  
 isEligible: true,  
 'availability.isAvailable': true,  
 status: 'active',  
 $or: [  
 { nextEligibleDate: { $exists: false } },  
 { nextEligibleDate: { $lte: new Date() } }  
 ]  
 };  
   
 if (location) {  
 query['availability.preferredLocations'] = { $regex: location, $options: 'i' };  
 }  
   
 return this.find(query);  
};  
  
donorBloodSchema.statics.findByBloodGroup = function(bloodGroup) {  
 return this.find({ bloodGroup, status: 'active' });  
};  
  
donorBloodSchema.statics.findAvailableDonors = function() {  
 return this.find({  
 'availability.isAvailable': true,  
 isEligible: true,  
 status: 'active'  
 });  
};  
  
donorBloodSchema.statics.findEmergencyDonors = function(bloodGroup) {  
 return this.find({  
 bloodGroup,  
 'availability.emergencyOnly': true,  
 'availability.isAvailable': true,  
 isEligible: true,  
 status: 'active'  
 });  
};  
  
// Instance methods  
donorBloodSchema.methods.addDonation = function(donationData) {  
 const donation = {  
 donationId: `DON${Date.now()}${Math.random().toString(36).substr(2, 5).toUpperCase()}`,  
 donationDate: new Date(),  
 ...donationData  
 };  
   
 this.donationHistory.push(donation);  
 this.lastDonationDate = donation.donationDate;  
 this.totalDonations += 1;  
   
 if (donation.status === 'completed') {  
 this.totalUnitsDonated += donation.unitsDonated;  
 }  
   
 return this.save();  
};  
  
donorBloodSchema.methods.updateAvailability = function(isAvailable, reason = '') {  
 this.availability.isAvailable = isAvailable;  
 if (reason) {  
 this.availability.notes = reason;  
 }  
 return this.save();  
};  
  
donorBloodSchema.methods.updateStats = function(requestType) {  
 this.stats.totalRequests += 1;  
 this.stats.lastActiveDate = new Date();  
   
 if (requestType === 'accepted') {  
 this.stats.acceptedRequests += 1;  
 } else if (requestType === 'declined') {  
 this.stats.declinedRequests += 1;  
 }  
   
 return this.save();  
};  
  
donorBloodSchema.methods.suspend = function(reason) {  
 this.status = 'suspended';  
 this.flagReason = reason;  
 this.flaggedAt = new Date();  
 return this.save();  
};  
  
donorBloodSchema.methods.activate = function() {  
 this.status = 'active';  
 this.isFlagged = false;  
 this.flagReason = '';  
 return this.save();  
};  
  
// Ensure collection exists before creating model  
const initDonorBloodModel = async () => {  
 await ensureCollection('donorblood');  
 return mongoose.model('DonorBlood', donorBloodSchema);  
};  
  
// Create model immediately for backward compatibility  
const DonorBlood = mongoose.model('DonorBlood', donorBloodSchema);  
  
export default DonorBlood;  
export { initDonorBloodModel };

================================================================================

## 12. Notification

📂 File Path: C:\Users\Admin\React\sai\project(hemo\_connect)\server\models\Notification.js

import mongoose from 'mongoose';  
  
const notificationSchema = new mongoose.Schema({  
 userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },  
 type: { type: String, required: true },  
 title: { type: String },  
 message: { type: String, required: true },  
 isRead: { type: Boolean, default: false },  
 timestamp: { type: Date, default: Date.now },  
});  
  
const Notification = mongoose.model('Notification', notificationSchema);  
export default Notification;

================================================================================

## 13. User

📂 File Path: C:\Users\Admin\React\sai\project(hemo\_connect)\server\models\User.js

import mongoose from 'mongoose';  
import { ensureCollection } from '../utils/collectionManager.js';  
  
const userSchema = new mongoose.Schema({  
 name: { type: String, required: true },  
 email: { type: String, required: true, unique: true },  
 password: { type: String, required: true },  
 bloodGroup: { type: String, required: true },  
 location: { type: String, required: true },  
 role: { type: String, enum: ['donor', 'recipient', 'admin'], default: 'donor' },  
 profileComplete: { type: Boolean, default: false },  
 isAvailable: { type: Boolean, default: true },  
 phone: { type: String },  
 address: { type: String },  
 emergencyContact: { type: String },  
 createdAt: { type: Date, default: Date.now },  
 updatedAt: { type: Date, default: Date.now },  
 profilePic: { type: String },  
 // Donor specific fields  
 medicalConditions: { type: String },  
 medications: { type: String },  
 lastDonationDate: { type: Date },  
 lastActive: { type: Date },  
 // Recipient specific fields  
 hospital: { type: String },  
 doctorName: { type: String },  
 doctorContact: { type: String },  
 urgencyLevel: { type: String, default: 'medium' },  
});  
  
// Ensure collection exists before creating model  
const initUserModel = async () => {  
 await ensureCollection('users');  
 return mongoose.model('User', userSchema);  
};  
  
// Create model immediately for backward compatibility  
const User = mongoose.model('User', userSchema);  
  
export default User;  
export { initUserModel };

================================================================================

## 14. Admin

📂 File Path: C:\Users\Admin\React\sai\project(hemo\_connect)\server\routes\admin.js

import express from 'express';  
import User from '../models/User.js';  
import BloodRequest from '../models/BloodRequest.js';  
import { authenticateToken, requireRole } from '../middleware/auth.js';  
  
const router = express.Router();  
  
// Get admin statistics  
router.get('/stats', authenticateToken, requireRole('admin'), async (req, res) => {  
 try {  
 const totalUsers = await User.countDocuments();  
 const activeRequests = await BloodRequest.countDocuments({ status: { $in: ['pending', 'matched'] } });  
 const pendingReports = await BloodRequest.countDocuments({ isFlagged: true });  
 // completedDonations: This should be a separate collection in a real app  
 const stats = {  
 totalUsers,  
 activeRequests,  
 completedDonations: 45, // Mock data  
 pendingReports,  
 };  
 res.json(stats);  
 } catch (error) {  
 console.error('Get admin stats error:', error);  
 res.status(500).json({ message: 'Failed to fetch statistics' });  
 }  
});  
  
// Get recent users  
router.get('/recent-users', authenticateToken, requireRole('admin'), async (req, res) => {  
 try {  
 const recentUsers = await User.find().sort({ createdAt: -1 }).limit(10);  
 res.json(recentUsers);  
 } catch (error) {  
 console.error('Get recent users error:', error);  
 res.status(500).json({ message: 'Failed to fetch recent users' });  
 }  
});  
  
// Get flagged requests  
router.get('/flagged-requests', authenticateToken, requireRole('admin'), async (req, res) => {  
 try {  
 const flaggedRequests = await BloodRequest.find({ isFlagged: true });  
 res.json(flaggedRequests);  
 } catch (error) {  
 console.error('Get flagged requests error:', error);  
 res.status(500).json({ message: 'Failed to fetch flagged requests' });  
 }  
});  
  
// Approve request  
router.post('/requests/:id/approve', authenticateToken, requireRole('admin'), async (req, res) => {  
 try {  
 const requestId = req.params.id;  
 const updatedRequest = await BloodRequest.findByIdAndUpdate(  
 requestId,  
 { isFlagged: false, flagReason: null, updatedAt: new Date() },  
 { new: true }  
 );  
 if (!updatedRequest) {  
 return res.status(404).json({ message: 'Request not found' });  
 }  
 res.json({  
 message: 'Request approved successfully',  
 request: updatedRequest,  
 });  
 } catch (error) {  
 console.error('Approve request error:', error);  
 res.status(500).json({ message: 'Failed to approve request' });  
 }  
});  
  
// Reject request  
router.post('/requests/:id/reject', authenticateToken, requireRole('admin'), async (req, res) => {  
 try {  
 const requestId = req.params.id;  
 const updatedRequest = await BloodRequest.findByIdAndUpdate(  
 requestId,  
 { status: 'cancelled', flagReason: 'Rejected by admin', updatedAt: new Date() },  
 { new: true }  
 );  
 if (!updatedRequest) {  
 return res.status(404).json({ message: 'Request not found' });  
 }  
 // Notify the requester  
 const io = req.app.get('io');  
 const notificationData = {  
 userId: updatedRequest.userId,  
 type: 'request\_rejected',  
 title: 'Request Rejected',  
 message: 'Your blood request has been rejected by our moderation team',  
 timestamp: new Date(),  
 };  
 io.to(`user\_${updatedRequest.userId}`).emit('notification', notificationData);  
 // Save notification to DB  
 try {  
 const Notification = (await import('../models/Notification.js')).default;  
 await Notification.create(notificationData);  
 } catch (err) {  
 console.error('Failed to save notification:', err);  
 }  
 res.json({  
 message: 'Request rejected successfully',  
 request: updatedRequest,  
 });  
 } catch (error) {  
 console.error('Reject request error:', error);  
 res.status(500).json({ message: 'Failed to reject request' });  
 }  
});  
  
export default router;

================================================================================

## 15. Auth

📂 File Path: C:\Users\Admin\React\sai\project(hemo\_connect)\server\routes\auth.js

import express from 'express';  
import jwt from 'jsonwebtoken';  
import bcrypt from 'bcryptjs';  
import User from '../models/User.js';  
import { authenticateToken, JWT\_SECRET } from '../middleware/auth.js';  
  
const router = express.Router();  
  
// In-memory OTP store (for demo only)  
const otpStore = {};  
  
// Register  
router.post('/signup', async (req, res) => {  
 try {  
 const { name, email, password, bloodGroup, location, role } = req.body;  
 // Check if user already exists  
 const existingUser = await User.findOne({ email });  
 if (existingUser) {  
 return res.status(400).json({ message: 'User already exists with this email' });  
 }  
 // Hash password  
 const hashedPassword = await bcrypt.hash(password, 12);  
 // Create user  
 const userData = {  
 name,  
 email,  
 password: hashedPassword,  
 bloodGroup,  
 location,  
 role: role || 'donor',  
 };  
 const user = await User.create(userData);  
 // Generate JWT token  
 const token = jwt.sign(  
 { userId: user.\_id, email: user.email },  
 JWT\_SECRET,  
 { expiresIn: '24h' }  
 );  
 res.status(201).json({  
 message: 'User created successfully',  
 token,  
 user,  
 });  
 } catch (error) {  
 console.error('Signup error:', error);  
 res.status(500).json({ message: 'Failed to create user' });  
 }  
});  
  
// Login  
router.post('/login', async (req, res) => {  
 try {  
 const { email, password, rememberMe } = req.body;  
 // Find user  
 const user = await User.findOne({ email });  
 if (!user) {  
 return res.status(400).json({ message: 'Invalid email or password' });  
 }  
 // Check password  
 const isPasswordValid = await bcrypt.compare(password, user.password);  
 if (!isPasswordValid) {  
 return res.status(400).json({ message: 'Invalid email or password' });  
 }  
 // Generate JWT token  
 const expiresIn = rememberMe ? '7d' : '24h';  
 const token = jwt.sign(  
 { userId: user.\_id, email: user.email },  
 JWT\_SECRET,  
 { expiresIn }  
 );  
 // Update last login  
 user.lastLogin = new Date();  
 await user.save();  
 res.json({  
 message: 'Login successful',  
 token,  
 user,  
 });  
 } catch (error) {  
 console.error('Login error:', error);  
 res.status(500).json({ message: 'Failed to login' });  
 }  
});  
  
// Verify token  
router.get('/verify', authenticateToken, async (req, res) => {  
 const user = await User.findById(req.user.\_id);  
 if (!user) {  
 return res.status(404).json({ message: 'User not found' });  
 }  
 res.json(user);  
});  
  
// Forgot password  
router.post('/forgot-password', async (req, res) => {  
 try {  
 const { email } = req.body;  
 const user = await User.findOne({ email });  
 if (!user) {  
 return res.status(404).json({ message: 'User not found' });  
 }  
 // In a real app, send password reset email  
 res.json({ message: 'Password reset instructions sent to your email' });  
 } catch (error) {  
 console.error('Forgot password error:', error);  
 res.status(500).json({ message: 'Failed to process request' });  
 }  
});  
  
// Send OTP for password reset  
router.post('/send-otp', async (req, res) => {  
 try {  
 const { method, value } = req.body; // method: 'email' or 'phone', value: email or phone  
 let user;  
 if (method === 'email') {  
 user = await User.findOne({ email: value });  
 } else if (method === 'phone') {  
 user = await User.findOne({ phone: value });  
 } else {  
 return res.status(400).json({ message: 'Invalid method' });  
 }  
 if (!user) {  
 return res.status(404).json({ message: 'User not found' });  
 }  
 // Generate OTP  
 const otp = Math.floor(100000 + Math.random() \* 900000).toString();  
 otpStore[value] = { otp, expires: Date.now() + 5 \* 60 \* 1000 }; // 5 min expiry  
 // In a real app, send OTP via email/SMS  
 console.log(`OTP for ${value}: ${otp}`);  
 res.json({ message: 'OTP sent', otp }); // For demo, return OTP in response  
 } catch (error) {  
 console.error('Send OTP error:', error);  
 res.status(500).json({ message: 'Failed to send OTP' });  
 }  
});  
  
// Reset password using OTP  
router.post('/reset-password', async (req, res) => {  
 try {  
 const { method, value, otp, newPassword } = req.body;  
 let user;  
 if (method === 'email') {  
 user = await User.findOne({ email: value });  
 } else if (method === 'phone') {  
 user = await User.findOne({ phone: value });  
 } else {  
 return res.status(400).json({ message: 'Invalid method' });  
 }  
 if (!user) {  
 return res.status(404).json({ message: 'User not found' });  
 }  
 const stored = otpStore[value];  
 if (!stored || stored.otp !== otp || stored.expires < Date.now()) {  
 return res.status(400).json({ message: 'Invalid or expired OTP' });  
 }  
 user.password = await bcrypt.hash(newPassword, 12);  
 await user.save();  
 delete otpStore[value];  
 res.json({ message: 'Password reset successful' });  
 } catch (error) {  
 console.error('Reset password error:', error);  
 res.status(500).json({ message: 'Failed to reset password' });  
 }  
});  
  
export default router;

================================================================================

## 16. Blood Requests

📂 File Path: C:\Users\Admin\React\sai\project(hemo\_connect)\server\routes\blood-requests.js

import express from 'express';  
import BloodRequest from '../models/BloodRequest.js';  
import { authenticateToken } from '../middleware/auth.js';  
  
const router = express.Router();  
  
// Create blood request  
router.post('/', authenticateToken, async (req, res) => {  
 try {  
 const requestData = {  
 ...req.body,  
 requesterId: req.user.\_id,  
 requesterName: req.user.name,  
 requesterEmail: req.user.email,  
 requesterPhone: req.user.phone || req.body.requesterPhone  
 };  
   
 const newRequest = await BloodRequest.create(requestData);  
 console.log('Blood request created:', newRequest.\_id);  
   
 res.status(201).json({  
 success: true,  
 message: 'Blood request created successfully',  
 request: newRequest  
 });  
 } catch (error) {  
 console.error('Create blood request error:', error);  
 res.status(500).json({   
 success: false,  
 message: 'Failed to create blood request',  
 error: error.message   
 });  
 }  
});  
  
// Get active blood requests (for ticker)  
router.get('/active', async (req, res) => {  
 try {  
 const activeRequests = await BloodRequest.findActive()  
 .sort({ urgencyLevel: -1, createdAt: -1 })  
 .limit(10)  
 .populate('requesterId', 'name location');  
   
 res.json({  
 success: true,  
 requests: activeRequests  
 });  
 } catch (error) {  
 console.error('Get active requests error:', error);  
 res.status(500).json({   
 success: false,  
 message: 'Failed to fetch active requests',  
 error: error.message   
 });  
 }  
});  
  
// Get nearby requests for donor  
router.get('/nearby', authenticateToken, async (req, res) => {  
 try {  
 if (req.user.role !== 'donor') {  
 return res.status(403).json({   
 success: false,  
 message: 'Only donors can view nearby requests'   
 });  
 }  
   
 const { location, bloodGroup } = req.query;  
 const searchLocation = location || req.user.location;  
 const searchBloodGroup = bloodGroup || req.user.bloodGroup;  
   
 const nearbyRequests = await BloodRequest.find({  
 status: 'pending',  
 location: { $regex: searchLocation, $options: 'i' },  
 bloodGroup: searchBloodGroup,  
 expiresAt: { $gt: new Date() }  
 }).populate('requesterId', 'name phone');  
   
 res.json({  
 success: true,  
 requests: nearbyRequests  
 });  
 } catch (error) {  
 console.error('Get nearby requests error:', error);  
 res.status(500).json({   
 success: false,  
 message: 'Failed to fetch nearby requests',  
 error: error.message   
 });  
 }  
});  
  
// Get user's own requests  
router.get('/my-requests', authenticateToken, async (req, res) => {  
 try {  
 const { status } = req.query;  
 const query = { requesterId: req.user.\_id };  
   
 if (status) {  
 query.status = status;  
 }  
   
 const userRequests = await BloodRequest.find(query)  
 .sort({ createdAt: -1 })  
 .populate('responses.donorId', 'name phone email');  
   
 res.json({  
 success: true,  
 requests: userRequests  
 });  
 } catch (error) {  
 console.error('Get my requests error:', error);  
 res.status(500).json({   
 success: false,  
 message: 'Failed to fetch your requests',  
 error: error.message   
 });  
 }  
});  
  
// Get user's request history  
router.get('/my-history', authenticateToken, async (req, res) => {  
 try {  
 const { page = 1, limit = 10 } = req.query;  
   
 const userHistory = await BloodRequest.find({ requesterId: req.user.\_id })  
 .sort({ createdAt: -1 })  
 .limit(limit \* 1)  
 .skip((page - 1) \* limit)  
 .populate('responses.donorId', 'name phone email')  
 .populate('matchedDonors.donorId', 'name phone email');  
   
 const total = await BloodRequest.countDocuments({ requesterId: req.user.\_id });  
   
 res.json({  
 success: true,  
 requests: userHistory,  
 pagination: {  
 current: page,  
 pages: Math.ceil(total / limit),  
 total  
 }  
 });  
 } catch (error) {  
 console.error('Get request history error:', error);  
 res.status(500).json({   
 success: false,  
 message: 'Failed to fetch request history',  
 error: error.message   
 });  
 }  
});  
  
// Get blood request by ID  
router.get('/:id', authenticateToken, async (req, res) => {  
 try {  
 const request = await BloodRequest.findById(req.params.id)  
 .populate('requesterId', 'name email phone')  
 .populate('responses.donorId', 'name phone email')  
 .populate('matchedDonors.donorId', 'name phone email');  
   
 if (!request) {  
 return res.status(404).json({   
 success: false,  
 message: 'Blood request not found'   
 });  
 }  
   
 // Check if user can view this request  
 if (request.requesterId.\_id.toString() !== req.user.\_id.toString() &&   
 req.user.role !== 'admin' &&   
 req.user.role !== 'donor') {  
 return res.status(403).json({   
 success: false,  
 message: 'Not authorized to view this request'   
 });  
 }  
   
 res.json({  
 success: true,  
 request  
 });  
 } catch (error) {  
 console.error('Get blood request error:', error);  
 res.status(500).json({   
 success: false,  
 message: 'Failed to fetch blood request',  
 error: error.message   
 });  
 }  
});  
  
// Respond to blood request (donor)  
router.post('/:id/respond', authenticateToken, async (req, res) => {  
 try {  
 if (req.user.role !== 'donor') {  
 return res.status(403).json({   
 success: false,  
 message: 'Only donors can respond to requests'   
 });  
 }  
   
 const { responseType, responseMessage, availabilityDate, location } = req.body;  
 const requestId = req.params.id;  
   
 const request = await BloodRequest.findById(requestId);  
 if (!request) {  
 return res.status(404).json({   
 success: false,  
 message: 'Blood request not found'   
 });  
 }  
   
 // Check if donor already responded  
 const existingResponse = request.responses.find(  
 r => r.donorId.toString() === req.user.\_id.toString()  
 );  
   
 if (existingResponse) {  
 return res.status(400).json({   
 success: false,  
 message: 'You have already responded to this request'   
 });  
 }  
   
 // Add response  
 await request.addResponse(req.user.\_id, responseType, responseMessage);  
   
 // Update donor info in response  
 const response = request.responses[request.responses.length - 1];  
 response.donorName = req.user.name;  
 response.donorPhone = req.user.phone;  
 response.donorEmail = req.user.email;  
 response.availabilityDate = availabilityDate;  
 response.location = location;  
   
 await request.save();  
   
 res.json({  
 success: true,  
 message: `Response recorded: ${responseType}`,  
 request: request  
 });  
 } catch (error) {  
 console.error('Respond to request error:', error);  
 res.status(500).json({   
 success: false,  
 message: 'Failed to respond to request',  
 error: error.message   
 });  
 }  
});  
  
// Confirm donor for blood request  
router.post('/:id/confirm-donor', authenticateToken, async (req, res) => {  
 try {  
 const { donorId, donationDate } = req.body;  
 const requestId = req.params.id;  
   
 const request = await BloodRequest.findById(requestId);  
 if (!request) {  
 return res.status(404).json({   
 success: false,  
 message: 'Blood request not found'   
 });  
 }  
   
 // Check if user can confirm donors for this request  
 if (request.requesterId.toString() !== req.user.\_id.toString() &&   
 req.user.role !== 'admin') {  
 return res.status(403).json({   
 success: false,  
 message: 'Not authorized to confirm donors for this request'   
 });  
 }  
   
 await request.confirmDonor(donorId, donationDate);  
   
 res.json({  
 success: true,  
 message: 'Donor confirmed successfully',  
 request: request  
 });  
 } catch (error) {  
 console.error('Confirm donor error:', error);  
 res.status(500).json({   
 success: false,  
 message: 'Failed to confirm donor',  
 error: error.message   
 });  
 }  
});  
  
// Cancel blood request  
router.post('/:id/cancel', authenticateToken, async (req, res) => {  
 try {  
 const { reason } = req.body;  
 const requestId = req.params.id;  
   
 const request = await BloodRequest.findById(requestId);  
 if (!request) {  
 return res.status(404).json({   
 success: false,  
 message: 'Blood request not found'   
 });  
 }  
   
 if (request.requesterId.toString() !== req.user.\_id.toString() &&   
 req.user.role !== 'admin') {  
 return res.status(403).json({   
 success: false,  
 message: 'Not authorized to cancel this request'   
 });  
 }  
   
 await request.cancel(reason);  
   
 res.json({  
 success: true,  
 message: 'Request cancelled successfully',  
 request: request  
 });  
 } catch (error) {  
 console.error('Cancel request error:', error);  
 res.status(500).json({   
 success: false,  
 message: 'Failed to cancel request',  
 error: error.message   
 });  
 }  
});  
  
// Mark request as fulfilled  
router.post('/:id/fulfill', authenticateToken, async (req, res) => {  
 try {  
 const requestId = req.params.id;  
   
 const request = await BloodRequest.findById(requestId);  
 if (!request) {  
 return res.status(404).json({   
 success: false,  
 message: 'Blood request not found'   
 });  
 }  
   
 if (request.requesterId.toString() !== req.user.\_id.toString() &&   
 req.user.role !== 'admin') {  
 return res.status(403).json({   
 success: false,  
 message: 'Not authorized to fulfill this request'   
 });  
 }  
   
 await request.markAsFulfilled();  
   
 res.json({  
 success: true,  
 message: 'Request marked as fulfilled',  
 request: request  
 });  
 } catch (error) {  
 console.error('Fulfill request error:', error);  
 res.status(500).json({   
 success: false,  
 message: 'Failed to fulfill request',  
 error: error.message   
 });  
 }  
});  
  
export default router;

================================================================================

## 17. Collections

📂 File Path: C:\Users\Admin\React\sai\project(hemo\_connect)\server\routes\collections.js

import express from 'express';  
import {   
 createCollection,   
 createMultipleCollections,   
 listCollections,   
 dropCollection,   
 validateCollectionName,  
 ensureCollection   
} from '../utils/collectionManager.js';  
import { authenticateToken, requireRole } from '../middleware/auth.js';  
  
const router = express.Router();  
  
/\*\*  
 \* GET /api/collections  
 \* List all collections in the database  
 \*/  
router.get('/', authenticateToken, requireRole('admin'), async (req, res) => {  
 try {  
 const collections = await listCollections();  
 res.json({  
 success: true,  
 collections,  
 count: collections.length  
 });  
 } catch (error) {  
 console.error('List collections error:', error);  
 res.status(500).json({  
 success: false,  
 message: 'Failed to list collections',  
 error: error.message  
 });  
 }  
});  
  
/\*\*  
 \* POST /api/collections  
 \* Create a new collection  
 \* Body: { name: string, options?: object }  
 \*/  
router.post('/', authenticateToken, requireRole('admin'), async (req, res) => {  
 try {  
 const { name, options = {} } = req.body;  
  
 if (!name) {  
 return res.status(400).json({  
 success: false,  
 message: 'Collection name is required'  
 });  
 }  
  
 const result = await createCollection(name, options);  
   
 if (result.success) {  
 res.status(201).json(result);  
 } else {  
 res.status(400).json(result);  
 }  
 } catch (error) {  
 console.error('Create collection error:', error);  
 res.status(500).json({  
 success: false,  
 message: 'Failed to create collection',  
 error: error.message  
 });  
 }  
});  
  
/\*\*  
 \* POST /api/collections/batch  
 \* Create multiple collections at once  
 \* Body: { names: string[], options?: object }  
 \*/  
router.post('/batch', authenticateToken, requireRole('admin'), async (req, res) => {  
 try {  
 const { names, options = {} } = req.body;  
  
 if (!names || !Array.isArray(names) || names.length === 0) {  
 return res.status(400).json({  
 success: false,  
 message: 'Array of collection names is required'  
 });  
 }  
  
 // Validate all names first  
 const validationErrors = [];  
 for (const name of names) {  
 const validation = validateCollectionName(name);  
 if (!validation.isValid) {  
 validationErrors.push({ name, error: validation.error });  
 }  
 }  
  
 if (validationErrors.length > 0) {  
 return res.status(400).json({  
 success: false,  
 message: 'Some collection names are invalid',  
 errors: validationErrors  
 });  
 }  
  
 const results = await createMultipleCollections(names, options);  
 const successCount = Object.values(results).filter(r => r.success).length;  
   
 res.status(201).json({  
 success: true,  
 message: `Created ${successCount}/${names.length} collections`,  
 results  
 });  
 } catch (error) {  
 console.error('Batch create collections error:', error);  
 res.status(500).json({  
 success: false,  
 message: 'Failed to create collections',  
 error: error.message  
 });  
 }  
});  
  
/\*\*  
 \* POST /api/collections/ensure  
 \* Ensure a collection exists (create if it doesn't)  
 \* Body: { name: string, options?: object }  
 \*/  
router.post('/ensure', authenticateToken, async (req, res) => {  
 try {  
 const { name, options = {} } = req.body;  
  
 if (!name) {  
 return res.status(400).json({  
 success: false,  
 message: 'Collection name is required'  
 });  
 }  
  
 const result = await createCollection(name, options);  
 res.json(result);  
 } catch (error) {  
 console.error('Ensure collection error:', error);  
 res.status(500).json({  
 success: false,  
 message: 'Failed to ensure collection exists',  
 error: error.message  
 });  
 }  
});  
  
/\*\*  
 \* DELETE /api/collections/:name  
 \* Drop a collection (admin only)  
 \*/  
router.delete('/:name', authenticateToken, requireRole('admin'), async (req, res) => {  
 try {  
 const { name } = req.params;  
  
 if (!name) {  
 return res.status(400).json({  
 success: false,  
 message: 'Collection name is required'  
 });  
 }  
  
 const result = await dropCollection(name);  
   
 if (result.success) {  
 res.json(result);  
 } else {  
 res.status(400).json(result);  
 }  
 } catch (error) {  
 console.error('Drop collection error:', error);  
 res.status(500).json({  
 success: false,  
 message: 'Failed to drop collection',  
 error: error.message  
 });  
 }  
});  
  
/\*\*  
 \* POST /api/collections/validate  
 \* Validate a collection name  
 \* Body: { name: string }  
 \*/  
router.post('/validate', async (req, res) => {  
 try {  
 const { name } = req.body;  
  
 if (!name) {  
 return res.status(400).json({  
 success: false,  
 message: 'Collection name is required'  
 });  
 }  
  
 const validation = validateCollectionName(name);  
 res.json({  
 success: true,  
 isValid: validation.isValid,  
 error: validation.error || null  
 });  
 } catch (error) {  
 console.error('Validate collection name error:', error);  
 res.status(500).json({  
 success: false,  
 message: 'Failed to validate collection name',  
 error: error.message  
 });  
 }  
});  
  
export default router;

================================================================================

## 18. Contact

📂 File Path: C:\Users\Admin\React\sai\project(hemo\_connect)\server\routes\contact.js

import express from 'express';  
import ContactMessage from '../models/ContactMessage.js';  
  
const router = express.Router();  
  
// Handle contact form submission  
router.post('/', async (req, res) => {  
 try {  
 const { name, email, phone, subject, message, category, priority } = req.body;  
   
 // Validate required fields  
 if (!name || !email || !subject || !message) {  
 return res.status(400).json({   
 message: 'Name, email, subject, and message are required'   
 });  
 }  
   
 // Create contact message  
 const contactMessage = new ContactMessage({  
 name,  
 email,  
 phone,  
 subject,  
 message,  
 category: category || 'general',  
 priority: priority || 'medium',  
 source: 'website',  
 userAgent: req.get('User-Agent'),  
 ipAddress: req.ip || req.connection.remoteAddress  
 });  
   
 const savedMessage = await contactMessage.save();  
 console.log('Contact message saved:', savedMessage.\_id);  
   
 res.status(201).json({   
 success: true,  
 message: 'Thank you for your message. We will get back to you soon!',  
 id: savedMessage.\_id,  
 category: savedMessage.category,  
 priority: savedMessage.priority  
 });  
 } catch (error) {  
 console.error('Contact form error:', error);  
 res.status(500).json({   
 success: false,  
 message: 'Failed to send message',  
 error: error.message   
 });  
 }  
});  
  
// Get all contact messages (admin only)  
router.get('/', async (req, res) => {  
 try {  
 const { status, category, priority, page = 1, limit = 10 } = req.query;  
   
 const query = {};  
 if (status) query.status = status;  
 if (category) query.category = category;  
 if (priority) query.priority = priority;  
   
 const messages = await ContactMessage.find(query)  
 .sort({ createdAt: -1 })  
 .limit(limit \* 1)  
 .skip((page - 1) \* limit)  
 .populate('response.respondedBy', 'name email');  
   
 const total = await ContactMessage.countDocuments(query);  
   
 res.json({  
 success: true,  
 messages,  
 pagination: {  
 current: page,  
 pages: Math.ceil(total / limit),  
 total  
 }  
 });  
 } catch (error) {  
 console.error('Get contact messages error:', error);  
 res.status(500).json({   
 success: false,  
 message: 'Failed to fetch contact messages',  
 error: error.message   
 });  
 }  
});  
  
// Get contact message by ID  
router.get('/:id', async (req, res) => {  
 try {  
 const message = await ContactMessage.findById(req.params.id)  
 .populate('response.respondedBy', 'name email');  
   
 if (!message) {  
 return res.status(404).json({   
 success: false,  
 message: 'Contact message not found'   
 });  
 }  
   
 res.json({  
 success: true,  
 message  
 });  
 } catch (error) {  
 console.error('Get contact message error:', error);  
 res.status(500).json({   
 success: false,  
 message: 'Failed to fetch contact message',  
 error: error.message   
 });  
 }  
});  
  
// Update contact message status  
router.patch('/:id/status', async (req, res) => {  
 try {  
 const { status, responseMessage } = req.body;  
   
 const message = await ContactMessage.findById(req.params.id);  
 if (!message) {  
 return res.status(404).json({   
 success: false,  
 message: 'Contact message not found'   
 });  
 }  
   
 if (status === 'responded' && responseMessage) {  
 await message.markAsResponded(responseMessage, req.user?.\_id);  
 } else if (status === 'resolved') {  
 await message.markAsResolved();  
 } else {  
 message.status = status;  
 await message.save();  
 }  
   
 res.json({  
 success: true,  
 message: 'Contact message updated successfully',  
 updatedMessage: message  
 });  
 } catch (error) {  
 console.error('Update contact message error:', error);  
 res.status(500).json({   
 success: false,  
 message: 'Failed to update contact message',  
 error: error.message   
 });  
 }  
});  
  
export default router;

================================================================================

## 19. Donor Blood

📂 File Path: C:\Users\Admin\React\sai\project(hemo\_connect)\server\routes\donor-blood.js

import express from 'express';  
import DonorBlood from '../models/DonorBlood.js';  
import { authenticateToken } from '../middleware/auth.js';  
  
const router = express.Router();  
  
// Create or update donor blood profile  
router.post('/', authenticateToken, async (req, res) => {  
 try {  
 if (req.user.role !== 'donor') {  
 return res.status(403).json({   
 success: false,  
 message: 'Only donors can create blood profiles'   
 });  
 }  
   
 const donorBloodData = {  
 ...req.body,  
 donorId: req.user.\_id,  
 donorName: req.user.name,  
 donorEmail: req.user.email,  
 donorPhone: req.user.phone  
 };  
   
 // Check if profile already exists  
 let donorBlood = await DonorBlood.findOne({ donorId: req.user.\_id });  
   
 if (donorBlood) {  
 // Update existing profile  
 Object.assign(donorBlood, donorBloodData);  
 await donorBlood.save();  
 } else {  
 // Create new profile  
 donorBlood = await DonorBlood.create(donorBloodData);  
 }  
   
 console.log('Donor blood profile saved:', donorBlood.\_id);  
   
 res.status(201).json({  
 success: true,  
 message: 'Donor blood profile saved successfully',  
 profile: donorBlood  
 });  
 } catch (error) {  
 console.error('Save donor blood profile error:', error);  
 res.status(500).json({   
 success: false,  
 message: 'Failed to save donor blood profile',  
 error: error.message   
 });  
 }  
});  
  
// Get donor's own blood profile  
router.get('/my-profile', authenticateToken, async (req, res) => {  
 try {  
 if (req.user.role !== 'donor') {  
 return res.status(403).json({   
 success: false,  
 message: 'Only donors can view blood profiles'   
 });  
 }  
   
 const profile = await DonorBlood.findOne({ donorId: req.user.\_id });  
   
 if (!profile) {  
 return res.status(404).json({   
 success: false,  
 message: 'Blood profile not found. Please create your profile first.'   
 });  
 }  
   
 res.json({  
 success: true,  
 profile  
 });  
 } catch (error) {  
 console.error('Get donor blood profile error:', error);  
 res.status(500).json({   
 success: false,  
 message: 'Failed to fetch blood profile',  
 error: error.message   
 });  
 }  
});  
  
// Get eligible donors for blood group  
router.get('/eligible/:bloodGroup', async (req, res) => {  
 try {  
 const { bloodGroup } = req.params;  
 const { location } = req.query;  
   
 const eligibleDonors = await DonorBlood.findEligibleDonors(bloodGroup, location);  
   
 res.json({  
 success: true,  
 donors: eligibleDonors,  
 count: eligibleDonors.length  
 });  
 } catch (error) {  
 console.error('Get eligible donors error:', error);  
 res.status(500).json({   
 success: false,  
 message: 'Failed to fetch eligible donors',  
 error: error.message   
 });  
 }  
});  
  
// Get all available donors  
router.get('/available', async (req, res) => {  
 try {  
 const { bloodGroup, location, page = 1, limit = 10 } = req.query;  
   
 const query = {  
 'availability.isAvailable': true,  
 isEligible: true,  
 status: 'active'  
 };  
   
 if (bloodGroup) query.bloodGroup = bloodGroup;  
 if (location) {  
 query['availability.preferredLocations'] = { $regex: location, $options: 'i' };  
 }  
   
 const donors = await DonorBlood.find(query)  
 .sort({ 'stats.responseRate': -1, 'stats.lastActiveDate': -1 })  
 .limit(limit \* 1)  
 .skip((page - 1) \* limit)  
 .select('-medicalHistory -donationHistory');  
   
 const total = await DonorBlood.countDocuments(query);  
   
 res.json({  
 success: true,  
 donors,  
 pagination: {  
 current: page,  
 pages: Math.ceil(total / limit),  
 total  
 }  
 });  
 } catch (error) {  
 console.error('Get available donors error:', error);  
 res.status(500).json({   
 success: false,  
 message: 'Failed to fetch available donors',  
 error: error.message   
 });  
 }  
});  
  
// Get emergency donors  
router.get('/emergency/:bloodGroup', async (req, res) => {  
 try {  
 const { bloodGroup } = req.params;  
   
 const emergencyDonors = await DonorBlood.findEmergencyDonors(bloodGroup);  
   
 res.json({  
 success: true,  
 donors: emergencyDonors,  
 count: emergencyDonors.length  
 });  
 } catch (error) {  
 console.error('Get emergency donors error:', error);  
 res.status(500).json({   
 success: false,  
 message: 'Failed to fetch emergency donors',  
 error: error.message   
 });  
 }  
});  
  
// Update donor availability  
router.patch('/availability', authenticateToken, async (req, res) => {  
 try {  
 if (req.user.role !== 'donor') {  
 return res.status(403).json({   
 success: false,  
 message: 'Only donors can update availability'   
 });  
 }  
   
 const { isAvailable, reason } = req.body;  
   
 const profile = await DonorBlood.findOne({ donorId: req.user.\_id });  
 if (!profile) {  
 return res.status(404).json({   
 success: false,  
 message: 'Blood profile not found'   
 });  
 }  
   
 await profile.updateAvailability(isAvailable, reason);  
   
 res.json({  
 success: true,  
 message: 'Availability updated successfully',  
 profile: profile  
 });  
 } catch (error) {  
 console.error('Update availability error:', error);  
 res.status(500).json({   
 success: false,  
 message: 'Failed to update availability',  
 error: error.message   
 });  
 }  
});  
  
// Add donation record  
router.post('/donation', authenticateToken, async (req, res) => {  
 try {  
 if (req.user.role !== 'donor') {  
 return res.status(403).json({   
 success: false,  
 message: 'Only donors can add donation records'   
 });  
 }  
   
 const profile = await DonorBlood.findOne({ donorId: req.user.\_id });  
 if (!profile) {  
 return res.status(404).json({   
 success: false,  
 message: 'Blood profile not found'   
 });  
 }  
   
 await profile.addDonation(req.body);  
   
 res.json({  
 success: true,  
 message: 'Donation record added successfully',  
 profile: profile  
 });  
 } catch (error) {  
 console.error('Add donation record error:', error);  
 res.status(500).json({   
 success: false,  
 message: 'Failed to add donation record',  
 error: error.message   
 });  
 }  
});  
  
// Get donation history  
router.get('/donation-history', authenticateToken, async (req, res) => {  
 try {  
 if (req.user.role !== 'donor') {  
 return res.status(403).json({   
 success: false,  
 message: 'Only donors can view donation history'   
 });  
 }  
   
 let profile = await DonorBlood.findOne({ donorId: req.user.\_id })  
 .select('donationHistory totalDonations totalUnitsDonated');  
  
 if (!profile) {  
 // Auto-create donor profile if missing  
 profile = await DonorBlood.create({  
 donorId: req.user.\_id,  
 donorName: req.user.name,  
 donorEmail: req.user.email,  
 donorPhone: req.user.phone,  
 bloodGroup: req.user.bloodGroup || 'O+',  
 rhFactor: 'positive',  
 physicalInfo: { age: 30, weight: 60, height: 170, gender: 'other' },  
 });  
 }  
  
 res.json({  
 success: true,  
 donationHistory: profile.donationHistory,  
 totalDonations: profile.totalDonations,  
 totalUnitsDonated: profile.totalUnitsDonated  
 });  
 } catch (error) {  
 console.error('Get donation history error:', error);  
 res.status(500).json({   
 success: false,  
 message: 'Failed to fetch donation history',  
 error: error.message   
 });  
 }  
});  
  
// Update donor statistics  
router.patch('/stats', authenticateToken, async (req, res) => {  
 try {  
 if (req.user.role !== 'donor') {  
 return res.status(403).json({   
 success: false,  
 message: 'Only donors can update statistics'   
 });  
 }  
   
 const { requestType } = req.body;  
   
 const profile = await DonorBlood.findOne({ donorId: req.user.\_id });  
 if (!profile) {  
 return res.status(404).json({   
 success: false,  
 message: 'Blood profile not found'   
 });  
 }  
   
 await profile.updateStats(requestType);  
   
 res.json({  
 success: true,  
 message: 'Statistics updated successfully',  
 stats: profile.stats  
 });  
 } catch (error) {  
 console.error('Update stats error:', error);  
 res.status(500).json({   
 success: false,  
 message: 'Failed to update statistics',  
 error: error.message   
 });  
 }  
});  
  
// Suspend donor (admin only)  
router.patch('/:id/suspend', authenticateToken, async (req, res) => {  
 try {  
 if (req.user.role !== 'admin') {  
 return res.status(403).json({   
 success: false,  
 message: 'Only admins can suspend donors'   
 });  
 }  
   
 const { reason } = req.body;  
 const donorId = req.params.id;  
   
 const profile = await DonorBlood.findOne({ donorId });  
 if (!profile) {  
 return res.status(404).json({   
 success: false,  
 message: 'Donor profile not found'   
 });  
 }  
   
 await profile.suspend(reason);  
   
 res.json({  
 success: true,  
 message: 'Donor suspended successfully',  
 profile: profile  
 });  
 } catch (error) {  
 console.error('Suspend donor error:', error);  
 res.status(500).json({   
 success: false,  
 message: 'Failed to suspend donor',  
 error: error.message   
 });  
 }  
});  
  
// Activate donor (admin only)  
router.patch('/:id/activate', authenticateToken, async (req, res) => {  
 try {  
 if (req.user.role !== 'admin') {  
 return res.status(403).json({   
 success: false,  
 message: 'Only admins can activate donors'   
 });  
 }  
   
 const donorId = req.params.id;  
   
 const profile = await DonorBlood.findOne({ donorId });  
 if (!profile) {  
 return res.status(404).json({   
 success: false,  
 message: 'Donor profile not found'   
 });  
 }  
   
 await profile.activate();  
   
 res.json({  
 success: true,  
 message: 'Donor activated successfully',  
 profile: profile  
 });  
 } catch (error) {  
 console.error('Activate donor error:', error);  
 res.status(500).json({   
 success: false,  
 message: 'Failed to activate donor',  
 error: error.message   
 });  
 }  
});  
  
export default router;

================================================================================

## 20. Dynamic Models

📂 File Path: C:\Users\Admin\React\sai\project(hemo\_connect)\server\routes\dynamic-models.js

import express from 'express';  
import { createDynamicModel, createModelFromDefinition, ModelFactory } from '../utils/modelFactory.js';  
import { authenticateToken, requireRole } from '../middleware/auth.js';  
  
const router = express.Router();  
  
/\*\*  
 \* POST /api/dynamic-models  
 \* Create a new model dynamically  
 \* Body: { name: string, schema: object, collectionName?: string, options?: object }  
 \*/  
router.post('/', authenticateToken, requireRole('admin'), async (req, res) => {  
 try {  
 const { name, schema, collectionName, options = {} } = req.body;  
  
 if (!name || !schema) {  
 return res.status(400).json({  
 success: false,  
 message: 'Model name and schema are required'  
 });  
 }  
  
 const model = await createModelFromDefinition(name, schema, collectionName, options);  
   
 res.status(201).json({  
 success: true,  
 message: `Model '${name}' created successfully`,  
 modelName: name,  
 collectionName: collectionName || name.toLowerCase()  
 });  
 } catch (error) {  
 console.error('Create dynamic model error:', error);  
 res.status(500).json({  
 success: false,  
 message: 'Failed to create dynamic model',  
 error: error.message  
 });  
 }  
});  
  
/\*\*  
 \* POST /api/dynamic-models/simple  
 \* Create a simple model with basic fields  
 \* Body: { name: string, fields?: object, collectionName?: string }  
 \*/  
router.post('/simple', authenticateToken, requireRole('admin'), async (req, res) => {  
 try {  
 const { name, fields = {}, collectionName } = req.body;  
  
 if (!name) {  
 return res.status(400).json({  
 success: false,  
 message: 'Model name is required'  
 });  
 }  
  
 const model = await ModelFactory.createSimple(name, fields, collectionName);  
   
 res.status(201).json({  
 success: true,  
 message: `Simple model '${name}' created successfully`,  
 modelName: name,  
 collectionName: collectionName || name.toLowerCase()  
 });  
 } catch (error) {  
 console.error('Create simple model error:', error);  
 res.status(500).json({  
 success: false,  
 message: 'Failed to create simple model',  
 error: error.message  
 });  
 }  
});  
  
/\*\*  
 \* POST /api/dynamic-models/user-related  
 \* Create a user-related model  
 \* Body: { name: string, fields?: object, collectionName?: string }  
 \*/  
router.post('/user-related', authenticateToken, requireRole('admin'), async (req, res) => {  
 try {  
 const { name, fields = {}, collectionName } = req.body;  
  
 if (!name) {  
 return res.status(400).json({  
 success: false,  
 message: 'Model name is required'  
 });  
 }  
  
 const model = await ModelFactory.createUserRelated(name, fields, collectionName);  
   
 res.status(201).json({  
 success: true,  
 message: `User-related model '${name}' created successfully`,  
 modelName: name,  
 collectionName: collectionName || name.toLowerCase()  
 });  
 } catch (error) {  
 console.error('Create user-related model error:', error);  
 res.status(500).json({  
 success: false,  
 message: 'Failed to create user-related model',  
 error: error.message  
 });  
 }  
});  
  
/\*\*  
 \* POST /api/dynamic-models/content  
 \* Create a content model  
 \* Body: { name: string, fields?: object, collectionName?: string }  
 \*/  
router.post('/content', authenticateToken, requireRole('admin'), async (req, res) => {  
 try {  
 const { name, fields = {}, collectionName } = req.body;  
  
 if (!name) {  
 return res.status(400).json({  
 success: false,  
 message: 'Model name is required'  
 });  
 }  
  
 const model = await ModelFactory.createContent(name, fields, collectionName);  
   
 res.status(201).json({  
 success: true,  
 message: `Content model '${name}' created successfully`,  
 modelName: name,  
 collectionName: collectionName || name.toLowerCase()  
 });  
 } catch (error) {  
 console.error('Create content model error:', error);  
 res.status(500).json({  
 success: false,  
 message: 'Failed to create content model',  
 error: error.message  
 });  
 }  
});  
  
/\*\*  
 \* POST /api/dynamic-models/transaction  
 \* Create a transaction model  
 \* Body: { name: string, fields?: object, collectionName?: string }  
 \*/  
router.post('/transaction', authenticateToken, requireRole('admin'), async (req, res) => {  
 try {  
 const { name, fields = {}, collectionName } = req.body;  
  
 if (!name) {  
 return res.status(400).json({  
 success: false,  
 message: 'Model name is required'  
 });  
 }  
  
 const model = await ModelFactory.createTransaction(name, fields, collectionName);  
   
 res.status(201).json({  
 success: true,  
 message: `Transaction model '${name}' created successfully`,  
 modelName: name,  
 collectionName: collectionName || name.toLowerCase()  
 });  
 } catch (error) {  
 console.error('Create transaction model error:', error);  
 res.status(500).json({  
 success: false,  
 message: 'Failed to create transaction model',  
 error: error.message  
 });  
 }  
});  
  
/\*\*  
 \* GET /api/dynamic-models/templates  
 \* Get available model templates  
 \*/  
router.get('/templates', authenticateToken, (req, res) => {  
 const templates = {  
 simple: {  
 description: 'Basic model with name, description, and isActive fields',  
 fields: {  
 name: { type: 'String', required: true },  
 description: { type: 'String' },  
 isActive: { type: 'Boolean', default: true }  
 }  
 },  
 userRelated: {  
 description: 'Model with user references and audit fields',  
 fields: {  
 userId: { type: 'ObjectId', ref: 'User', required: true },  
 createdBy: { type: 'ObjectId', ref: 'User' },  
 updatedBy: { type: 'ObjectId', ref: 'User' }  
 }  
 },  
 content: {  
 description: 'Content model for posts, articles, etc.',  
 fields: {  
 title: { type: 'String', required: true },  
 content: { type: 'String', required: true },  
 author: { type: 'ObjectId', ref: 'User', required: true },  
 tags: { type: '[String]' },  
 isPublished: { type: 'Boolean', default: false },  
 views: { type: 'Number', default: 0 },  
 likes: { type: 'Number', default: 0 }  
 }  
 },  
 transaction: {  
 description: 'Transaction/audit model',  
 fields: {  
 type: { type: 'String', required: true },  
 amount: { type: 'Number' },  
 currency: { type: 'String', default: 'USD' },  
 fromUser: { type: 'ObjectId', ref: 'User' },  
 toUser: { type: 'ObjectId', ref: 'User' },  
 status: { type: 'String', enum: ['pending', 'completed', 'failed', 'cancelled'], default: 'pending' },  
 reference: { type: 'String' },  
 metadata: { type: 'Mixed' }  
 }  
 }  
 };  
  
 res.json({  
 success: true,  
 templates  
 });  
});  
  
export default router;

================================================================================

## 21. Notifications

📂 File Path: C:\Users\Admin\React\sai\project(hemo\_connect)\server\routes\notifications.js

import express from 'express';  
import Notification from '../models/Notification.js';  
import { authenticateToken } from '../middleware/auth.js';  
  
const router = express.Router();  
  
// Create a notification  
router.post('/', authenticateToken, async (req, res) => {  
 try {  
 const { type, title, message } = req.body;  
 const notification = await Notification.create({  
 userId: req.user.\_id,  
 type,  
 title,  
 message,  
 });  
 res.status(201).json(notification);  
 } catch (error) {  
 console.error('Create notification error:', error);  
 res.status(500).json({ message: 'Failed to create notification' });  
 }  
});  
  
// Get all notifications for user  
router.get('/', authenticateToken, async (req, res) => {  
 try {  
 const notifications = await Notification.find({ userId: req.user.\_id }).sort({ timestamp: -1 });  
 res.json(notifications);  
 } catch (error) {  
 console.error('Get notifications error:', error);  
 res.status(500).json({ message: 'Failed to fetch notifications' });  
 }  
});  
  
// Mark notification as read  
router.put('/:id/read', authenticateToken, async (req, res) => {  
 try {  
 const notification = await Notification.findOneAndUpdate(  
 { \_id: req.params.id, userId: req.user.\_id },  
 { isRead: true },  
 { new: true }  
 );  
 if (!notification) {  
 return res.status(404).json({ message: 'Notification not found' });  
 }  
 res.json(notification);  
 } catch (error) {  
 console.error('Mark notification read error:', error);  
 res.status(500).json({ message: 'Failed to mark notification as read' });  
 }  
});  
  
export default router;

================================================================================

## 22. Requests

📂 File Path: C:\Users\Admin\React\sai\project(hemo\_connect)\server\routes\requests.js

================================================================================

## 23. Users

📂 File Path: C:\Users\Admin\React\sai\project(hemo\_connect)\server\routes\users.js

import express from 'express';  
import bcrypt from 'bcryptjs';  
import User from '../models/User.js';  
import { authenticateToken } from '../middleware/auth.js';  
  
const router = express.Router();  
  
// Update password  
router.put('/update-password', authenticateToken, async (req, res) => {  
 try {  
 const { oldPassword, newPassword } = req.body;  
 const user = await User.findById(req.user.\_id);  
 if (!user) {  
 return res.status(404).json({ message: 'User not found' });  
 }  
 const isMatch = await bcrypt.compare(oldPassword, user.password);  
 if (!isMatch) {  
 return res.status(400).json({ message: 'Old password is incorrect' });  
 }  
 user.password = await bcrypt.hash(newPassword, 12);  
 user.updatedAt = new Date();  
 await user.save();  
 res.json({ message: 'Password updated successfully' });  
 } catch (error) {  
 console.error('Password update error:', error);  
 res.status(500).json({ message: 'Failed to update password' });  
 }  
});  
  
// Update profile picture  
router.put('/profile-pic', authenticateToken, async (req, res) => {  
 try {  
 const { profilePic } = req.body;  
 const user = await User.findByIdAndUpdate(  
 req.user.\_id,  
 { profilePic, updatedAt: new Date() },  
 { new: true }  
 );  
 if (!user) {  
 return res.status(404).json({ message: 'User not found' });  
 }  
 res.json({ message: 'Profile picture updated', profilePic: user.profilePic });  
 } catch (error) {  
 console.error('Profile pic update error:', error);  
 res.status(500).json({ message: 'Failed to update profile picture' });  
 }  
});  
  
// Complete profile setup  
router.post('/complete-profile', authenticateToken, async (req, res) => {  
 try {  
 const updateData = {  
 ...req.body,  
 profileComplete: true,  
 updatedAt: new Date(),  
 };  
 let userId = req.user.\_id;  
 if (typeof userId !== 'string') userId = String(userId);  
 console.log('Profile setup for userId:', userId);  
 const updatedUser = await User.findByIdAndUpdate(userId, updateData, { new: true });  
 if (!updatedUser) {  
 console.error('Profile setup failed: User not found for userId', userId);  
 return res.status(404).json({ message: 'User not found', userId });  
 }  
 res.json(updatedUser);  
 } catch (error) {  
 console.error('Profile setup error:', error);  
 res.status(500).json({ message: 'Failed to complete profile setup' });  
 }  
});  
  
// Save draft profile  
router.post('/save-draft', authenticateToken, async (req, res) => {  
 try {  
 const updateData = {  
 ...req.body,  
 updatedAt: new Date(),  
 };  
 let userId = req.user.\_id;  
 if (typeof userId !== 'string') userId = String(userId);  
 console.log('Save draft for userId:', userId);  
 const updatedUser = await User.findByIdAndUpdate(userId, updateData, { new: true });  
 if (!updatedUser) {  
 console.error('Save draft failed: User not found for userId', userId);  
 return res.status(404).json({ message: 'User not found', userId });  
 }  
 res.json({ message: 'Draft saved successfully' });  
 } catch (error) {  
 console.error('Save draft error:', error);  
 res.status(500).json({ message: 'Failed to save draft' });  
 }  
});  
  
// Toggle availability (for donors)  
router.post('/toggle-availability', authenticateToken, async (req, res) => {  
 try {  
 if (req.user.role !== 'donor') {  
 return res.status(403).json({ message: 'Only donors can toggle availability' });  
 }  
  
 const { isAvailable } = req.body;  
 const updateData = {  
 isAvailable,  
 lastActive: new Date(),  
 updatedAt: new Date(),  
 };  
  
 let userId = req.user.\_id;  
 if (typeof userId !== 'string') userId = String(userId);  
 console.log('Toggle availability for userId:', userId);  
 const updatedUser = await User.findByIdAndUpdate(userId, updateData, { new: true });  
 if (!updatedUser) {  
 console.error('Toggle availability failed: User not found for userId', userId);  
 return res.status(404).json({ message: 'User not found', userId });  
 }  
 res.json({   
 message: 'Availability updated successfully',  
 isAvailable: updatedUser.isAvailable   
 });  
 } catch (error) {  
 console.error('Toggle availability error:', error);  
 res.status(500).json({ message: 'Failed to update availability' });  
 }  
});  
  
// Get user profile  
router.get('/profile', authenticateToken, async (req, res) => {  
 let userId = req.user.\_id;  
 if (typeof userId !== 'string') userId = String(userId);  
 console.log('Get profile for userId:', userId);  
 const user = await User.findById(userId);  
 if (!user) {  
 console.error('Get profile failed: User not found for userId', userId);  
 return res.status(404).json({ message: 'User not found', userId });  
 }  
 res.json(user);  
});  
  
// Update user profile  
router.put('/profile', authenticateToken, async (req, res) => {  
 try {  
 const updateData = {  
 ...req.body,  
 updatedAt: new Date(),  
 };  
  
 let userId = req.user.\_id;  
 if (typeof userId !== 'string') userId = String(userId);  
 console.log('Update profile for userId:', userId);  
 const updatedUser = await User.findByIdAndUpdate(userId, updateData, { new: true });  
 if (!updatedUser) {  
 console.error('Update profile failed: User not found for userId', userId);  
 return res.status(404).json({ message: 'User not found', userId });  
 }  
 res.json(updatedUser);  
 } catch (error) {  
 console.error('Profile update error:', error);  
 res.status(500).json({ message: 'Failed to update profile' });  
 }  
});  
  
export default router;

================================================================================

## 24. Collectionmanager

📂 File Path: C:\Users\Admin\React\sai\project(hemo\_connect)\server\utils\collectionManager.js

import mongoose from 'mongoose';  
  
/\*\*  
 \* Dynamic Collection Manager  
 \* Handles creation and management of MongoDB collections on-demand  
 \*/  
  
// Cache for created collections to avoid redundant operations  
const createdCollections = new Set();  
  
/\*\*  
 \* Validates collection name according to MongoDB naming rules  
 \* @param {string} name - Collection name to validate  
 \* @returns {object} - { isValid: boolean, error?: string }  
 \*/  
export const validateCollectionName = (name) => {  
 if (!name || typeof name !== 'string') {  
 return { isValid: false, error: 'Collection name must be a non-empty string' };  
 }  
  
 // MongoDB collection name rules  
 const rules = [  
 { test: name.length === 0, error: 'Collection name cannot be empty' },  
 { test: name.length > 120, error: 'Collection name cannot exceed 120 characters' },  
 { test: name.startsWith('$'), error: 'Collection name cannot start with $' },  
 { test: name.includes('..'), error: 'Collection name cannot contain ..' },  
 { test: name.includes('\0'), error: 'Collection name cannot contain null character' },  
 { test: /[<>:"/\\|?\*]/.test(name), error: 'Collection name contains invalid characters' },  
 { test: name.trim() !== name, error: 'Collection name cannot have leading/trailing spaces' }  
 ];  
  
 for (const rule of rules) {  
 if (rule.test) {  
 return { isValid: false, error: rule.error };  
 }  
 }  
  
 return { isValid: true };  
};  
  
/\*\*  
 \* Creates a collection if it doesn't exist  
 \* @param {string} collectionName - Name of the collection to create  
 \* @param {object} options - Additional options for collection creation  
 \* @returns {Promise<object>} - { success: boolean, message: string, collectionName?: string }  
 \*/  
export const createCollection = async (collectionName, options = {}) => {  
 try {  
 // Validate collection name  
 const validation = validateCollectionName(collectionName);  
 if (!validation.isValid) {  
 return { success: false, message: validation.error };  
 }  
  
 // Check if already created in this session  
 if (createdCollections.has(collectionName)) {  
 return {   
 success: true,   
 message: `Collection '${collectionName}' already exists in this session`,  
 collectionName   
 };  
 }  
  
 // Check if collection already exists in database  
 const db = mongoose.connection.db;  
 if (!db) {  
 return { success: false, message: 'Database connection not available' };  
 }  
  
 const existingCollections = await db.listCollections({ name: collectionName }).toArray();  
 if (existingCollections.length > 0) {  
 createdCollections.add(collectionName);  
 return {   
 success: true,   
 message: `Collection '${collectionName}' already exists in database`,  
 collectionName   
 };  
 }  
  
 // Create the collection  
 await db.createCollection(collectionName, {  
 validator: options.validator || {},  
 validationLevel: options.validationLevel || 'strict',  
 validationAction: options.validationAction || 'error',  
 ...options  
 });  
  
 createdCollections.add(collectionName);  
 console.log(`✅ Created collection: ${collectionName}`);  
   
 return {   
 success: true,   
 message: `Collection '${collectionName}' created successfully`,  
 collectionName   
 };  
  
 } catch (error) {  
 console.error(`❌ Failed to create collection '${collectionName}':`, error);  
 return {   
 success: false,   
 message: `Failed to create collection: ${error.message}`   
 };  
 }  
};  
  
/\*\*  
 \* Creates multiple collections at once  
 \* @param {string[]} collectionNames - Array of collection names  
 \* @param {object} options - Options for all collections  
 \* @returns {Promise<object>} - Results for each collection  
 \*/  
export const createMultipleCollections = async (collectionNames, options = {}) => {  
 const results = {};  
   
 for (const name of collectionNames) {  
 results[name] = await createCollection(name, options);  
 }  
   
 return results;  
};  
  
/\*\*  
 \* Ensures a collection exists before performing operations  
 \* @param {string} collectionName - Name of the collection  
 \* @param {object} options - Creation options  
 \* @returns {Promise<boolean>} - True if collection exists or was created successfully  
 \*/  
export const ensureCollection = async (collectionName, options = {}) => {  
 const result = await createCollection(collectionName, options);  
 return result.success;  
};  
  
/\*\*  
 \* Gets list of all collections in the database  
 \* @returns {Promise<string[]>} - Array of collection names  
 \*/  
export const listCollections = async () => {  
 try {  
 const db = mongoose.connection.db;  
 if (!db) {  
 throw new Error('Database connection not available');  
 }  
  
 const collections = await db.listCollections().toArray();  
 return collections.map(col => col.name);  
 } catch (error) {  
 console.error('Failed to list collections:', error);  
 throw error;  
 }  
};  
  
/\*\*  
 \* Drops a collection (use with caution)  
 \* @param {string} collectionName - Name of the collection to drop  
 \* @returns {Promise<object>} - { success: boolean, message: string }  
 \*/  
export const dropCollection = async (collectionName) => {  
 try {  
 const db = mongoose.connection.db;  
 if (!db) {  
 return { success: false, message: 'Database connection not available' };  
 }  
  
 await db.dropCollection(collectionName);  
 createdCollections.delete(collectionName);  
 console.log(`🗑️ Dropped collection: ${collectionName}`);  
   
 return {   
 success: true,   
 message: `Collection '${collectionName}' dropped successfully`   
 };  
 } catch (error) {  
 console.error(`Failed to drop collection '${collectionName}':`, error);  
 return {   
 success: false,   
 message: `Failed to drop collection: ${error.message}`   
 };  
 }  
};  
  
/\*\*  
 \* Creates a Mongoose model with dynamic collection creation  
 \* @param {string} modelName - Name of the model  
 \* @param {mongoose.Schema} schema - Mongoose schema  
 \* @param {string} collectionName - Name of the collection (optional, defaults to modelName)  
 \* @param {object} options - Collection creation options  
 \* @returns {mongoose.Model} - Mongoose model  
 \*/  
export const createModelWithCollection = async (modelName, schema, collectionName = null, options = {}) => {  
 const finalCollectionName = collectionName || modelName.toLowerCase();  
   
 // Ensure collection exists  
 await ensureCollection(finalCollectionName, options);  
   
 // Create and return the model  
 return mongoose.model(modelName, schema, finalCollectionName);  
};  
  
/\*\*  
 \* Middleware for automatic collection creation on model operations  
 \* @param {string} collectionName - Name of the collection  
 \* @param {object} options - Collection creation options  
 \* @returns {Function} - Express middleware function  
 \*/  
export const autoCreateCollection = (collectionName, options = {}) => {  
 return async (req, res, next) => {  
 try {  
 await ensureCollection(collectionName, options);  
 next();  
 } catch (error) {  
 console.error(`Auto-create collection failed for '${collectionName}':`, error);  
 res.status(500).json({   
 message: 'Failed to ensure collection exists',  
 error: error.message   
 });  
 }  
 };  
};  
  
export default {  
 validateCollectionName,  
 createCollection,  
 createMultipleCollections,  
 ensureCollection,  
 listCollections,  
 dropCollection,  
 createModelWithCollection,  
 autoCreateCollection  
};

================================================================================

## 25. Modelfactory

📂 File Path: C:\Users\Admin\React\sai\project(hemo\_connect)\server\utils\modelFactory.js

import mongoose from 'mongoose';  
import { createModelWithCollection, ensureCollection } from './collectionManager.js';  
  
/\*\*  
 \* Model Factory for creating dynamic Mongoose models with automatic collection creation  
 \*/  
  
/\*\*  
 \* Creates a model with automatic collection creation  
 \* @param {string} modelName - Name of the model (e.g., 'User', 'Product')  
 \* @param {mongoose.Schema} schema - Mongoose schema definition  
 \* @param {string} collectionName - Custom collection name (optional)  
 \* @param {object} options - Collection creation options  
 \* @returns {Promise<mongoose.Model>} - Mongoose model  
 \*/  
export const createDynamicModel = async (modelName, schema, collectionName = null, options = {}) => {  
 try {  
 const finalCollectionName = collectionName || modelName.toLowerCase();  
   
 // Ensure collection exists  
 await ensureCollection(finalCollectionName, options);  
   
 // Create and return the model  
 const model = mongoose.model(modelName, schema, finalCollectionName);  
 console.log(`✅ Created dynamic model: ${modelName} -> collection: ${finalCollectionName}`);  
   
 return model;  
 } catch (error) {  
 console.error(`❌ Failed to create dynamic model ${modelName}:`, error);  
 throw error;  
 }  
};  
  
/\*\*  
 \* Creates a model from a schema definition object  
 \* @param {string} modelName - Name of the model  
 \* @param {object} schemaDefinition - Schema definition object  
 \* @param {string} collectionName - Custom collection name (optional)  
 \* @param {object} options - Collection creation options  
 \* @returns {Promise<mongoose.Model>} - Mongoose model  
 \*/  
export const createModelFromDefinition = async (modelName, schemaDefinition, collectionName = null, options = {}) => {  
 try {  
 const schema = new mongoose.Schema(schemaDefinition, {  
 timestamps: true, // Automatically add createdAt and updatedAt  
 ...options.schemaOptions  
 });  
   
 return await createDynamicModel(modelName, schema, collectionName, options);  
 } catch (error) {  
 console.error(`❌ Failed to create model from definition ${modelName}:`, error);  
 throw error;  
 }  
};  
  
/\*\*  
 \* Creates multiple models at once  
 \* @param {Array} modelDefinitions - Array of model definitions  
 \* @returns {Promise<object>} - Object with model names as keys and models as values  
 \*/  
export const createMultipleModels = async (modelDefinitions) => {  
 const models = {};  
   
 for (const definition of modelDefinitions) {  
 const { name, schema, collectionName, options = {} } = definition;  
   
 try {  
 models[name] = await createDynamicModel(name, schema, collectionName, options);  
 } catch (error) {  
 console.error(`Failed to create model ${name}:`, error);  
 models[name] = null;  
 }  
 }  
   
 return models;  
};  
  
/\*\*  
 \* Factory function for common model patterns  
 \*/  
export const ModelFactory = {  
 /\*\*  
 \* Creates a simple model with basic fields  
 \* @param {string} name - Model name  
 \* @param {object} fields - Additional fields to add to basic schema  
 \* @param {string} collectionName - Collection name  
 \* @returns {Promise<mongoose.Model>}  
 \*/  
 createSimple: async (name, fields = {}, collectionName = null) => {  
 const basicSchema = {  
 name: { type: String, required: true },  
 description: { type: String },  
 isActive: { type: Boolean, default: true },  
 ...fields  
 };  
   
 return await createModelFromDefinition(name, basicSchema, collectionName);  
 },  
  
 /\*\*  
 \* Creates a user-related model  
 \* @param {string} name - Model name  
 \* @param {object} fields - Additional fields  
 \* @param {string} collectionName - Collection name  
 \* @returns {Promise<mongoose.Model>}  
 \*/  
 createUserRelated: async (name, fields = {}, collectionName = null) => {  
 const userSchema = {  
 userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },  
 createdBy: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },  
 updatedBy: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },  
 ...fields  
 };  
   
 return await createModelFromDefinition(name, userSchema, collectionName);  
 },  
  
 /\*\*  
 \* Creates a content model (for posts, articles, etc.)  
 \* @param {string} name - Model name  
 \* @param {object} fields - Additional fields  
 \* @param {string} collectionName - Collection name  
 \* @returns {Promise<mongoose.Model>}  
 \*/  
 createContent: async (name, fields = {}, collectionName = null) => {  
 const contentSchema = {  
 title: { type: String, required: true },  
 content: { type: String, required: true },  
 author: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },  
 tags: [{ type: String }],  
 isPublished: { type: Boolean, default: false },  
 views: { type: Number, default: 0 },  
 likes: { type: Number, default: 0 },  
 ...fields  
 };  
   
 return await createModelFromDefinition(name, contentSchema, collectionName);  
 },  
  
 /\*\*  
 \* Creates a transaction/audit model  
 \* @param {string} name - Model name  
 \* @param {object} fields - Additional fields  
 \* @param {string} collectionName - Collection name  
 \* @returns {Promise<mongoose.Model>}  
 \*/  
 createTransaction: async (name, fields = {}, collectionName = null) => {  
 const transactionSchema = {  
 type: { type: String, required: true },  
 amount: { type: Number },  
 currency: { type: String, default: 'USD' },  
 fromUser: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },  
 toUser: { type: mongoose.Schema.Types.ObjectId, ref: 'User' },  
 status: { type: String, enum: ['pending', 'completed', 'failed', 'cancelled'], default: 'pending' },  
 reference: { type: String },  
 metadata: { type: mongoose.Schema.Types.Mixed },  
 ...fields  
 };  
   
 return await createModelFromDefinition(name, transactionSchema, collectionName);  
 }  
};  
  
/\*\*  
 \* Middleware for automatic model creation  
 \* @param {string} modelName - Name of the model  
 \* @param {object} schemaDefinition - Schema definition  
 \* @param {string} collectionName - Collection name  
 \* @returns {Function} - Express middleware  
 \*/  
export const autoCreateModel = (modelName, schemaDefinition, collectionName = null) => {  
 let model = null;  
   
 return async (req, res, next) => {  
 try {  
 if (!model) {  
 model = await createModelFromDefinition(modelName, schemaDefinition, collectionName);  
 }  
   
 req.model = model;  
 next();  
 } catch (error) {  
 console.error(`Auto-create model failed for '${modelName}':`, error);  
 res.status(500).json({  
 message: 'Failed to create model',  
 error: error.message  
 });  
 }  
 };  
};  
  
export default {  
 createDynamicModel,  
 createModelFromDefinition,  
 createMultipleModels,  
 ModelFactory,  
 autoCreateModel  
};

================================================================================

## 26. App

📂 File Path: C:\Users\Admin\React\sai\project(hemo\_connect)\src\App.jsx

import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';  
import { AuthProvider } from './context/AuthContext';  
import { SocketProvider } from './context/SocketContext';  
import Layout from './components/Layout/Layout';  
import ProtectedRoute from './components/Auth/ProtectedRoute';  
import HomePage from './pages/Home/HomePage';  
import LoginPage from './pages/Auth/LoginPage';  
import SignupPage from './pages/Auth/SignupPage';  
import ForgotPasswordPage from './pages/Auth/ForgotPasswordPage';  
import ProfileSetupPage from './pages/Auth/ProfileSetupPage';  
import ProfilePage from './pages/Auth/ProfilePage';  
import DonorDashboard from './pages/Dashboard/DonorDashboard';  
import RecipientDashboard from './pages/Dashboard/RecipientDashboard';  
import AdminDashboard from './pages/Dashboard/AdminDashboard';  
import BloodRequestPage from './pages/Request/BloodRequestPage';  
import DonorMatchingPage from './pages/Matching/DonorMatchingPage';  
import NotificationsPage from './pages/Notifications/NotificationsPage';  
import DonorHistoryPage from './pages/History/DonorHistoryPage';  
import RecipientHistoryPage from './pages/History/RecipientHistoryPage';  
import AboutPage from './pages/About/AboutPage';  
import ContactPage from './pages/Contact/ContactPage';  
import NotFoundPage from './pages/Error/NotFoundPage';  
import UnauthorizedPage from './pages/Error/UnauthorizedPage';  
import ServerErrorPage from './pages/Error/ServerErrorPage';  
import NoDonorsPage from './pages/Error/NoDonorsPage';  
  
function App() {  
 return (  
 <AuthProvider>  
 <SocketProvider>  
 <Router>  
 <Layout>  
 <Routes>  
 {/\* Public Routes \*/}  
 <Route path="/" element={<HomePage />} />  
 <Route path="/login" element={<LoginPage />} />  
 <Route path="/forgot-password" element={<ForgotPasswordPage />} />  
 <Route path="/signup" element={<SignupPage />} />  
 <Route path="/about" element={<AboutPage />} />  
 <Route path="/contact" element={<ContactPage />} />  
   
 {/\* Protected Routes \*/}  
 <Route path="/profile-setup" element={  
 <ProtectedRoute>  
 <ProfileSetupPage />  
 </ProtectedRoute>  
 } />  
  
 <Route path="/profile" element={  
 <ProtectedRoute>  
 <ProfilePage />  
 </ProtectedRoute>  
 } />  
   
 {/\* Dashboard Routes \*/}  
 <Route path="/dashboard" element={  
 <ProtectedRoute>  
 <DonorDashboard />  
 </ProtectedRoute>  
 } />  
 <Route path="/donor-dashboard" element={  
 <ProtectedRoute requiredRole="donor">  
 <DonorDashboard />  
 </ProtectedRoute>  
 } />  
 <Route path="/recipient-dashboard" element={  
 <ProtectedRoute requiredRole="recipient">  
 <RecipientDashboard />  
 </ProtectedRoute>  
 } />  
 <Route path="/admin-dashboard" element={  
 <ProtectedRoute requiredRole="admin">  
 <AdminDashboard />  
 </ProtectedRoute>  
 } />  
   
 {/\* Feature Routes \*/}  
 <Route path="/request-blood" element={  
 <ProtectedRoute>  
 <BloodRequestPage />  
 </ProtectedRoute>  
 } />  
 <Route path="/find-donors" element={  
 <ProtectedRoute>  
 <DonorMatchingPage />  
 </ProtectedRoute>  
 } />  
 <Route path="/notifications" element={  
 <ProtectedRoute>  
 <NotificationsPage />  
 </ProtectedRoute>  
 } />  
 <Route path="/donor-history" element={  
 <ProtectedRoute requiredRole="donor">  
 <DonorHistoryPage />  
 </ProtectedRoute>  
 } />  
 <Route path="/recipient-history" element={  
 <ProtectedRoute requiredRole="recipient">  
 <RecipientHistoryPage />  
 </ProtectedRoute>  
 } />  
   
 {/\* Error Routes \*/}  
 <Route path="/unauthorized" element={<UnauthorizedPage />} />  
 <Route path="/server-error" element={<ServerErrorPage />} />  
 <Route path="/no-donors" element={<NoDonorsPage />} />  
 <Route path="\*" element={<NotFoundPage />} />  
 </Routes>  
 </Layout>  
 </Router>  
 </SocketProvider>  
 </AuthProvider>  
 );  
}  
  
export default App;

================================================================================

## 27. Index

📂 File Path: C:\Users\Admin\React\sai\project(hemo\_connect)\src\index.css

@tailwind base;  
@tailwind components;  
@tailwind utilities;

================================================================================

## 28. Main

📂 File Path: C:\Users\Admin\React\sai\project(hemo\_connect)\src\main.jsx

import { StrictMode } from 'react';  
import { createRoot } from 'react-dom/client';  
import { ChakraProvider } from '@chakra-ui/react';  
import App from './App.jsx';  
import theme from './theme/index.jsx';  
import './index.css';  
  
createRoot(document.getElementById('root')).render(  
 <StrictMode>  
 <ChakraProvider theme={theme}>  
 <App />  
 </ChakraProvider>  
 </StrictMode>  
);

================================================================================

## 29. Protectedroute

📂 File Path: C:\Users\Admin\React\sai\project(hemo\_connect)\src\components\Auth\ProtectedRoute.jsx

import { Navigate, useLocation } from 'react-router-dom';  
import { Box, Spinner, Center } from '@chakra-ui/react';  
import { useAuth } from '../../context/AuthContext';  
  
function ProtectedRoute({ children, requiredRole = null }) {  
 const { isAuthenticated, isLoading, user } = useAuth();  
 const location = useLocation();  
  
 if (isLoading) {  
 return (  
 <Center minH="50vh">  
 <Spinner size="xl" color="primary.500" />  
 </Center>  
 );  
 }  
  
 if (!isAuthenticated) {  
 return <Navigate to="/login" state={{ from: location }} replace />;  
 }  
  
 if (requiredRole && user?.role !== requiredRole) {  
 return <Navigate to="/unauthorized" replace />;  
 }  
  
 // Check if user needs to complete profile setup  
 if (user && !user.profileComplete && location.pathname !== '/profile-setup') {  
 return <Navigate to="/profile-setup" replace />;  
 }  
  
 return <Box>{children}</Box>;  
}  
  
export default ProtectedRoute;

================================================================================

## 30. Footer

📂 File Path: C:\Users\Admin\React\sai\project(hemo\_connect)\src\components\Layout\Footer.jsx

import {  
 Box,  
 Container,  
 Stack,  
 Text,  
 Link,  
 HStack,  
 IconButton,  
} from '@chakra-ui/react';  
import { FiHeart, FiFacebook, FiTwitter, FiInstagram } from 'react-icons/fi';  
  
function Footer() {  
 return (  
 <Box bg="gray.900" color="white">  
 <Container as={Stack} maxW="6xl" py={10}>  
 <Stack  
 direction={{ base: 'column', md: 'row' }}  
 spacing={4}  
 justify={{ base: 'center', md: 'space-between' }}  
 align={{ base: 'center', md: 'center' }}  
 >  
 <HStack>  
 <Box as={FiHeart} color="primary.400" />  
 <Text fontWeight="bold">HEMO CONNECT</Text>  
 </HStack>  
   
 <Text fontSize="sm">  
 Saving lives one donation at a time  
 </Text>  
   
 <HStack spacing={4}>  
 <IconButton  
 as="a"  
 href="#"  
 aria-label="Facebook"  
 icon={<FiFacebook />}  
 variant="ghost"  
 size="sm"  
 color="white"  
 \_hover={{ color: 'primary.400' }}  
 />  
 <IconButton  
 as="a"  
 href="#"  
 aria-label="Twitter"  
 icon={<FiTwitter />}  
 variant="ghost"  
 size="sm"  
 color="white"  
 \_hover={{ color: 'primary.400' }}  
 />  
 <IconButton  
 as="a"  
 href="#"  
 aria-label="Instagram"  
 icon={<FiInstagram />}  
 variant="ghost"  
 size="sm"  
 color="white"  
 \_hover={{ color: 'primary.400' }}  
 />  
 </HStack>  
 </Stack>  
   
 <Text pt={6} fontSize="xs" textAlign="center" color="gray.400">  
 © 2025 Hemo Connect. All rights reserved.  
 </Text>  
 </Container>  
 </Box>  
 );  
}  
  
export default Footer;

================================================================================

## 31. Layout

📂 File Path: C:\Users\Admin\React\sai\project(hemo\_connect)\src\components\Layout\Layout.jsx

import { Box } from '@chakra-ui/react';  
import Navbar from './Navbar';  
import Footer from './Footer';  
  
function Layout({ children }) {  
 return (  
 <Box minH="100vh" display="flex" flexDirection="column">  
 <Navbar />  
 <Box flex="1" pt="80px">  
 {children}  
 </Box>  
 <Footer />  
 </Box>  
 );  
}  
  
export default Layout;

================================================================================

## 32. Navbar

📂 File Path: C:\Users\Admin\React\sai\project(hemo\_connect)\src\components\Layout\Navbar.jsx

import {  
 Box,  
 Flex,  
 HStack,  
 Link,  
 IconButton,  
 Button,  
 Menu,  
 MenuButton,  
 MenuList,  
 MenuItem,  
 MenuDivider,  
 useDisclosure,  
 Stack,  
 Avatar,  
 Text,  
 Badge,  
} from '@chakra-ui/react';  
import { HamburgerIcon, CloseIcon, BellIcon } from '@chakra-ui/icons';  
import { FiHeart } from 'react-icons/fi';  
import { Link as RouterLink, useNavigate } from 'react-router-dom';  
import { useAuth } from '../../context/AuthContext';  
import { useSocket } from '../../context/SocketContext';  
  
function Navbar() {  
 const { isOpen, onOpen, onClose } = useDisclosure();  
 const { isAuthenticated, user, logout } = useAuth();  
 const { notifications } = useSocket();  
 const navigate = useNavigate();  
  
 const handleLogout = () => {  
 logout();  
 navigate('/');  
 };  
  
 const getDashboardLink = () => {  
 if (!user) return '/dashboard';  
 switch (user.role) {  
 case 'donor':  
 return '/donor-dashboard';  
 case 'recipient':  
 return '/recipient-dashboard';  
 case 'admin':  
 return '/admin-dashboard';  
 default:  
 return '/dashboard';  
 }  
 };  
  
 return (  
 <Box   
 bg="white"   
 px={4}   
 position="fixed"   
 top={0}   
 left={0}   
 right={0}   
 zIndex={1000}  
 borderBottom="1px"  
 borderColor="gray.200"  
 boxShadow="sm"  
 >  
 <Flex h={16} alignItems="center" justifyContent="space-between">  
 <IconButton  
 size="md"  
 icon={isOpen ? <CloseIcon /> : <HamburgerIcon />}  
 aria-label="Open Menu"  
 display={{ md: 'none' }}  
 onClick={isOpen ? onClose : onOpen}  
 />  
   
 <HStack spacing={8} alignItems="center">  
 <Box as={RouterLink} to="/" display="flex" alignItems="center" gap={2}>  
 <Box as={FiHeart} size="24px" color="primary.500" />  
 <Text fontSize="xl" fontWeight="bold" color="primary.500">  
 HEMO CONNECT  
 </Text>  
 </Box>  
   
 <HStack as="nav" spacing={4} display={{ base: 'none', md: 'flex' }}>  
 <Link as={RouterLink} to="/about">About</Link>  
 <Link as={RouterLink} to="/contact">Contact</Link>  
 {isAuthenticated && (  
 <>  
 <Link as={RouterLink} to={getDashboardLink()}>Dashboard</Link>  
 <Link as={RouterLink} to="/find-donors">Find Donors</Link>  
 </>  
 )}  
 </HStack>  
 </HStack>  
  
 <Flex alignItems="center">  
 {isAuthenticated ? (  
 <HStack spacing={4}>  
 <Menu>  
 <MenuButton  
 as={IconButton}  
 icon={<BellIcon />}  
 variant="ghost"  
 position="relative"  
 >  
 {notifications.length > 0 && (  
 <Badge  
 colorScheme="red"  
 borderRadius="full"  
 position="absolute"  
 top="-1"  
 right="-1"  
 fontSize="xs"  
 >  
 {notifications.length}  
 </Badge>  
 )}  
 </MenuButton>  
 <MenuList>  
 {notifications.length > 0 ? (  
 notifications.slice(0, 5).map((notification) => (  
 <MenuItem key={notification.id}>  
 <Box>  
 <Text fontWeight="medium">{notification.title}</Text>  
 <Text fontSize="sm" color="gray.600">  
 {notification.message}  
 </Text>  
 </Box>  
 </MenuItem>  
 ))  
 ) : (  
 <MenuItem>  
 <Text color="gray.500">No new notifications</Text>  
 </MenuItem>  
 )}  
 <MenuDivider />  
 <MenuItem as={RouterLink} to="/notifications">  
 View All Notifications  
 </MenuItem>  
 </MenuList>  
 </Menu>  
  
 <Menu>  
 <MenuButton  
 as={Button}  
 rounded="full"  
 variant="link"  
 cursor="pointer"  
 minW={0}  
 >  
 <Avatar size="sm" name={user?.name} />  
 </MenuButton>  
 <MenuList>  
 <MenuItem>  
 <Text fontWeight="medium">{user?.name}</Text>  
 </MenuItem>  
 <MenuItem>  
 <Badge colorScheme="primary">{user?.role}</Badge>  
 </MenuItem>  
 <MenuDivider />  
 <MenuItem as={RouterLink} to={getDashboardLink()}>  
 Dashboard  
 </MenuItem>  
 {user?.role === 'donor' && (  
 <MenuItem as={RouterLink} to="/profile">  
 Profile  
 </MenuItem>  
 )}  
 <MenuItem onClick={handleLogout}>  
 Logout  
 </MenuItem>  
 </MenuList>  
 </Menu>  
 </HStack>  
 ) : (  
 <HStack spacing={4}>  
 <Button   
 as={RouterLink}   
 to="/login"   
 variant="ghost"  
 >  
 Login  
 </Button>  
 <Button   
 as={RouterLink}   
 to="/signup"  
 colorScheme="primary"  
 >  
 Sign Up  
 </Button>  
 </HStack>  
 )}  
 </Flex>  
 </Flex>  
  
 {isOpen ? (  
 <Box pb={4} display={{ md: 'none' }}>  
 <Stack as="nav" spacing={4}>  
 <Link as={RouterLink} to="/about">About</Link>  
 <Link as={RouterLink} to="/contact">Contact</Link>  
 {isAuthenticated ? (  
 <>  
 <Link as={RouterLink} to={getDashboardLink()}>Dashboard</Link>  
 <Link as={RouterLink} to="/find-donors">Find Donors</Link>  
 <Button onClick={handleLogout} variant="ghost" size="sm">  
 Logout  
 </Button>  
 </>  
 ) : (  
 <>  
 <Link as={RouterLink} to="/login">Login</Link>  
 <Link as={RouterLink} to="/signup">Sign Up</Link>  
 </>  
 )}  
 </Stack>  
 </Box>  
 ) : null}  
 </Box>  
 );  
}  
  
export default Navbar;

================================================================================

## 33. Authcontext

📂 File Path: C:\Users\Admin\React\sai\project(hemo\_connect)\src\context\AuthContext.jsx

import { createContext, useContext, useReducer, useEffect } from 'react';  
  
const AuthContext = createContext();  
  
const initialState = {  
 user: null,  
 isAuthenticated: false,  
 isLoading: true,  
};  
  
function authReducer(state, action) {  
 switch (action.type) {  
 case 'LOGIN\_SUCCESS':  
 return {  
 ...state,  
 user: action.payload,  
 isAuthenticated: true,  
 isLoading: false,  
 };  
 case 'LOGOUT':  
 return {  
 ...state,  
 user: null,  
 isAuthenticated: false,  
 isLoading: false,  
 };  
 case 'SET\_LOADING':  
 return {  
 ...state,  
 isLoading: action.payload,  
 };  
 case 'UPDATE\_USER':  
 return {  
 ...state,  
 user: { ...state.user, ...action.payload },  
 };  
 default:  
 return state;  
 }  
}  
  
export function AuthProvider({ children }) {  
 const [state, dispatch] = useReducer(authReducer, initialState);  
  
 useEffect(() => {  
 const token = localStorage.getItem('token');  
 if (token) {  
 // Verify token with backend  
 verifyToken(token);  
 } else {  
 dispatch({ type: 'SET\_LOADING', payload: false });  
 }  
 }, []);  
  
 const verifyToken = async (token) => {  
 try {  
 const response = await fetch('/api/auth/verify', {  
 headers: {  
 Authorization: `Bearer ${token}`,  
 },  
 });  
   
 if (response.ok) {  
 const user = await response.json();  
 dispatch({ type: 'LOGIN\_SUCCESS', payload: user });  
 } else {  
 localStorage.removeItem('token');  
 dispatch({ type: 'SET\_LOADING', payload: false });  
 }  
 } catch (error) {  
 console.error('Token verification failed:', error);  
 localStorage.removeItem('token');  
 dispatch({ type: 'SET\_LOADING', payload: false });  
 }  
 };  
  
 const login = async (email, password, rememberMe = false) => {  
 try {  
 const response = await fetch('/api/auth/login', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 },  
 body: JSON.stringify({ email, password, rememberMe }),  
 });  
  
 const data = await response.json();  
  
 if (response.ok) {  
 localStorage.setItem('token', data.token);  
 dispatch({ type: 'LOGIN\_SUCCESS', payload: data.user });  
 return { success: true, user: data.user };  
 } else {  
 return { success: false, message: data.message };  
 }  
 } catch (error) {  
 return { success: false, message: 'Network error occurred' };  
 }  
 };  
  
 const signup = async (userData) => {  
 try {  
 const response = await fetch('/api/auth/signup', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 },  
 body: JSON.stringify(userData),  
 });  
  
 const data = await response.json();  
  
 if (response.ok) {  
 localStorage.setItem('token', data.token);  
 dispatch({ type: 'LOGIN\_SUCCESS', payload: data.user });  
 return { success: true, user: data.user };  
 } else {  
 return { success: false, message: data.message };  
 }  
 } catch (error) {  
 return { success: false, message: 'Network error occurred' };  
 }  
 };  
  
 const logout = () => {  
 localStorage.removeItem('token');  
 dispatch({ type: 'LOGOUT' });  
 };  
  
 const updateUser = (userData) => {  
 dispatch({ type: 'UPDATE\_USER', payload: userData });  
 };  
  
 const value = {  
 ...state,  
 login,  
 signup,  
 logout,  
 updateUser,  
 };  
  
 return (  
 <AuthContext.Provider value={value}>  
 {children}  
 </AuthContext.Provider>  
 );  
}  
  
export const useAuth = () => {  
 const context = useContext(AuthContext);  
 if (!context) {  
 throw new Error('useAuth must be used within an AuthProvider');  
 }  
 return context;  
};

================================================================================

## 34. Socketcontext

📂 File Path: C:\Users\Admin\React\sai\project(hemo\_connect)\src\context\SocketContext.jsx

import { createContext, useContext, useEffect, useState } from 'react';  
import io from 'socket.io-client';  
import { useAuth } from './AuthContext';  
  
const SocketContext = createContext();  
  
export function SocketProvider({ children }) {  
 const [socket, setSocket] = useState(null);  
 const [notifications, setNotifications] = useState([]);  
 const { isAuthenticated, user } = useAuth();  
  
 useEffect(() => {  
 if (isAuthenticated && user) {  
 const newSocket = io('http://localhost:5000', {  
 auth: {  
 token: localStorage.getItem('token'),  
 userId: user.id,  
 },  
 });  
  
 newSocket.on('connect', () => {  
 console.log('Connected to socket server');  
 });  
  
 newSocket.on('notification', (notification) => {  
 setNotifications(prev => [notification, ...prev]);  
 });  
  
 newSocket.on('bloodRequest', (request) => {  
 if (user.role === 'donor') {  
 setNotifications(prev => [{  
 id: Date.now(),  
 type: 'blood\_request',  
 title: 'New Blood Request',  
 message: `${request.bloodGroup} blood needed at ${request.hospital}`,  
 data: request,  
 timestamp: new Date(),  
 }, ...prev]);  
 }  
 });  
  
 setSocket(newSocket);  
  
 return () => {  
 newSocket.disconnect();  
 };  
 }  
 }, [isAuthenticated, user]);  
  
 const value = {  
 socket,  
 notifications,  
 setNotifications,  
 };  
  
 return (  
 <SocketContext.Provider value={value}>  
 {children}  
 </SocketContext.Provider>  
 );  
}  
  
export const useSocket = () => {  
 const context = useContext(SocketContext);  
 if (!context) {  
 throw new Error('useSocket must be used within a SocketProvider');  
 }  
 return context;  
};

================================================================================

## 35. Aboutpage

📂 File Path: C:\Users\Admin\React\sai\project(hemo\_connect)\src\pages\About\AboutPage.jsx

import {  
 Container,  
 VStack,  
 Heading,  
 Text,  
 Box,  
 Grid,  
 GridItem,  
 Card,  
 CardBody,  
 Icon,  
 HStack,  
 Avatar,  
 Stack,  
} from '@chakra-ui/react';  
import { FiHeart, FiTarget, FiUsers, FiAward } from 'react-icons/fi';  
  
function AboutPage() {  
 const teamMembers = [  
 {  
 name: "Aditya Kumar Jha",  
 role: "Medical Director",  
 image: "",  
 },  
 {  
 name: "Arun Jelsinge",  
 role: "Technology Lead",  
 image: "",  
 },  
   
 ];  
  
 return (  
 <Container maxW="6xl" py={12}>  
 <VStack spacing={16}>  
 {/\* Hero Section \*/}  
 <VStack spacing={6} textAlign="center" maxW="4xl">  
 <Icon as={FiHeart} w={16} h={16} color="primary.500" />  
 <Heading size="2xl">About Hemo Connect</Heading>  
 <Text fontSize="lg" color="gray.600">  
 We're on a mission to make blood donation more accessible and efficient,   
 connecting donors with recipients when every second counts.  
 </Text>  
 </VStack>  
  
 {/\* Mission & Values \*/}  
 <Grid templateColumns={{ base: '1fr', md: 'repeat(3, 1fr)' }} gap={8}>  
 <GridItem>  
 <Card h="full" textAlign="center">  
 <CardBody p={8}>  
 <VStack spacing={4}>  
 <Icon as={FiTarget} w={12} h={12} color="primary.500" />  
 <Heading size="md">Our Mission</Heading>  
 <Text color="gray.600">  
 To save lives by creating a seamless connection between   
 blood donors and recipients through technology.  
 </Text>  
 </VStack>  
 </CardBody>  
 </Card>  
 </GridItem>  
  
 <GridItem>  
 <Card h="full" textAlign="center">  
 <CardBody p={8}>  
 <VStack spacing={4}>  
 <Icon as={FiUsers} w={12} h={12} color="primary.500" />  
 <Heading size="md">Community First</Heading>  
 <Text color="gray.600">  
 Building a supportive community where every donation   
 creates a ripple effect of hope and healing.  
 </Text>  
 </VStack>  
 </CardBody>  
 </Card>  
 </GridItem>  
  
 <GridItem>  
 <Card h="full" textAlign="center">  
 <CardBody p={8}>  
 <VStack spacing={4}>  
 <Icon as={FiAward} w={12} h={12} color="primary.500" />  
 <Heading size="md">Excellence</Heading>  
 <Text color="gray.600">  
 Committed to the highest standards of safety,   
 reliability, and user experience in healthcare technology.  
 </Text>  
 </VStack>  
 </CardBody>  
 </Card>  
 </GridItem>  
 </Grid>  
  
 {/\* Impact Stats \*/}  
 <Box bg="primary.50" p={12} rounded="2xl" w="full">  
 <VStack spacing={8} textAlign="center">  
 <Heading size="xl">Our Impact</Heading>  
 <Grid templateColumns={{ base: '1fr', md: 'repeat(3, 1fr)' }} gap={8}>  
 <VStack>  
 <Text fontSize="4xl" fontWeight="bold" color="primary.500">  
 2,547  
 </Text>  
 <Text color="gray.600">Lives Saved</Text>  
 </VStack>  
 <VStack>  
 <Text fontSize="4xl" fontWeight="bold" color="primary.500">  
 1,234  
 </Text>  
 <Text color="gray.600">Active Donors</Text>  
 </VStack>  
 <VStack>  
 <Text fontSize="4xl" fontWeight="bold" color="primary.500">  
 856  
 </Text>  
 <Text color="gray.600">Successful Matches</Text>  
 </VStack>  
 </Grid>  
 </VStack>  
 </Box>  
  
 {/\* Team Section \*/}  
 <VStack spacing={8}>  
 <VStack spacing={4} textAlign="center">  
 <Heading size="xl">Meet Our Team</Heading>  
 <Text fontSize="lg" color="gray.600" maxW="2xl">  
 Our dedicated team combines medical expertise with cutting-edge   
 technology to create a platform that saves lives.  
 </Text>  
 </VStack>  
  
 <Grid templateColumns={{ base: '1fr', md: 'repeat(3, 1fr)' }} gap={8}>  
 {teamMembers.map((member, index) => (  
 <Card key={index} textAlign="center">  
 <CardBody p={8}>  
 <VStack spacing={4}>  
 <Avatar  
 size="xl"  
 name={member.name}  
 src={member.image}  
 />  
 <Box>  
 <Heading size="md">{member.name}</Heading>  
 <Text color="gray.600">{member.role}</Text>  
 </Box>  
 </VStack>  
 </CardBody>  
 </Card>  
 ))}  
 </Grid>  
 </VStack>  
  
 {/\* Story Section \*/}  
 <Box maxW="4xl">  
 <VStack spacing={6} textAlign="center">  
 <Heading size="xl">Our Story</Heading>  
 <Text fontSize="lg" color="gray.600" lineHeight="tall">  
 Hemo Connect was born from a personal experience when our founder's   
 family member needed an emergency blood transfusion. The traditional   
 process was slow and stressful. We knew technology could do better.  
 </Text>  
 <Text fontSize="lg" color="gray.600" lineHeight="tall">  
 Today, we're proud to serve thousands of donors and recipients,   
 making blood donation as simple as a few taps on your phone.   
 Every donation facilitated through our platform represents hope,   
 healing, and the power of human connection.  
 </Text>  
 </VStack>  
 </Box>  
 </VStack>  
 </Container>  
 );  
}  
  
export default AboutPage;

================================================================================

## 36. Forgotpasswordpage

📂 File Path: C:\Users\Admin\React\sai\project(hemo\_connect)\src\pages\Auth\ForgotPasswordPage.jsx

import React, { useState } from 'react';  
import { Container, VStack, Heading, Text, FormControl, FormLabel, Input, Button, RadioGroup, Radio, Stack, useToast, Card, CardBody, Link } from '@chakra-ui/react';  
import { Link as RouterLink } from 'react-router-dom';  
  
function ForgotPasswordPage() {  
 const [method, setMethod] = useState('email');  
 const [value, setValue] = useState('');  
 const [otp, setOtp] = useState('');  
 const [newPassword, setNewPassword] = useState('');  
 const [otpSent, setOtpSent] = useState(false);  
 const [loading, setLoading] = useState(false);  
 const toast = useToast();  
  
 const sendOtp = async () => {  
 setLoading(true);  
 try {  
 const res = await fetch('/api/auth/send-otp', {  
 method: 'POST',  
 headers: { 'Content-Type': 'application/json' },  
 body: JSON.stringify({ method, value }),  
 });  
 const data = await res.json();  
 if (res.ok) {  
 setOtpSent(true);  
 toast({ title: 'OTP sent', status: 'success', duration: 3000 });  
 } else {  
 toast({ title: 'Failed to send OTP', description: data?.message, status: 'error', duration: 4000 });  
 }  
 } catch (e) {  
 toast({ title: 'Network error', status: 'error', duration: 4000 });  
 }  
 setLoading(false);  
 };  
  
 const resetPassword = async () => {  
 setLoading(true);  
 try {  
 const res = await fetch('/api/auth/reset-password', {  
 method: 'POST',  
 headers: { 'Content-Type': 'application/json' },  
 body: JSON.stringify({ method, value, otp, newPassword }),  
 });  
 const data = await res.json();  
 if (res.ok) {  
 toast({ title: 'Password reset successful', status: 'success', duration: 3000 });  
 setOtpSent(false);  
 setOtp('');  
 setNewPassword('');  
 } else {  
 toast({ title: 'Failed to reset', description: data?.message, status: 'error', duration: 4000 });  
 }  
 } catch (e) {  
 toast({ title: 'Network error', status: 'error', duration: 4000 });  
 }  
 setLoading(false);  
 };  
  
 return (  
 <Container maxW="md" py={12}>  
 <VStack spacing={8}>  
 <VStack spacing={2} textAlign="center">  
 <Heading size="lg">Forgot Password</Heading>  
 <Text color="gray.600">Reset your password using an OTP</Text>  
 </VStack>  
 <Card w="full">  
 <CardBody p={8}>  
 <VStack spacing={5} align="stretch">  
 <RadioGroup value={method} onChange={setMethod}>  
 <Stack direction="row">  
 <Radio value="email">Email</Radio>  
 <Radio value="phone">Phone</Radio>  
 </Stack>  
 </RadioGroup>  
 <FormControl>  
 <FormLabel>{method === 'email' ? 'Email' : 'Phone'}</FormLabel>  
 <Input placeholder={method === 'email' ? 'Enter your registered email' : 'Enter your registered phone'} value={value} onChange={e => setValue(e.target.value)} isDisabled={otpSent} />  
 </FormControl>  
 {!otpSent ? (  
 <Button colorScheme="primary" onClick={sendOtp} isLoading={loading}>Send OTP</Button>  
 ) : (  
 <>  
 <FormControl>  
 <FormLabel>OTP</FormLabel>  
 <Input placeholder="Enter OTP" value={otp} onChange={e => setOtp(e.target.value)} />  
 </FormControl>  
 <FormControl>  
 <FormLabel>New Password</FormLabel>  
 <Input type="password" placeholder="Enter new password" value={newPassword} onChange={e => setNewPassword(e.target.value)} />  
 </FormControl>  
 <Button colorScheme="primary" onClick={resetPassword} isLoading={loading}>Reset Password</Button>  
 </>  
 )}  
 <Text>  
 Remembered your password? <Link as={RouterLink} to="/login" color="primary.500">Back to login</Link>  
 </Text>  
 </VStack>  
 </CardBody>  
 </Card>  
 </VStack>  
 </Container>  
 );  
}  
  
export default ForgotPasswordPage;

================================================================================

## 37. Loginpage

📂 File Path: C:\Users\Admin\React\sai\project(hemo\_connect)\src\pages\Auth\LoginPage.jsx

import {  
 Box,  
 Button,  
 Checkbox,  
 Container,  
 FormControl,  
 FormLabel,  
 Heading,  
 Input,  
 Link,  
 Stack,  
 Text,  
 useToast,  
 VStack,  
 Card,  
 CardBody,  
 Icon,  
} from '@chakra-ui/react';  
import { FiHeart } from 'react-icons/fi';  
import { useState } from 'react';  
import { Link as RouterLink, useNavigate, useLocation, Navigate } from 'react-router-dom';  
import { useAuth } from '../../context/AuthContext';  
  
function LoginPage() {  
 const [formData, setFormData] = useState({  
 email: '',  
 password: '',  
 rememberMe: false,  
 });  
 const [isLoading, setIsLoading] = useState(false);  
 const { login, isAuthenticated } = useAuth();  
 const toast = useToast();  
 const navigate = useNavigate();  
 const location = useLocation();  
  
 const from = location.state?.from?.pathname || '/dashboard';  
  
 const handleChange = (e) => {  
 const { name, value, type, checked } = e.target;  
 setFormData(prev => ({  
 ...prev,  
 [name]: type === 'checkbox' ? checked : value,  
 }));  
 };  
  
 const handleSubmit = async (e) => {  
 e.preventDefault();  
 setIsLoading(true);  
  
 const result = await login(formData.email, formData.password, formData.rememberMe);  
  
 if (result.success) {  
 toast({  
 title: 'Login Successful',  
 description: `Welcome back, ${result.user.name}!`,  
 status: 'success',  
 duration: 3000,  
 });  
 navigate(from, { replace: true });  
 } else {  
 toast({  
 title: 'Login Failed',  
 description: result.message || 'Invalid email or password',  
 status: 'error',  
 duration: 5000,  
 });  
 }  
  
 setIsLoading(false);  
 };  
  
 if (isAuthenticated) {  
 return <Navigate to="/dashboard" replace />;  
 }  
  
 return (  
 <Container maxW="md" py={12}>  
 <VStack spacing={8}>  
 <VStack spacing={4} textAlign="center">  
 <Icon as={FiHeart} w={12} h={12} color="primary.500" />  
 <Heading size="lg">Welcome Back</Heading>  
 <Text color="gray.600">  
 Sign in to your Hemo Connect account  
 </Text>  
 </VStack>  
  
 <Card w="full">  
 <CardBody p={8}>  
 <form onSubmit={handleSubmit}>  
 <Stack spacing={6}>  
 <FormControl isRequired>  
 <FormLabel>Email</FormLabel>  
 <Input  
 name="email"  
 type="email"  
 value={formData.email}  
 onChange={handleChange}  
 placeholder="Enter your email"  
 />  
 </FormControl>  
  
 <FormControl isRequired>  
 <FormLabel>Password</FormLabel>  
 <Input  
 name="password"  
 type="password"  
 value={formData.password}  
 onChange={handleChange}  
 placeholder="Enter your password"  
 />  
 </FormControl>  
  
 <Stack spacing={6}>  
 <Stack direction="row" align="start" justify="space-between">  
 <Checkbox  
 name="rememberMe"  
 checked={formData.rememberMe}  
 onChange={handleChange}  
 >  
 Remember me  
 </Checkbox>  
 <Link as={RouterLink} to="/forgot-password" color="primary.500">  
 Forgot password?  
 </Link>  
 </Stack>  
  
 <Button  
 type="submit"  
 colorScheme="primary"  
 size="lg"  
 fontSize="md"  
 isLoading={isLoading}  
 loadingText="Signing in..."  
 >  
 Sign In  
 </Button>  
 </Stack>  
 </Stack>  
 </form>  
 </CardBody>  
 </Card>  
  
 <Text textAlign="center">  
 Don't have an account?{' '}  
 <Link as={RouterLink} to="/signup" color="primary.500" fontWeight="medium">  
 Sign up here  
 </Link>  
 </Text>  
 </VStack>  
 </Container>  
 );  
}  
  
export default LoginPage;

================================================================================

## 38. Profilepage

📂 File Path: C:\Users\Admin\React\sai\project(hemo\_connect)\src\pages\Auth\ProfilePage.jsx

import React, { useState } from 'react';  
import { Container, VStack, Heading, Text, Box, Badge, Avatar, Divider, HStack, Icon, Input, Button, useToast, FormControl, FormLabel } from '@chakra-ui/react';  
import { Modal, ModalOverlay, ModalContent, ModalHeader, ModalBody, ModalFooter, ModalCloseButton, useDisclosure, RadioGroup, Radio, Stack } from '@chakra-ui/react';  
import { FiUser, FiMail, FiCheckCircle, FiXCircle, FiShield, FiPhone, FiLock } from 'react-icons/fi';  
import { useAuth } from '../../context/AuthContext';  
  
function ProfilePage() {  
 const { user, updateUser } = useAuth();  
 const toast = useToast();  
 const { isOpen, onOpen, onClose } = useDisclosure();  
 const [otpMethod, setOtpMethod] = useState('email');  
 const [otpSent, setOtpSent] = useState(false);  
 const [otp, setOtp] = useState('');  
 const [resetEmailOrPhone, setResetEmailOrPhone] = useState(user?.email || '');  
 const [newResetPassword, setNewResetPassword] = useState('');  
 const [otpLoading, setOtpLoading] = useState(false);  
 const [phone, setPhone] = useState(user?.phone || '');  
 const [address, setAddress] = useState(user?.address || '');  
 const [emergencyContact, setEmergencyContact] = useState(user?.emergencyContact || '');  
 const [profilePic, setProfilePic] = useState(user?.profilePic || '');  
 const [profilePicFile, setProfilePicFile] = useState(null);  
 const [oldPassword, setOldPassword] = useState('');  
 const [newPassword, setNewPassword] = useState('');  
 const [loading, setLoading] = useState(false);  
  
 // Handle sending OTP  
 const handleSendOtp = async () => {  
 setOtpLoading(true);  
 try {  
 const response = await fetch('/api/auth/send-otp', {  
 method: 'POST',  
 headers: { 'Content-Type': 'application/json' },  
 body: JSON.stringify({ method: otpMethod, value: resetEmailOrPhone }),  
 });  
 const data = await response.json();  
 if (response.ok) {  
 setOtpSent(true);  
 toast({ title: 'OTP sent', status: 'success', duration: 3000 });  
 } else {  
 toast({ title: 'Failed to send OTP', description: data.message, status: 'error', duration: 3000 });  
 }  
 } catch (error) {  
 toast({ title: 'Network error', status: 'error', duration: 3000 });  
 }  
 setOtpLoading(false);  
 };  
  
 // Handle OTP verification and password reset  
 const handleVerifyOtpAndReset = async () => {  
 setOtpLoading(true);  
 try {  
 const response = await fetch('/api/auth/reset-password', {  
 method: 'POST',  
 headers: { 'Content-Type': 'application/json' },  
 body: JSON.stringify({ method: otpMethod, value: resetEmailOrPhone, otp, newPassword: newResetPassword }),  
 });  
 const data = await response.json();  
 if (response.ok) {  
 toast({ title: 'Password reset successful', status: 'success', duration: 3000 });  
 setOtpSent(false);  
 setOtp('');  
 setNewResetPassword('');  
 onClose();  
 } else {  
 toast({ title: 'Failed to reset password', description: data.message, status: 'error', duration: 3000 });  
 }  
 } catch (error) {  
 toast({ title: 'Network error', status: 'error', duration: 3000 });  
 }  
 setOtpLoading(false);  
 };  
  
 if (!user) {  
 return (  
 <Container maxW="md" py={20}>  
 <VStack spacing={8} textAlign="center">  
 <Heading size="lg">Unauthorized</Heading>  
 <Text color="gray.600">You must be logged in to view your profile.</Text>  
 </VStack>  
 </Container>  
 );  
 }  
  
 // Handle contact details update  
 const handleContactUpdate = async () => {  
 setLoading(true);  
 try {  
 const response = await fetch('/api/users/profile', {  
 method: 'PUT',  
 headers: {  
 'Content-Type': 'application/json',  
 Authorization: `Bearer ${localStorage.getItem('token')}`,  
 },  
 body: JSON.stringify({ phone, address, emergencyContact }),  
 });  
 const data = await response.json();  
 if (response.ok) {  
 updateUser(data);  
 toast({ title: 'Contact details updated', status: 'success', duration: 3000 });  
 } else {  
 toast({ title: 'Update failed', description: data.message, status: 'error', duration: 3000 });  
 }  
 } catch (error) {  
 toast({ title: 'Network error', status: 'error', duration: 3000 });  
 }  
 setLoading(false);  
 };  
  
 // Handle password update  
 const handlePasswordUpdate = async () => {  
 setLoading(true);  
 try {  
 const response = await fetch('/api/users/update-password', {  
 method: 'PUT',  
 headers: {  
 'Content-Type': 'application/json',  
 Authorization: `Bearer ${localStorage.getItem('token')}`,  
 },  
 body: JSON.stringify({ oldPassword, newPassword }),  
 });  
 const data = await response.json();  
 if (response.ok) {  
 toast({ title: 'Password updated', status: 'success', duration: 3000 });  
 setOldPassword('');  
 setNewPassword('');  
 } else {  
 toast({ title: 'Update failed', description: data.message, status: 'error', duration: 3000 });  
 }  
 } catch (error) {  
 toast({ title: 'Network error', status: 'error', duration: 3000 });  
 }  
 setLoading(false);  
 };  
  
 // Handle profile picture upload  
 const handleProfilePicUpload = async (e) => {  
 const file = e.target.files[0];  
 setProfilePicFile(file);  
 if (!file) return;  
 const formData = new FormData();  
 formData.append('profilePic', file);  
 setLoading(true);  
 try {  
 // You need a backend endpoint to handle file upload, here we assume /api/upload/profile-pic returns the URL  
 const uploadRes = await fetch('/api/upload/profile-pic', {  
 method: 'POST',  
 headers: {  
 Authorization: `Bearer ${localStorage.getItem('token')}`,  
 },  
 body: formData,  
 });  
 const uploadData = await uploadRes.json();  
 if (uploadRes.ok && uploadData.url) {  
 // Save URL to user profile  
 const response = await fetch('/api/users/profile-pic', {  
 method: 'PUT',  
 headers: {  
 'Content-Type': 'application/json',  
 Authorization: `Bearer ${localStorage.getItem('token')}`,  
 },  
 body: JSON.stringify({ profilePic: uploadData.url }),  
 });  
 const data = await response.json();  
 if (response.ok) {  
 setProfilePic(uploadData.url);  
 updateUser({ profilePic: uploadData.url });  
 toast({ title: 'Profile picture updated', status: 'success', duration: 3000 });  
 } else {  
 toast({ title: 'Update failed', description: data.message, status: 'error', duration: 3000 });  
 }  
 } else {  
 toast({ title: 'Upload failed', description: uploadData.message, status: 'error', duration: 3000 });  
 }  
 } catch (error) {  
 toast({ title: 'Network error', status: 'error', duration: 3000 });  
 }  
 setLoading(false);  
 };  
  
 return (  
 <Container maxW="md" py={20}>  
 <VStack spacing={8} textAlign="center">  
 <Heading size="lg" color="primary.500">Your Profile</Heading>  
 <Box  
 bg="white"  
 boxShadow="lg"  
 borderRadius="xl"  
 p={8}  
 w="100%"  
 >  
 <VStack spacing={6}>  
 <Avatar size="2xl" name={user.name} src={profilePic} mb={2} />  
 <FormControl>  
 <FormLabel>Update Profile Picture</FormLabel>  
 <Input type="file" accept="image/\*" onChange={handleProfilePicUpload} isDisabled={loading} />  
 </FormControl>  
 <Divider />  
 <HStack spacing={4} w="100%" justify="center">  
 <Icon as={FiUser} color="primary.500" boxSize={6} />  
 <Text fontWeight="bold" fontSize="lg">{user.name}</Text>  
 </HStack>  
 <HStack spacing={4} w="100%" justify="center">  
 <Icon as={FiMail} color="primary.500" boxSize={6} />  
 <Text fontSize="md">{user.email}</Text>  
 </HStack>  
 <HStack spacing={4} w="100%" justify="center">  
 <Icon as={FiShield} color="primary.500" boxSize={6} />  
 <Text fontSize="md">Role: <Badge colorScheme="primary">{user.role}</Badge></Text>  
 </HStack>  
 <HStack spacing={4} w="100%" justify="center">  
 {user.profileComplete ? (  
 <Icon as={FiCheckCircle} color="green.400" boxSize={6} />  
 ) : (  
 <Icon as={FiXCircle} color="red.400" boxSize={6} />  
 )}  
 <Text fontSize="md">  
 Profile Complete: {user.profileComplete ? 'Yes' : 'No'}  
 </Text>  
 </HStack>  
 <Divider />  
 <FormControl>  
 <FormLabel><Icon as={FiPhone} mr={2} />Phone</FormLabel>  
 <Input value={phone} onChange={e => setPhone(e.target.value)} placeholder="Phone number" isDisabled={loading} />  
 </FormControl>  
 <FormControl>  
 <FormLabel>Address</FormLabel>  
 <Input value={address} onChange={e => setAddress(e.target.value)} placeholder="Address" isDisabled={loading} />  
 </FormControl>  
 <FormControl>  
 <FormLabel>Emergency Contact</FormLabel>  
 <Input value={emergencyContact} onChange={e => setEmergencyContact(e.target.value)} placeholder="Emergency Contact" isDisabled={loading} />  
 </FormControl>  
 <Button colorScheme="primary" onClick={handleContactUpdate} isLoading={loading} w="100%">Update Contact Details</Button>  
 <Divider />  
 <FormControl>  
 <FormLabel><Icon as={FiLock} mr={2} />Change Password</FormLabel>  
 <Input type="password" value={oldPassword} onChange={e => setOldPassword(e.target.value)} placeholder="Old Password" isDisabled={loading} />  
 <Input type="password" value={newPassword} onChange={e => setNewPassword(e.target.value)} placeholder="New Password" mt={2} isDisabled={loading} />  
 </FormControl>  
 <Button colorScheme="primary" onClick={handlePasswordUpdate} isLoading={loading} w="100%">Update Password</Button>  
 <Button variant="link" colorScheme="primary" onClick={onOpen} w="100%" mt={2}>Forgot Password?</Button>  
  
 {/\* Modal for OTP-based password reset \*/}  
 <Modal isOpen={isOpen} onClose={onClose} isCentered>  
 <ModalOverlay />  
 <ModalContent>  
 <ModalHeader>Reset Password via OTP</ModalHeader>  
 <ModalCloseButton />  
 <ModalBody>  
 <RadioGroup value={otpMethod} onChange={setOtpMethod} mb={4}>  
 <Stack direction="row">  
 <Radio value="email">Email</Radio>  
 <Radio value="phone">Phone</Radio>  
 </Stack>  
 </RadioGroup>  
 <FormControl mb={3}>  
 <FormLabel>{otpMethod === 'email' ? 'Email' : 'Phone'}</FormLabel>  
 <Input value={resetEmailOrPhone} onChange={e => setResetEmailOrPhone(e.target.value)} placeholder={otpMethod === 'email' ? 'Enter your registered email' : 'Enter your registered phone'} isDisabled={otpSent} />  
 </FormControl>  
 {!otpSent ? (  
 <Button colorScheme="primary" onClick={handleSendOtp} isLoading={otpLoading} w="100%">Send OTP</Button>  
 ) : (  
 <>  
 <FormControl mb={3} mt={3}>  
 <FormLabel>Enter OTP</FormLabel>  
 <Input value={otp} onChange={e => setOtp(e.target.value)} placeholder="Enter OTP" />  
 </FormControl>  
 <FormControl mb={3}>  
 <FormLabel>New Password</FormLabel>  
 <Input type="password" value={newResetPassword} onChange={e => setNewResetPassword(e.target.value)} placeholder="New Password" />  
 </FormControl>  
 <Button colorScheme="primary" onClick={handleVerifyOtpAndReset} isLoading={otpLoading} w="100%">Reset Password</Button>  
 </>  
 )}  
 </ModalBody>  
 <ModalFooter>  
 <Button onClick={onClose}>Close</Button>  
 </ModalFooter>  
 </ModalContent>  
 </Modal>  
 </VStack>  
 </Box>  
 </VStack>  
 </Container>  
 );  
}  
  
export default ProfilePage;

================================================================================

## 39. Profilesetuppage

📂 File Path: C:\Users\Admin\React\sai\project(hemo\_connect)\src\pages\Auth\ProfileSetupPage.jsx

import {  
 Box,  
 Button,  
 Container,  
 FormControl,  
 FormLabel,  
 Heading,  
 Input,  
 Stack,  
 Text,  
 useToast,  
 VStack,  
 Card,  
 CardBody,  
 Switch,  
 Textarea,  
 Select,  
 HStack,  
 Progress,  
 Badge,  
} from '@chakra-ui/react';  
import { useState, useEffect } from 'react';  
import { useNavigate } from 'react-router-dom';  
import { useAuth } from '../../context/AuthContext';  
  
function ProfileSetupPage() {  
 const { user, updateUser } = useAuth();  
 const [formData, setFormData] = useState({  
 // Donor fields  
 medicalConditions: '',  
 medications: '',  
 lastDonationDate: '',  
 isAvailable: true,  
 // Recipient fields  
 hospital: '',  
 doctorName: '',  
 doctorContact: '',  
 urgencyLevel: 'medium',  
 // Common fields  
 phone: '',  
 address: '',  
 emergencyContact: '',  
 });  
 const [isLoading, setIsLoading] = useState(false);  
 const [step, setStep] = useState(1);  
 const toast = useToast();  
 const navigate = useNavigate();  
  
 useEffect(() => {  
 // Auto-save draft every 30 seconds  
 const interval = setInterval(() => {  
 saveDraft();  
 }, 30000);  
  
 return () => clearInterval(interval);  
 }, [formData]);  
  
 const handleChange = (e) => {  
 const { name, value, type, checked } = e.target;  
 setFormData(prev => ({  
 ...prev,  
 [name]: type === 'checkbox' ? checked : value,  
 }));  
 };  
  
 const saveDraft = async () => {  
 try {  
 await fetch('/api/users/save-draft', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 Authorization: `Bearer ${localStorage.getItem('token')}`,  
 },  
 body: JSON.stringify(formData),  
 });  
 } catch (error) {  
 console.error('Failed to save draft:', error);  
 }  
 };  
  
 const handleNext = () => {  
 if (step < 2) setStep(step + 1);  
 };  
  
 const handleBack = () => {  
 if (step > 1) setStep(step - 1);  
 };  
  
 const handleSubmit = async (e) => {  
 e.preventDefault();  
 setIsLoading(true);  
  
 try {  
 const response = await fetch('/api/users/complete-profile', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 Authorization: `Bearer ${localStorage.getItem('token')}`,  
 },  
 body: JSON.stringify(formData),  
 });  
  
 if (response.ok) {  
 const updatedUser = await response.json();  
 updateUser(updatedUser);  
 toast({  
 title: 'Profile Complete',  
 description: 'Your profile has been set up successfully!',  
 status: 'success',  
 duration: 3000,  
 });  
   
 const dashboardPath = user.role === 'donor' ? '/donor-dashboard' : '/recipient-dashboard';  
 navigate(dashboardPath);  
 } else {  
 const data = await response.json();  
 throw new Error(data.message);  
 }  
 } catch (error) {  
 toast({  
 title: 'Setup Failed',  
 description: error.message || 'Failed to complete profile setup',  
 status: 'error',  
 duration: 5000,  
 });  
 }  
  
 setIsLoading(false);  
 };  
  
 const progressValue = (step / 2) \* 100;  
  
 return (  
 <Container maxW="2xl" py={12}>  
 <VStack spacing={8}>  
 <VStack spacing={4} textAlign="center">  
 <Heading size="lg">Complete Your Profile</Heading>  
 <Text color="gray.600">  
 Let's set up your {user?.role} profile to get started  
 </Text>  
 <Progress value={progressValue} colorScheme="primary" w="full" />  
 </VStack>  
  
 <Card w="full">  
 <CardBody p={8}>  
 <form onSubmit={handleSubmit}>  
 <Stack spacing={6}>  
 {step === 1 && (  
 <>  
 <VStack align="start" spacing={4}>  
 <HStack>  
 <Heading size="md">Step 1: Basic Information</Heading>  
 <Badge colorScheme="primary">{user?.role}</Badge>  
 </HStack>  
 </VStack>  
  
 <FormControl isRequired>  
 <FormLabel>Phone Number</FormLabel>  
 <Input  
 name="phone"  
 value={formData.phone}  
 onChange={handleChange}  
 placeholder="Enter your phone number"  
 />  
 </FormControl>  
  
 <FormControl isRequired>  
 <FormLabel>Address</FormLabel>  
 <Textarea  
 name="address"  
 value={formData.address}  
 onChange={handleChange}  
 placeholder="Enter your complete address"  
 rows={3}  
 />  
 </FormControl>  
  
 <FormControl isRequired>  
 <FormLabel>Emergency Contact</FormLabel>  
 <Input  
 name="emergencyContact"  
 value={formData.emergencyContact}  
 onChange={handleChange}  
 placeholder="Emergency contact number"  
 />  
 </FormControl>  
  
 <Button onClick={handleNext} colorScheme="primary" size="lg">  
 Continue  
 </Button>  
 </>  
 )}  
  
 {step === 2 && user?.role === 'donor' && (  
 <>  
 <VStack align="start" spacing={4}>  
 <Heading size="md">Step 2: Donor Information</Heading>  
 </VStack>  
  
 <FormControl>  
 <FormLabel>Medical Conditions</FormLabel>  
 <Textarea  
 name="medicalConditions"  
 value={formData.medicalConditions}  
 onChange={handleChange}  
 placeholder="List any medical conditions (optional)"  
 rows={3}  
 />  
 </FormControl>  
  
 <FormControl>  
 <FormLabel>Current Medications</FormLabel>  
 <Textarea  
 name="medications"  
 value={formData.medications}  
 onChange={handleChange}  
 placeholder="List current medications (optional)"  
 rows={3}  
 />  
 </FormControl>  
  
 <FormControl>  
 <FormLabel>Last Donation Date</FormLabel>  
 <Input  
 name="lastDonationDate"  
 type="date"  
 value={formData.lastDonationDate}  
 onChange={handleChange}  
 />  
 </FormControl>  
  
 <FormControl display="flex" alignItems="center">  
 <FormLabel mb="0">Available for Donations</FormLabel>  
 <Switch  
 name="isAvailable"  
 isChecked={formData.isAvailable}  
 onChange={handleChange}  
 colorScheme="primary"  
 />  
 </FormControl>  
  
 <HStack spacing={4}>  
 <Button onClick={handleBack} variant="outline" size="lg">  
 Back  
 </Button>  
 <Button  
 type="submit"  
 colorScheme="primary"  
 size="lg"  
 isLoading={isLoading}  
 loadingText="Setting up..."  
 flex={1}  
 >  
 Complete Setup  
 </Button>  
 </HStack>  
 </>  
 )}  
  
 {step === 2 && user?.role === 'recipient' && (  
 <>  
 <VStack align="start" spacing={4}>  
 <Heading size="md">Step 2: Recipient Information</Heading>  
 </VStack>  
  
 <FormControl isRequired>  
 <FormLabel>Preferred Hospital</FormLabel>  
 <Input  
 name="hospital"  
 value={formData.hospital}  
 onChange={handleChange}  
 placeholder="Enter hospital name"  
 />  
 </FormControl>  
  
 <FormControl isRequired>  
 <FormLabel>Doctor Name</FormLabel>  
 <Input  
 name="doctorName"  
 value={formData.doctorName}  
 onChange={handleChange}  
 placeholder="Enter doctor's name"  
 />  
 </FormControl>  
  
 <FormControl isRequired>  
 <FormLabel>Doctor Contact</FormLabel>  
 <Input  
 name="doctorContact"  
 value={formData.doctorContact}  
 onChange={handleChange}  
 placeholder="Doctor's phone number"  
 />  
 </FormControl>  
  
 <FormControl isRequired>  
 <FormLabel>Default Urgency Level</FormLabel>  
 <Select  
 name="urgencyLevel"  
 value={formData.urgencyLevel}  
 onChange={handleChange}  
 >  
 <option value="low">Low - Planned Surgery</option>  
 <option value="medium">Medium - Within 48 hours</option>  
 <option value="high">High - Within 24 hours</option>  
 <option value="critical">Critical - Immediate</option>  
 </Select>  
 </FormControl>  
  
 <HStack spacing={4}>  
 <Button onClick={handleBack} variant="outline" size="lg">  
 Back  
 </Button>  
 <Button  
 type="submit"  
 colorScheme="primary"  
 size="lg"  
 isLoading={isLoading}  
 loadingText="Setting up..."  
 flex={1}  
 >  
 Complete Setup  
 </Button>  
 </HStack>  
 </>  
 )}  
 </Stack>  
 </form>  
 </CardBody>  
 </Card>  
 </VStack>  
 </Container>  
 );  
}  
  
export default ProfileSetupPage;

================================================================================

## 40. Signuppage

📂 File Path: C:\Users\Admin\React\sai\project(hemo\_connect)\src\pages\Auth\SignupPage.jsx

import {  
 Box,  
 Button,  
 Container,  
 FormControl,  
 FormLabel,  
 Heading,  
 Input,  
 Link,  
 Stack,  
 Text,  
 useToast,  
 VStack,  
 Card,  
 CardBody,  
 Icon,  
 Select,  
 RadioGroup,  
 HStack,  
 Radio,  
} from '@chakra-ui/react';  
import { FiHeart } from 'react-icons/fi';  
import { useState } from 'react';  
import { Link as RouterLink, useNavigate, Navigate } from 'react-router-dom';  
import { useAuth } from '../../context/AuthContext';  
  
function SignupPage() {  
 const [formData, setFormData] = useState({  
 name: '',  
 email: '',  
 password: '',  
 confirmPassword: '',  
 bloodGroup: '',  
 location: '',  
 role: 'donor',  
 });  
 const [isLoading, setIsLoading] = useState(false);  
 const { signup, isAuthenticated } = useAuth();  
 const toast = useToast();  
 const navigate = useNavigate();  
  
 const bloodGroups = ['A+', 'A-', 'B+', 'B-', 'AB+', 'AB-', 'O+', 'O-'];  
  
 const handleChange = (e) => {  
 const { name, value } = e.target;  
 setFormData(prev => ({  
 ...prev,  
 [name]: value,  
 }));  
 };  
  
 const handleSubmit = async (e) => {  
 e.preventDefault();  
   
 if (formData.password !== formData.confirmPassword) {  
 toast({  
 title: 'Password Mismatch',  
 description: 'Passwords do not match',  
 status: 'error',  
 duration: 3000,  
 });  
 return;  
 }  
  
 setIsLoading(true);  
  
 const result = await signup(formData);  
  
 if (result.success) {  
 toast({  
 title: 'Account Created',  
 description: 'Welcome to Hemo Connect! Please complete your profile.',  
 status: 'success',  
 duration: 3000,  
 });  
 navigate('/profile-setup');  
 } else {  
 toast({  
 title: 'Signup Failed',  
 description: result.message || 'Failed to create account',  
 status: 'error',  
 duration: 5000,  
 });  
 }  
  
 setIsLoading(false);  
 };  
  
 if (isAuthenticated) {  
 return <Navigate to="/profile-setup" replace />;  
 }  
  
 return (  
 <Container maxW="md" py={12}>  
 <VStack spacing={8}>  
 <VStack spacing={4} textAlign="center">  
 <Icon as={FiHeart} w={12} h={12} color="primary.500" />  
 <Heading size="lg">Join Hemo Connect</Heading>  
 <Text color="gray.600">  
 Create your account and start saving lives  
 </Text>  
 </VStack>  
  
 <Card w="full">  
 <CardBody p={8}>  
 <form onSubmit={handleSubmit}>  
 <Stack spacing={6}>  
 <FormControl isRequired>  
 <FormLabel>Full Name</FormLabel>  
 <Input  
 name="name"  
 value={formData.name}  
 onChange={handleChange}  
 placeholder="Enter your full name"  
 />  
 </FormControl>  
  
 <FormControl isRequired>  
 <FormLabel>Email</FormLabel>  
 <Input  
 name="email"  
 type="email"  
 value={formData.email}  
 onChange={handleChange}  
 placeholder="Enter your email"  
 />  
 </FormControl>  
  
  
 <FormControl isRequired>  
 <FormLabel>Password</FormLabel>  
 <Input  
 name="password"  
 type="password"  
 value={formData.password}  
 onChange={handleChange}  
 placeholder="Create a password"  
 autoComplete="new-password"  
 />  
 </FormControl>  
  
 <FormControl isRequired>  
 <FormLabel>Confirm Password</FormLabel>  
 <Input  
 name="confirmPassword"  
 type="password"  
 value={formData.confirmPassword}  
 onChange={handleChange}  
 placeholder="Confirm your password"  
 autoComplete="new-password"  
 />  
 </FormControl>  
  
 <FormControl isRequired>  
 <FormLabel>Blood Group</FormLabel>  
 <Select  
 name="bloodGroup"  
 value={formData.bloodGroup}  
 onChange={handleChange}  
 placeholder="Select your blood group"  
 >  
 {bloodGroups.map(group => (  
 <option key={group} value={group}>{group}</option>  
 ))}  
 </Select>  
 </FormControl>  
  
 <FormControl isRequired>  
 <FormLabel>Location</FormLabel>  
 <Input  
 name="location"  
 value={formData.location}  
 onChange={handleChange}  
 placeholder="Enter your city/location"  
 />  
 </FormControl>  
  
 <FormControl isRequired>  
 <FormLabel>I want to</FormLabel>  
 <RadioGroup  
 name="role"  
 value={formData.role}  
 onChange={(value) => setFormData(prev => ({ ...prev, role: value }))}  
 >  
 <HStack spacing={6}>  
 <Radio value="donor">Become a Donor</Radio>  
 <Radio value="recipient">Request Blood</Radio>  
 </HStack>  
 </RadioGroup>  
 </FormControl>  
  
 <Button  
 type="submit"  
 colorScheme="primary"  
 size="lg"  
 fontSize="md"  
 isLoading={isLoading}  
 loadingText="Creating account..."  
 >  
 Create Account  
 </Button>  
 </Stack>  
 </form>  
 </CardBody>  
 </Card>  
  
 <Text textAlign="center">  
 Already have an account?{' '}  
 <Link as={RouterLink} to="/login" color="primary.500" fontWeight="medium">  
 Sign in here  
 </Link>  
 </Text>  
 </VStack>  
 </Container>  
 );  
}  
  
export default SignupPage;

================================================================================

## 41. Contactpage

📂 File Path: C:\Users\Admin\React\sai\project(hemo\_connect)\src\pages\Contact\ContactPage.jsx

import {  
 Container,  
 Grid,  
 GridItem,  
 Card,  
 CardBody,  
 CardHeader,  
 Heading,  
 Text,  
 Button,  
 FormControl,  
 FormLabel,  
 Input,  
 Textarea,  
 VStack,  
 HStack,  
 Icon,  
 Box,  
 useToast,  
 Stack,  
} from '@chakra-ui/react';  
import { FiMail, FiPhone, FiMapPin, FiClock, FiHeart } from 'react-icons/fi';  
import { useState } from 'react';  
  
function ContactPage() {  
 const [formData, setFormData] = useState({  
 name: '',  
 email: '',  
 subject: '',  
 message: '',  
 });  
 const [isLoading, setIsLoading] = useState(false);  
 const toast = useToast();  
  
 const handleChange = (e) => {  
 const { name, value } = e.target;  
 setFormData(prev => ({  
 ...prev,  
 [name]: value,  
 }));  
 };  
  
 const handleSubmit = async (e) => {  
 e.preventDefault();  
 setIsLoading(true);  
  
 try {  
 const response = await fetch('/api/contact', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 },  
 body: JSON.stringify(formData),  
 });  
  
 if (response.ok) {  
 toast({  
 title: 'Message Sent',  
 description: 'Thank you for contacting us! We\'ll get back to you soon.',  
 status: 'success',  
 duration: 5000,  
 });  
 setFormData({  
 name: '',  
 email: '',  
 subject: '',  
 message: '',  
 });  
 } else {  
 throw new Error('Failed to send message');  
 }  
 } catch (error) {  
 toast({  
 title: 'Message Failed',  
 description: 'Failed to send your message. Please try again.',  
 status: 'error',  
 duration: 5000,  
 });  
 }  
  
 setIsLoading(false);  
 };  
  
 return (  
 <Container maxW="6xl" py={12}>  
 <VStack spacing={12}>  
 <VStack spacing={4} textAlign="center">  
 <Heading size="xl">Contact Us</Heading>  
 <Text fontSize="lg" color="gray.600" maxW="2xl">  
 Have questions or need support? We're here to help you make a difference   
 in saving lives through blood donation.  
 </Text>  
 </VStack>  
  
 <Grid templateColumns={{ base: '1fr', lg: '1fr 1fr' }} gap={12}>  
 {/\* Contact Form \*/}  
 <GridItem>  
 <Card>  
 <CardHeader>  
 <Heading size="md">Send us a Message</Heading>  
 </CardHeader>  
 <CardBody>  
 <form onSubmit={handleSubmit}>  
 <Stack spacing={6}>  
 <FormControl isRequired>  
 <FormLabel>Name</FormLabel>  
 <Input  
 name="name"  
 value={formData.name}  
 onChange={handleChange}  
 placeholder="Your full name"  
 />  
 </FormControl>  
  
 <FormControl isRequired>  
 <FormLabel>Email</FormLabel>  
 <Input  
 name="email"  
 type="email"  
 value={formData.email}  
 onChange={handleChange}  
 placeholder="Your email address"  
 />  
 </FormControl>  
  
 <FormControl isRequired>  
 <FormLabel>Subject</FormLabel>  
 <Input  
 name="subject"  
 value={formData.subject}  
 onChange={handleChange}  
 placeholder="Message subject"  
 />  
 </FormControl>  
  
 <FormControl isRequired>  
 <FormLabel>Message</FormLabel>  
 <Textarea  
 name="message"  
 value={formData.message}  
 onChange={handleChange}  
 placeholder="Your message"  
 rows={6}  
 />  
 </FormControl>  
  
 <Button  
 type="submit"  
 colorScheme="primary"  
 size="lg"  
 isLoading={isLoading}  
 loadingText="Sending..."  
 >  
 Send Message  
 </Button>  
 </Stack>  
 </form>  
 </CardBody>  
 </Card>  
 </GridItem>  
  
 {/\* Contact Information \*/}  
 <GridItem>  
 <VStack spacing={8}>  
 <Card w="full">  
 <CardBody p={8}>  
 <VStack spacing={6}>  
 <Heading size="md">Get in Touch</Heading>  
   
 <VStack spacing={4} align="stretch">  
 <HStack>  
 <Icon as={FiMail} color="primary.500" />  
 <Box>  
 <Text fontWeight="medium">Email</Text>  
 <Text color="gray.600" fontSize="sm">  
 support@hemoconnect.org  
 </Text>  
 </Box>  
 </HStack>  
  
 <HStack>  
 <Icon as={FiPhone} color="primary.500" />  
 <Box>  
 <Text fontWeight="medium">Emergency Helpline</Text>  
 <Text color="gray.600" fontSize="sm">  
 +1 (555) 123-BLOOD  
 </Text>  
 </Box>  
 </HStack>  
  
 <HStack>  
 <Icon as={FiMapPin} color="primary.500" />  
 <Box>  
 <Text fontWeight="medium">Address</Text>  
 <Text color="gray.600" fontSize="sm">  
 123 Health Street<br />  
 Medical District<br />  
 City, State 12345  
 </Text>  
 </Box>  
 </HStack>  
  
 <HStack>  
 <Icon as={FiClock} color="primary.500" />  
 <Box>  
 <Text fontWeight="medium">Support Hours</Text>  
 <Text color="gray.600" fontSize="sm">  
 24/7 for emergencies<br />  
 Mon-Fri 9 AM - 6 PM for general inquiries  
 </Text>  
 </Box>  
 </HStack>  
 </VStack>  
 </VStack>  
 </CardBody>  
 </Card>  
  
 <Card w="full" bg="primary.50">  
 <CardBody p={8}>  
 <VStack spacing={4} textAlign="center">  
 <Icon as={FiHeart} w={12} h={12} color="primary.500" />  
 <Heading size="md">Emergency Support</Heading>  
 <Text color="gray.700" fontSize="sm">  
 For life-threatening emergencies requiring immediate blood,   
 call 911 first, then contact our emergency helpline for   
 donor coordination support.  
 </Text>  
 </VStack>  
 </CardBody>  
 </Card>  
 </VStack>  
 </GridItem>  
 </Grid>  
 </VStack>  
 </Container>  
 );  
}  
  
export default ContactPage;

================================================================================

## 42. Admindashboard

📂 File Path: C:\Users\Admin\React\sai\project(hemo\_connect)\src\pages\Dashboard\AdminDashboard.jsx

import {  
 Container,  
 Grid,  
 GridItem,  
 Card,  
 CardBody,  
 CardHeader,  
 Heading,  
 Text,  
 Button,  
 VStack,  
 HStack,  
 Badge,  
 Box,  
 useToast,  
 Alert,  
 AlertIcon,  
 Stack,  
 Icon,  
 Stat,  
 StatLabel,  
 StatNumber,  
 StatHelpText,  
 Table,  
 Thead,  
 Tbody,  
 Tr,  
 Th,  
 Td,  
 TableContainer,  
} from '@chakra-ui/react';  
import { FiUsers, FiActivity, FiAlertTriangle, FiBarChart2 } from 'react-icons/fi';  
import { useState, useEffect } from 'react';  
  
function AdminDashboard() {  
 const [stats, setStats] = useState({  
 totalUsers: 0,  
 activeRequests: 0,  
 completedDonations: 0,  
 pendingReports: 0,  
 });  
 const [recentUsers, setRecentUsers] = useState([]);  
 const [flaggedRequests, setFlaggedRequests] = useState([]);  
 const [isLoading, setIsLoading] = useState(true);  
 const toast = useToast();  
  
 useEffect(() => {  
 fetchAdminData();  
 }, []);  
  
 const fetchAdminData = async () => {  
 try {  
 const [statsRes, usersRes, flaggedRes] = await Promise.all([  
 fetch('/api/admin/stats', {  
 headers: {  
 Authorization: `Bearer ${localStorage.getItem('token')}`,  
 },  
 }),  
 fetch('/api/admin/recent-users', {  
 headers: {  
 Authorization: `Bearer ${localStorage.getItem('token')}`,  
 },  
 }),  
 fetch('/api/admin/flagged-requests', {  
 headers: {  
 Authorization: `Bearer ${localStorage.getItem('token')}`,  
 },  
 }),  
 ]);  
  
 if (statsRes.ok) {  
 const statsData = await statsRes.json();  
 setStats(statsData);  
 }  
  
 if (usersRes.ok) {  
 const usersData = await usersRes.json();  
 setRecentUsers(usersData);  
 }  
  
 if (flaggedRes.ok) {  
 const flaggedData = await flaggedRes.json();  
 setFlaggedRequests(flaggedData);  
 }  
 } catch (error) {  
 console.error('Failed to fetch admin data:', error);  
 } finally {  
 setIsLoading(false);  
 }  
 };  
  
 const handleRequestAction = async (requestId, action) => {  
 try {  
 const response = await fetch(`/api/admin/requests/${requestId}/${action}`, {  
 method: 'POST',  
 headers: {  
 Authorization: `Bearer ${localStorage.getItem('token')}`,  
 },  
 });  
  
 if (response.ok) {  
 toast({  
 title: `Request ${action}ed`,  
 description: `The request has been ${action}ed successfully`,  
 status: 'success',  
 duration: 3000,  
 });  
 fetchAdminData();  
 }  
 } catch (error) {  
 toast({  
 title: 'Action Failed',  
 description: `Failed to ${action} the request`,  
 status: 'error',  
 duration: 3000,  
 });  
 }  
 };  
  
 if (isLoading) {  
 return (  
 <Container maxW="6xl" py={8}>  
 <Text>Loading admin dashboard...</Text>  
 </Container>  
 );  
 }  
  
 return (  
 <Container maxW="6xl" py={8}>  
 <VStack spacing={8} align="stretch">  
 <Box>  
 <Heading size="lg" mb={2}>  
 Admin Dashboard  
 </Heading>  
 <Text color="gray.600">  
 Manage the Hemo Connect platform and monitor activity  
 </Text>  
 </Box>  
  
 {/\* Stats Cards \*/}  
 <Grid templateColumns={{ base: '1fr', md: 'repeat(2, 1fr)', lg: 'repeat(4, 1fr)' }} gap={6}>  
 <Card>  
 <CardBody>  
 <Stat>  
 <StatLabel>  
 <HStack>  
 <Icon as={FiUsers} color="primary.500" />  
 <Text>Total Users</Text>  
 </HStack>  
 </StatLabel>  
 <StatNumber>{stats.totalUsers}</StatNumber>  
 <StatHelpText>Registered users</StatHelpText>  
 </Stat>  
 </CardBody>  
 </Card>  
  
 <Card>  
 <CardBody>  
 <Stat>  
 <StatLabel>  
 <HStack>  
 <Icon as={FiActivity} color="blue.500" />  
 <Text>Active Requests</Text>  
 </HStack>  
 </StatLabel>  
 <StatNumber>{stats.activeRequests}</StatNumber>  
 <StatHelpText>Pending blood requests</StatHelpText>  
 </Stat>  
 </CardBody>  
 </Card>  
  
 <Card>  
 <CardBody>  
 <Stat>  
 <StatLabel>  
 <HStack>  
 <Icon as={FiBarChart2} color="green.500" />  
 <Text>Donations</Text>  
 </HStack>  
 </StatLabel>  
 <StatNumber>{stats.completedDonations}</StatNumber>  
 <StatHelpText>This month</StatHelpText>  
 </Stat>  
 </CardBody>  
 </Card>  
  
 <Card>  
 <CardBody>  
 <Stat>  
 <StatLabel>  
 <HStack>  
 <Icon as={FiAlertTriangle} color="red.500" />  
 <Text>Reports</Text>  
 </HStack>  
 </StatLabel>  
 <StatNumber>{stats.pendingReports}</StatNumber>  
 <StatHelpText>Pending review</StatHelpText>  
 </Stat>  
 </CardBody>  
 </Card>  
 </Grid>  
  
 <Grid templateColumns={{ base: '1fr', lg: 'repeat(2, 1fr)' }} gap={8}>  
 {/\* Flagged Requests \*/}  
 <GridItem>  
 <Card h="full">  
 <CardHeader>  
 <HStack>  
 <Icon as={FiAlertTriangle} color="red.500" />  
 <Heading size="md">Flagged Requests</Heading>  
 <Badge colorScheme="red">{flaggedRequests.length}</Badge>  
 </HStack>  
 </CardHeader>  
 <CardBody>  
 {flaggedRequests.length > 0 ? (  
 <VStack spacing={4}>  
 {flaggedRequests.map(request => (  
 <Card key={request.\_id} w="full" variant="outline" borderColor="red.200">  
 <CardBody p={4}>  
 <Stack spacing={3}>  
 <HStack justify="space-between">  
 <Badge colorScheme="primary">{request.bloodGroup}</Badge>  
 <Badge colorScheme="red">Flagged</Badge>  
 </HStack>  
   
 <Text fontWeight="medium">{request.hospital}</Text>  
 <Text fontSize="sm" color="gray.600">  
 Requested by: {request.requesterName}  
 </Text>  
 <Text fontSize="sm" color="red.500">  
 Reason: {request.flagReason}  
 </Text>  
   
 <HStack spacing={2}>  
 <Button  
 size="sm"  
 colorScheme="green"  
 onClick={() => handleRequestAction(request.\_id, 'approve')}  
 >  
 Approve  
 </Button>  
 <Button  
 size="sm"  
 colorScheme="red"  
 variant="outline"  
 onClick={() => handleRequestAction(request.\_id, 'reject')}  
 >  
 Reject  
 </Button>  
 </HStack>  
 </Stack>  
 </CardBody>  
 </Card>  
 ))}  
 </VStack>  
 ) : (  
 <Alert status="success">  
 <AlertIcon />  
 No flagged requests to review.  
 </Alert>  
 )}  
 </CardBody>  
 </Card>  
 </GridItem>  
  
 {/\* Recent Users \*/}  
 <GridItem>  
 <Card h="full">  
 <CardHeader>  
 <HStack>  
 <Icon as={FiUsers} color="primary.500" />  
 <Heading size="md">Recent Users</Heading>  
 </HStack>  
 </CardHeader>  
 <CardBody>  
 {recentUsers.length > 0 ? (  
 <TableContainer>  
 <Table size="sm">  
 <Thead>  
 <Tr>  
 <Th>Name</Th>  
 <Th>Role</Th>  
 <Th>Blood Group</Th>  
 <Th>Joined</Th>  
 </Tr>  
 </Thead>  
 <Tbody>  
 {recentUsers.slice(0, 5).map(user => (  
 <Tr key={user.\_id}>  
 <Td>{user.name}</Td>  
 <Td>  
 <Badge colorScheme="primary">{user.role}</Badge>  
 </Td>  
 <Td>{user.bloodGroup}</Td>  
 <Td>{new Date(user.createdAt).toLocaleDateString()}</Td>  
 </Tr>  
 ))}  
 </Tbody>  
 </Table>  
 </TableContainer>  
 ) : (  
 <Alert status="info">  
 <AlertIcon />  
 No recent user registrations.  
 </Alert>  
 )}  
 </CardBody>  
 </Card>  
 </GridItem>  
 </Grid>  
  
 {/\* Quick Actions \*/}  
 <Card>  
 <CardHeader>  
 <Heading size="md">Quick Actions</Heading>  
 </CardHeader>  
 <CardBody>  
 <Grid templateColumns={{ base: '1fr', md: 'repeat(3, 1fr)' }} gap={4}>  
 <Button  
 colorScheme="primary"  
 variant="outline"  
 h="80px"  
 flexDirection="column"  
 gap={2}  
 >  
 <Icon as={FiUsers} />  
 <Text fontSize="sm">Manage Users</Text>  
 </Button>  
 <Button  
 colorScheme="blue"  
 variant="outline"  
 h="80px"  
 flexDirection="column"  
 gap={2}  
 >  
 <Icon as={FiBarChart2} />  
 <Text fontSize="sm">View Reports</Text>  
 </Button>  
 <Button  
 colorScheme="green"  
 variant="outline"  
 h="80px"  
 flexDirection="column"  
 gap={2}  
 >  
 <Icon as={FiActivity} />  
 <Text fontSize="sm">System Health</Text>  
 </Button>  
 </Grid>  
 </CardBody>  
 </Card>  
 </VStack>  
 </Container>  
 );  
}  
  
export default AdminDashboard;

================================================================================

## 43. Donordashboard

📂 File Path: C:\Users\Admin\React\sai\project(hemo\_connect)\src\pages\Dashboard\DonorDashboard.jsx

import {  
 Container,  
 Grid,  
 GridItem,  
 Card,  
 CardBody,  
 CardHeader,  
 Heading,  
 Text,  
 Button,  
 Switch,  
 FormControl,  
 FormLabel,  
 VStack,  
 HStack,  
 Badge,  
 Box,  
 useToast,  
 Alert,  
 AlertIcon,  
 Stack,  
 Icon,  
} from '@chakra-ui/react';  
import { FiHeart, FiMapPin, FiClock, FiUser } from 'react-icons/fi';  
import { useState, useEffect } from 'react';  
import { useAuth } from '../../context/AuthContext';  
import { useSocket } from '../../context/SocketContext';  
  
function DonorDashboard() {  
 const { user, updateUser } = useAuth();  
 const { notifications } = useSocket();  
 const [isAvailable, setIsAvailable] = useState(user?.isAvailable || false);  
 const [nearbyRequests, setNearbyRequests] = useState([]);  
 const [donationHistory, setDonationHistory] = useState([]);  
 const [isLoading, setIsLoading] = useState(true);  
 const toast = useToast();  
  
 useEffect(() => {  
 fetchDashboardData();  
 }, []);  
  
 const fetchDashboardData = async () => {  
 try {  
 const [requestsRes, historyRes] = await Promise.all([  
 fetch('/api/blood-requests/nearby', {  
 headers: {  
 Authorization: `Bearer ${localStorage.getItem('token')}`,  
 },  
 }),  
 fetch('/api/donor-blood/donation-history', {  
 headers: {  
 Authorization: `Bearer ${localStorage.getItem('token')}`,  
 },  
 }),  
 ]);  
  
 if (requestsRes.ok) {  
 const requests = await requestsRes.json();  
 setNearbyRequests(requests);  
 }  
  
 if (historyRes.ok) {  
 const history = await historyRes.json();  
 setDonationHistory(history);  
 }  
 } catch (error) {  
 console.error('Failed to fetch dashboard data:', error);  
 } finally {  
 setIsLoading(false);  
 }  
 };  
  
 const toggleAvailability = async () => {  
 try {  
 const response = await fetch('/api/users/toggle-availability', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 Authorization: `Bearer ${localStorage.getItem('token')}`,  
 },  
 body: JSON.stringify({ isAvailable: !isAvailable }),  
 });  
  
 if (response.ok) {  
 setIsAvailable(!isAvailable);  
 updateUser({ isAvailable: !isAvailable });  
 toast({  
 title: 'Availability Updated',  
 description: `You are now ${!isAvailable ? 'available' : 'unavailable'} for donations`,  
 status: 'success',  
 duration: 3000,  
 });  
 }  
 } catch (error) {  
 toast({  
 title: 'Update Failed',  
 description: 'Failed to update availability',  
 status: 'error',  
 duration: 3000,  
 });  
 }  
 };  
  
 const respondToRequest = async (requestId, response) => {  
 try {  
 const res = await fetch(`/api/requests/${requestId}/respond`, {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 Authorization: `Bearer ${localStorage.getItem('token')}`,  
 },  
 body: JSON.stringify({ response }),  
 });  
  
 if (res.ok) {  
 toast({  
 title: 'Response Sent',  
 description: `You have ${response}ed the blood request`,  
 status: 'success',  
 duration: 3000,  
 });  
 fetchDashboardData();  
 }  
 } catch (error) {  
 toast({  
 title: 'Response Failed',  
 description: 'Failed to send response',  
 status: 'error',  
 duration: 3000,  
 });  
 }  
 };  
  
 const getUrgencyColor = (urgency) => {  
 switch (urgency) {  
 case 'critical': return 'red';  
 case 'high': return 'orange';  
 case 'medium': return 'yellow';  
 case 'low': return 'green';  
 default: return 'gray';  
 }  
 };  
  
 if (isLoading) {  
 return (  
 <Container maxW="6xl" py={8}>  
 <Text>Loading dashboard...</Text>  
 </Container>  
 );  
 }  
  
 return (  
 <Container maxW="6xl" py={8}>  
 <VStack spacing={8} align="stretch">  
 <Box>  
 <Heading size="lg" mb={2}>  
 Welcome back, {user?.name}!  
 </Heading>  
 <Text color="gray.600">  
 Thank you for being a life-saver. Here's what's happening in your area.  
 </Text>  
 </Box>  
  
 {/\* Availability Toggle \*/}  
 <Card>  
 <CardBody>  
 <HStack justify="space-between">  
 <VStack align="start" spacing={1}>  
 <Text fontWeight="medium">Donation Availability</Text>  
 <Text fontSize="sm" color="gray.600">  
 Toggle to receive blood donation requests  
 </Text>  
 </VStack>  
 <FormControl display="flex" alignItems="center">  
 <Switch  
 id="availability"  
 isChecked={isAvailable}  
 onChange={toggleAvailability}  
 colorScheme="primary"  
 size="lg"  
 />  
 <FormLabel htmlFor="availability" ml={3} mb="0">  
 <Badge colorScheme={isAvailable ? 'green' : 'gray'}>  
 {isAvailable ? 'Available' : 'Unavailable'}  
 </Badge>  
 </FormLabel>  
 </FormControl>  
 </HStack>  
 </CardBody>  
 </Card>  
  
 <Grid templateColumns={{ base: '1fr', lg: 'repeat(2, 1fr)' }} gap={8}>  
 {/\* Active Blood Requests \*/}  
 <GridItem>  
 <Card h="full">  
 <CardHeader>  
 <HStack>  
 <Icon as={FiMapPin} color="primary.500" />  
 <Heading size="md">Nearby Blood Requests</Heading>  
 <Badge>{nearbyRequests.length}</Badge>  
 </HStack>  
 </CardHeader>  
 <CardBody>  
 {nearbyRequests.length > 0 ? (  
 <VStack spacing={4}>  
 {nearbyRequests.map(request => (  
 <Card key={request.\_id} w="full" variant="outline">  
 <CardBody p={4}>  
 <Stack spacing={3}>  
 <HStack justify="space-between">  
 <Badge  
 colorScheme="primary"  
 fontSize="sm"  
 px={2}  
 py={1}  
 >  
 {request.bloodGroup}  
 </Badge>  
 <Badge colorScheme={getUrgencyColor(request.urgencyLevel)}>  
 {request.urgencyLevel}  
 </Badge>  
 </HStack>  
   
 <Text fontWeight="medium">{request.hospital}</Text>  
 <Text fontSize="sm" color="gray.600">  
 {request.units} units needed  
 </Text>  
 <Text fontSize="sm" color="gray.500">  
 <Icon as={FiClock} mr={1} />  
 {new Date(request.createdAt).toLocaleDateString()}  
 </Text>  
   
 <HStack spacing={2}>  
 <Button  
 size="sm"  
 colorScheme="primary"  
 onClick={() => respondToRequest(request.\_id, 'accept')}  
 >  
 Accept  
 </Button>  
 <Button  
 size="sm"  
 variant="outline"  
 onClick={() => respondToRequest(request.\_id, 'decline')}  
 >  
 Decline  
 </Button>  
 </HStack>  
 </Stack>  
 </CardBody>  
 </Card>  
 ))}  
 </VStack>  
 ) : (  
 <Alert status="info">  
 <AlertIcon />  
 No blood requests in your area at the moment.  
 </Alert>  
 )}  
 </CardBody>  
 </Card>  
 </GridItem>  
  
 {/\* Recent Notifications \*/}  
 <GridItem>  
 <Card h="full">  
 <CardHeader>  
 <HStack>  
 <Icon as={FiUser} color="primary.500" />  
 <Heading size="md">Recent Notifications</Heading>  
 <Badge>{notifications.length}</Badge>  
 </HStack>  
 </CardHeader>  
 <CardBody>  
 {notifications.length > 0 ? (  
 <VStack spacing={4}>  
 {notifications.slice(0, 5).map(notification => (  
 <Card key={notification.id} w="full" variant="outline">  
 <CardBody p={4}>  
 <Stack spacing={2}>  
 <Text fontWeight="medium" fontSize="sm">  
 {notification.title}  
 </Text>  
 <Text fontSize="sm" color="gray.600">  
 {notification.message}  
 </Text>  
 <Text fontSize="xs" color="gray.500">  
 {new Date(notification.timestamp).toLocaleString()}  
 </Text>  
 </Stack>  
 </CardBody>  
 </Card>  
 ))}  
 </VStack>  
 ) : (  
 <Alert status="info">  
 <AlertIcon />  
 No new notifications.  
 </Alert>  
 )}  
 </CardBody>  
 </Card>  
 </GridItem>  
 </Grid>  
  
 {/\* Donation History \*/}  
 <Card>  
 <CardHeader>  
 <HStack>  
 <Icon as={FiHeart} color="primary.500" />  
 <Heading size="md">Recent Donations</Heading>  
 </HStack>  
 </CardHeader>  
 <CardBody>  
 {donationHistory.length > 0 ? (  
 <Grid templateColumns={{ base: '1fr', md: 'repeat(2, 1fr)', lg: 'repeat(3, 1fr)' }} gap={4}>  
 {donationHistory.slice(0, 6).map(donation => (  
 <Card key={donation.\_id} variant="outline">  
 <CardBody p={4}>  
 <Stack spacing={2}>  
 <HStack justify="space-between">  
 <Badge colorScheme="primary">{donation.bloodGroup}</Badge>  
 <Badge colorScheme="green">Completed</Badge>  
 </HStack>  
 <Text fontWeight="medium" fontSize="sm">  
 {donation.hospital}  
 </Text>  
 <Text fontSize="sm" color="gray.600">  
 {donation.units} units donated  
 </Text>  
 <Text fontSize="xs" color="gray.500">  
 {new Date(donation.donationDate).toLocaleDateString()}  
 </Text>  
 </Stack>  
 </CardBody>  
 </Card>  
 ))}  
 </Grid>  
 ) : (  
 <Alert status="info">  
 <AlertIcon />  
 No donation history available.  
 </Alert>  
 )}  
 </CardBody>  
 </Card>  
 </VStack>  
 </Container>  
 );  
}  
  
export default DonorDashboard;

================================================================================

## 44. Recipientdashboard

📂 File Path: C:\Users\Admin\React\sai\project(hemo\_connect)\src\pages\Dashboard\RecipientDashboard.jsx

import {  
 Container,  
 Grid,  
 GridItem,  
 Card,  
 CardBody,  
 CardHeader,  
 Heading,  
 Text,  
 Button,  
 VStack,  
 HStack,  
 Badge,  
 Box,  
 useToast,  
 Alert,  
 AlertIcon,  
 Stack,  
 Icon,  
 Progress,  
} from '@chakra-ui/react';  
import { FiPlus, FiMapPin, FiClock, FiList } from 'react-icons/fi';  
import { useState, useEffect } from 'react';  
import { Link as RouterLink } from 'react-router-dom';  
import { useAuth } from '../../context/AuthContext';  
  
function RecipientDashboard() {  
 const { user } = useAuth();  
 const [activeRequests, setActiveRequests] = useState([]);  
 const [requestHistory, setRequestHistory] = useState([]);  
 const [nearbyDonors, setNearbyDonors] = useState([]);  
 const [isLoading, setIsLoading] = useState(true);  
 const toast = useToast();  
  
 useEffect(() => {  
 fetchDashboardData();  
 }, []);  
  
 const fetchDashboardData = async () => {  
 try {  
 const [requestsRes, historyRes, donorsRes] = await Promise.all([  
 fetch('/api/requests/my-requests', {  
 headers: {  
 Authorization: `Bearer ${localStorage.getItem('token')}`,  
 },  
 }),  
 fetch('/api/requests/my-history', {  
 headers: {  
 Authorization: `Bearer ${localStorage.getItem('token')}`,  
 },  
 }),  
 fetch('/api/donors/nearby', {  
 headers: {  
 Authorization: `Bearer ${localStorage.getItem('token')}`,  
 },  
 }),  
 ]);  
  
 if (requestsRes.ok) {  
 const requests = await requestsRes.json();  
 setActiveRequests(requests);  
 }  
  
 if (historyRes.ok) {  
 const history = await historyRes.json();  
 setRequestHistory(history);  
 }  
  
 if (donorsRes.ok) {  
 const donors = await donorsRes.json();  
 setNearbyDonors(donors);  
 }  
 } catch (error) {  
 console.error('Failed to fetch dashboard data:', error);  
 } finally {  
 setIsLoading(false);  
 }  
 };  
  
 const cancelRequest = async (requestId) => {  
 try {  
 const response = await fetch(`/api/requests/${requestId}/cancel`, {  
 method: 'POST',  
 headers: {  
 Authorization: `Bearer ${localStorage.getItem('token')}`,  
 },  
 });  
  
 if (response.ok) {  
 toast({  
 title: 'Request Cancelled',  
 description: 'Your blood request has been cancelled',  
 status: 'success',  
 duration: 3000,  
 });  
 fetchDashboardData();  
 }  
 } catch (error) {  
 toast({  
 title: 'Cancellation Failed',  
 description: 'Failed to cancel the request',  
 status: 'error',  
 duration: 3000,  
 });  
 }  
 };  
  
 const getStatusColor = (status) => {  
 switch (status) {  
 case 'pending': return 'yellow';  
 case 'matched': return 'blue';  
 case 'fulfilled': return 'green';  
 case 'cancelled': return 'red';  
 default: return 'gray';  
 }  
 };  
  
 const getStatusProgress = (status) => {  
 switch (status) {  
 case 'pending': return 25;  
 case 'matched': return 75;  
 case 'fulfilled': return 100;  
 default: return 0;  
 }  
 };  
  
 if (isLoading) {  
 return (  
 <Container maxW="6xl" py={8}>  
 <Text>Loading dashboard...</Text>  
 </Container>  
 );  
 }  
  
 return (  
 <Container maxW="6xl" py={8}>  
 <VStack spacing={8} align="stretch">  
 <HStack justify="space-between" align="start">  
 <Box>  
 <Heading size="lg" mb={2}>  
 Welcome, {user?.name}  
 </Heading>  
 <Text color="gray.600">  
 Manage your blood requests and track their progress  
 </Text>  
 </Box>  
 <Button  
 as={RouterLink}  
 to="/request-blood"  
 colorScheme="primary"  
 leftIcon={<Icon as={FiPlus} />}  
 size="lg"  
 >  
 New Request  
 </Button>  
 </HStack>  
  
 <Grid templateColumns={{ base: '1fr', lg: 'repeat(2, 1fr)' }} gap={8}>  
 {/\* Active Requests \*/}  
 <GridItem>  
 <Card h="full">  
 <CardHeader>  
 <HStack>  
 <Icon as={FiClock} color="primary.500" />  
 <Heading size="md">Active Requests</Heading>  
 <Badge>{activeRequests.length}</Badge>  
 </HStack>  
 </CardHeader>  
 <CardBody>  
 {activeRequests.length > 0 ? (  
 <VStack spacing={4}>  
 {activeRequests.map(request => (  
 <Card key={request.\_id} w="full" variant="outline">  
 <CardBody p={4}>  
 <Stack spacing={3}>  
 <HStack justify="space-between">  
 <Badge  
 colorScheme="primary"  
 fontSize="sm"  
 px={2}  
 py={1}  
 >  
 {request.bloodGroup}  
 </Badge>  
 <Badge colorScheme={getStatusColor(request.status)}>  
 {request.status}  
 </Badge>  
 </HStack>  
   
 <Text fontWeight="medium">{request.hospital}</Text>  
 <Text fontSize="sm" color="gray.600">  
 {request.units} units needed  
 </Text>  
   
 <Box>  
 <HStack justify="space-between" mb={2}>  
 <Text fontSize="sm">Progress</Text>  
 <Text fontSize="sm">{request.status}</Text>  
 </HStack>  
 <Progress  
 value={getStatusProgress(request.status)}  
 colorScheme="primary"  
 size="sm"  
 rounded="full"  
 />  
 </Box>  
   
 {request.status === 'pending' && (  
 <HStack spacing={2}>  
 <Button  
 size="sm"  
 colorScheme="red"  
 variant="outline"  
 onClick={() => cancelRequest(request.\_id)}  
 >  
 Cancel Request  
 </Button>  
 </HStack>  
 )}  
 </Stack>  
 </CardBody>  
 </Card>  
 ))}  
 </VStack>  
 ) : (  
 <Alert status="info">  
 <AlertIcon />  
 No active requests. Click "New Request" to create one.  
 </Alert>  
 )}  
 </CardBody>  
 </Card>  
 </GridItem>  
  
 {/\* Nearby Donors \*/}  
 <GridItem>  
 <Card h="full">  
 <CardHeader>  
 <HStack>  
 <Icon as={FiMapPin} color="primary.500" />  
 <Heading size="md">Available Donors</Heading>  
 <Badge>{nearbyDonors.length}</Badge>  
 </HStack>  
 </CardHeader>  
 <CardBody>  
 {nearbyDonors.length > 0 ? (  
 <VStack spacing={4}>  
 {nearbyDonors.slice(0, 5).map(donor => (  
 <Card key={donor.\_id} w="full" variant="outline">  
 <CardBody p={4}>  
 <Stack spacing={2}>  
 <HStack justify="space-between">  
 <Text fontWeight="medium">{donor.name}</Text>  
 <Badge colorScheme="primary">{donor.bloodGroup}</Badge>  
 </HStack>  
 <Text fontSize="sm" color="gray.600">  
 <Icon as={FiMapPin} mr={1} />  
 {donor.location}  
 </Text>  
 <Text fontSize="sm" color="gray.500">  
 Last active: {new Date(donor.lastActive).toLocaleDateString()}  
 </Text>  
 <Button  
 size="sm"  
 colorScheme="primary"  
 variant="outline"  
 as={RouterLink}  
 to={`/contact?donorId=${donor.\_id}`}  
 >  
 Contact Donor  
 </Button>  
 </Stack>  
 </CardBody>  
 </Card>  
 ))}  
 </VStack>  
 ) : (  
 <Alert status="warning">  
 <AlertIcon />  
 No donors available in your area right now.  
 </Alert>  
 )}  
 </CardBody>  
 </Card>  
 </GridItem>  
 </Grid>  
  
 {/\* Request History \*/}  
 <Card>  
 <CardHeader>  
 <HStack>  
 <Icon as={FiList} color="primary.500" />  
 <Heading size="md">Request History</Heading>  
 </HStack>  
 </CardHeader>  
 <CardBody>  
 {requestHistory.length > 0 ? (  
 <Grid templateColumns={{ base: '1fr', md: 'repeat(2, 1fr)', lg: 'repeat(3, 1fr)' }} gap={4}>  
 {requestHistory.slice(0, 6).map(request => (  
 <Card key={request.\_id} variant="outline">  
 <CardBody p={4}>  
 <Stack spacing={2}>  
 <HStack justify="space-between">  
 <Badge colorScheme="primary">{request.bloodGroup}</Badge>  
 <Badge colorScheme={getStatusColor(request.status)}>  
 {request.status}  
 </Badge>  
 </HStack>  
 <Text fontWeight="medium" fontSize="sm">  
 {request.hospital}  
 </Text>  
 <Text fontSize="sm" color="gray.600">  
 {request.units} units  
 </Text>  
 <Text fontSize="xs" color="gray.500">  
 {new Date(request.createdAt).toLocaleDateString()}  
 </Text>  
 </Stack>  
 </CardBody>  
 </Card>  
 ))}  
 </Grid>  
 ) : (  
 <Alert status="info">  
 <AlertIcon />  
 No request history available.  
 </Alert>  
 )}  
 </CardBody>  
 </Card>  
 </VStack>  
 </Container>  
 );  
}  
  
export default RecipientDashboard;

================================================================================

## 45. Nodonorspage

📂 File Path: C:\Users\Admin\React\sai\project(hemo\_connect)\src\pages\Error\NoDonorsPage.jsx

import {  
 Container,  
 VStack,  
 Heading,  
 Text,  
 Button,  
 Box,  
 Icon,  
 HStack,  
 Alert,  
 AlertIcon,  
} from '@chakra-ui/react';  
import { FiUsers, FiPhone, FiHome } from 'react-icons/fi';  
import { Link as RouterLink } from 'react-router-dom';  
  
function NoDonorsPage() {  
 return (  
 <Container maxW="md" py={20}>  
 <VStack spacing={8} textAlign="center">  
 <Box>  
 <Icon as={FiUsers} w={20} h={20} color="gray.400" mb={4} />  
 <Heading size="lg" mb={4}>  
 No Donors Available  
 </Heading>  
 <Text color="gray.600" mb={8}>  
 We couldn't find any available donors in your area right now.   
 Don't worry, we have alternative options to help you.  
 </Text>  
 </Box>  
  
 <Alert status="info" rounded="lg">  
 <AlertIcon />  
 <Box>  
 <Text fontWeight="medium">Emergency Blood Banks</Text>  
 <Text fontSize="sm">  
 Contact these blood banks directly for immediate assistance:  
 </Text>  
 </Box>  
 </Alert>  
  
 <VStack spacing={4} w="full">  
 <Box p={4} borderWidth={1} borderColor="gray.200" rounded="lg" w="full">  
 <VStack spacing={2}>  
 <Text fontWeight="medium">Red Cross Blood Bank</Text>  
 <Text fontSize="sm" color="gray.600">24/7 Emergency Service</Text>  
 <Button  
 as="a"  
 href="tel:+1234567890"  
 size="sm"  
 colorScheme="primary"  
 leftIcon={<Icon as={FiPhone} />}  
 >  
 Call Now  
 </Button>  
 </VStack>  
 </Box>  
  
 <Box p={4} borderWidth={1} borderColor="gray.200" rounded="lg" w="full">  
 <VStack spacing={2}>  
 <Text fontWeight="medium">City General Hospital Blood Bank</Text>  
 <Text fontSize="sm" color="gray.600">Open 6 AM - 10 PM</Text>  
 <Button  
 as="a"  
 href="tel:+1234567891"  
 size="sm"  
 colorScheme="primary"  
 leftIcon={<Icon as={FiPhone} />}  
 >  
 Call Now  
 </Button>  
 </VStack>  
 </Box>  
 </VStack>  
   
 <HStack spacing={4}>  
 <Button  
 as={RouterLink}  
 to="/request-blood"  
 colorScheme="primary"  
 size="lg"  
 >  
 Try New Request  
 </Button>  
 <Button  
 as={RouterLink}  
 to="/"  
 variant="outline"  
 size="lg"  
 leftIcon={<Icon as={FiHome} />}  
 >  
 Go Home  
 </Button>  
 </HStack>  
 </VStack>  
 </Container>  
 );  
}  
  
export default NoDonorsPage;

================================================================================

## 46. Notfoundpage

📂 File Path: C:\Users\Admin\React\sai\project(hemo\_connect)\src\pages\Error\NotFoundPage.jsx

import {  
 Container,  
 VStack,  
 Heading,  
 Text,  
 Button,  
 Box,  
 Icon,  
} from '@chakra-ui/react';  
import { FiHome } from 'react-icons/fi';  
import { Link as RouterLink } from 'react-router-dom';  
  
function NotFoundPage() {  
 return (  
 <Container maxW="md" py={20}>  
 <VStack spacing={8} textAlign="center">  
 <Box>  
 <Text fontSize="6xl" fontWeight="bold" color="primary.500">  
 404  
 </Text>  
 <Heading size="lg" mb={4}>  
 Page Not Found  
 </Heading>  
 <Text color="gray.600" mb={8}>  
 The page you're looking for doesn't exist or has been moved.  
 </Text>  
 </Box>  
   
 <Button  
 as={RouterLink}  
 to="/"  
 colorScheme="primary"  
 size="lg"  
 leftIcon={<Icon as={FiHome} />}  
 >  
 Go Home  
 </Button>  
 </VStack>  
 </Container>  
 );  
}  
  
export default NotFoundPage;

================================================================================

## 47. Servererrorpage

📂 File Path: C:\Users\Admin\React\sai\project(hemo\_connect)\src\pages\Error\ServerErrorPage.jsx

import {  
 Container,  
 VStack,  
 Heading,  
 Text,  
 Button,  
 Box,  
 Icon,  
 HStack,  
} from '@chakra-ui/react';  
import { FiAlertTriangle, FiRefreshCw, FiHome } from 'react-icons/fi';  
import { Link as RouterLink } from 'react-router-dom';  
  
function ServerErrorPage() {  
 const handleRetry = () => {  
 window.location.reload();  
 };  
  
 return (  
 <Container maxW="md" py={20}>  
 <VStack spacing={8} textAlign="center">  
 <Box>  
 <Icon as={FiAlertTriangle} w={20} h={20} color="red.500" mb={4} />  
 <Heading size="lg" mb={4}>  
 Server Error  
 </Heading>  
 <Text color="gray.600" mb={8}>  
 Something went wrong on our end. Please try again or contact support if the problem persists.  
 </Text>  
 </Box>  
   
 <HStack spacing={4}>  
 <Button  
 onClick={handleRetry}  
 colorScheme="primary"  
 size="lg"  
 leftIcon={<Icon as={FiRefreshCw} />}  
 >  
 Try Again  
 </Button>  
 <Button  
 as={RouterLink}  
 to="/"  
 variant="outline"  
 size="lg"  
 leftIcon={<Icon as={FiHome} />}  
 >  
 Go Home  
 </Button>  
 </HStack>  
 </VStack>  
 </Container>  
 );  
}  
  
export default ServerErrorPage;

================================================================================

## 48. Unauthorizedpage

📂 File Path: C:\Users\Admin\React\sai\project(hemo\_connect)\src\pages\Error\UnauthorizedPage.jsx

import {  
 Container,  
 VStack,  
 Heading,  
 Text,  
 Button,  
 Box,  
 Icon,  
} from '@chakra-ui/react';  
import { FiLock, FiHome } from 'react-icons/fi';  
import { Link as RouterLink } from 'react-router-dom';  
  
function UnauthorizedPage() {  
 return (  
 <Container maxW="md" py={20}>  
 <VStack spacing={8} textAlign="center">  
 <Box>  
 <Icon as={FiLock} w={20} h={20} color="red.500" mb={4} />  
 <Heading size="lg" mb={4}>  
 Access Denied  
 </Heading>  
 <Text color="gray.600" mb={8}>  
 You don't have permission to access this page. Please check your account role or contact support.  
 </Text>  
 </Box>  
   
 <Button  
 as={RouterLink}  
 to="/"  
 colorScheme="primary"  
 size="lg"  
 leftIcon={<Icon as={FiHome} />}  
 >  
 Go Home  
 </Button>  
 </VStack>  
 </Container>  
 );  
}  
  
export default UnauthorizedPage;

================================================================================

## 49. Donorhistorypage

📂 File Path: C:\Users\Admin\React\sai\project(hemo\_connect)\src\pages\History\DonorHistoryPage.jsx

import {  
 Container,  
 VStack,  
 Heading,  
 Text,  
 Card,  
 CardBody,  
 Badge,  
 HStack,  
 Icon,  
 Box,  
 Alert,  
 AlertIcon,  
 Grid,  
 GridItem,  
 Stack,  
} from '@chakra-ui/react';  
import { FiHeart, FiCalendar, FiMapPin, FiAward } from 'react-icons/fi';  
import { useState, useEffect } from 'react';  
  
function DonorHistoryPage() {  
 const [donationHistory, setDonationHistory] = useState([]);  
 const [stats, setStats] = useState({  
 totalDonations: 0,  
 livessaved: 0,  
 nextEligibleDate: null,  
 });  
 const [isLoading, setIsLoading] = useState(true);  
  
 useEffect(() => {  
 fetchDonationHistory();  
 }, []);  
  
 const fetchDonationHistory = async () => {  
 try {  
 const response = await fetch('/api/donor-blood/donation-history', {  
 headers: {  
 Authorization: `Bearer ${localStorage.getItem('token')}`,  
 },  
 });  
  
 if (response.ok) {  
 const history = await response.json();  
 setDonationHistory(history);  
   
 // Calculate stats  
 setStats({  
 totalDonations: history.length,  
 livessaved: history.length \* 3, // Assuming each donation saves 3 lives  
 nextEligibleDate: calculateNextEligibleDate(history),  
 });  
 }  
 } catch (error) {  
 console.error('Failed to fetch donation history:', error);  
 } finally {  
 setIsLoading(false);  
 }  
 };  
  
 const calculateNextEligibleDate = (history) => {  
 if (history.length === 0) return new Date();  
   
 const lastDonation = history[0]; // Most recent  
 const nextDate = new Date(lastDonation.donationDate);  
 nextDate.setDate(nextDate.getDate() + 56); // 8 weeks between donations  
   
 return nextDate;  
 };  
  
 const isEligibleToDonate = () => {  
 return stats.nextEligibleDate && new Date() >= stats.nextEligibleDate;  
 };  
  
 if (isLoading) {  
 return (  
 <Container maxW="6xl" py={8}>  
 <Text>Loading donation history...</Text>  
 </Container>  
 );  
 }  
  
 return (  
 <Container maxW="6xl" py={8}>  
 <VStack spacing={8} align="stretch">  
 <Box>  
 <Heading size="lg" mb={2}>Donation History</Heading>  
 <Text color="gray.600">  
 Track your donation journey and see the impact you've made  
 </Text>  
 </Box>  
  
 {/\* Stats Cards \*/}  
 <Grid templateColumns={{ base: '1fr', md: 'repeat(3, 1fr)' }} gap={6}>  
 <Card>  
 <CardBody textAlign="center" p={8}>  
 <VStack spacing={4}>  
 <Icon as={FiHeart} w={12} h={12} color="primary.500" />  
 <Box>  
 <Text fontSize="3xl" fontWeight="bold" color="primary.500">  
 {stats.totalDonations}  
 </Text>  
 <Text color="gray.600">Total Donations</Text>  
 </Box>  
 </VStack>  
 </CardBody>  
 </Card>  
  
 <Card>  
 <CardBody textAlign="center" p={8}>  
 <VStack spacing={4}>  
 <Icon as={FiAward} w={12} h={12} color="green.500" />  
 <Box>  
 <Text fontSize="3xl" fontWeight="bold" color="green.500">  
 {stats.livessaved}  
 </Text>  
 <Text color="gray.600">Lives Potentially Saved</Text>  
 </Box>  
 </VStack>  
 </CardBody>  
 </Card>  
  
 <Card>  
 <CardBody textAlign="center" p={8}>  
 <VStack spacing={4}>  
 <Icon as={FiCalendar} w={12} h={12} color="blue.500" />  
 <Box>  
 <Text fontSize="lg" fontWeight="bold" color="blue.500">  
 {isEligibleToDonate() ? 'Eligible Now' : stats.nextEligibleDate?.toLocaleDateString()}  
 </Text>  
 <Text color="gray.600">Next Eligible Date</Text>  
 </Box>  
 </VStack>  
 </CardBody>  
 </Card>  
 </Grid>  
  
 {/\* Eligibility Status \*/}  
 <Alert   
 status={isEligibleToDonate() ? 'success' : 'info'}   
 variant="left-accent"  
 >  
 <AlertIcon />  
 <Box>  
 <Text fontWeight="medium">  
 {isEligibleToDonate()   
 ? 'You are eligible to donate blood!'   
 : 'You need to wait before your next donation'  
 }  
 </Text>  
 <Text fontSize="sm">  
 {isEligibleToDonate()  
 ? 'You can help save lives by responding to blood requests in your area.'  
 : `You can donate again on ${stats.nextEligibleDate?.toLocaleDateString()}`  
 }  
 </Text>  
 </Box>  
 </Alert>  
  
 {/\* Donation History \*/}  
 <Card>  
 <CardBody>  
 <VStack spacing={6} align="stretch">  
 <Heading size="md">Your Donations</Heading>  
   
 {donationHistory.length > 0 ? (  
 <Grid templateColumns={{ base: '1fr', md: 'repeat(2, 1fr)' }} gap={6}>  
 {donationHistory.map((donation) => (  
 <Card key={donation.\_id} variant="outline">  
 <CardBody>  
 <Stack spacing={4}>  
 <HStack justify="space-between">  
 <Badge colorScheme="primary" fontSize="md" px={3} py={1}>  
 {donation.bloodGroup}  
 </Badge>  
 <Badge colorScheme="green">Completed</Badge>  
 </HStack>  
   
 <Box>  
 <Text fontWeight="medium" mb={2}>  
 {donation.hospital}  
 </Text>  
 <VStack spacing={2} align="start">  
 <HStack>  
 <Icon as={FiHeart} color="primary.500" w={4} h={4} />  
 <Text fontSize="sm" color="gray.600">  
 {donation.units} units donated  
 </Text>  
 </HStack>  
 <HStack>  
 <Icon as={FiCalendar} color="blue.500" w={4} h={4} />  
 <Text fontSize="sm" color="gray.600">  
 {new Date(donation.donationDate).toLocaleDateString()}  
 </Text>  
 </HStack>  
 <HStack>  
 <Icon as={FiMapPin} color="green.500" w={4} h={4} />  
 <Text fontSize="sm" color="gray.600">  
 Saved approximately 3 lives  
 </Text>  
 </HStack>  
 </VStack>  
 </Box>  
 </Stack>  
 </CardBody>  
 </Card>  
 ))}  
 </Grid>  
 ) : (  
 <Alert status="info">  
 <AlertIcon />  
 <Box>  
 <Text fontWeight="medium">No donation history yet</Text>  
 <Text fontSize="sm">  
 Start your life-saving journey by accepting blood requests from your dashboard.  
 </Text>  
 </Box>  
 </Alert>  
 )}  
 </VStack>  
 </CardBody>  
 </Card>  
 </VStack>  
 </Container>  
 );  
}  
  
export default DonorHistoryPage;

================================================================================

## 50. Recipienthistorypage

📂 File Path: C:\Users\Admin\React\sai\project(hemo\_connect)\src\pages\History\RecipientHistoryPage.jsx

import {  
 Container,  
 VStack,  
 Heading,  
 Text,  
 Card,  
 CardBody,  
 Badge,  
 HStack,  
 Icon,  
 Box,  
 Alert,  
 AlertIcon,  
 Grid,  
 GridItem,  
 Stack,  
 Button,  
 Progress,  
} from '@chakra-ui/react';  
import { FiRefreshCw, FiCalendar, FiMapPin, FiClock } from 'react-icons/fi';  
import { useState, useEffect } from 'react';  
import { Link as RouterLink } from 'react-router-dom';  
  
function RecipientHistoryPage() {  
 const [requestHistory, setRequestHistory] = useState([]);  
 const [stats, setStats] = useState({  
 totalRequests: 0,  
 fulfilledRequests: 0,  
 averageResponseTime: 0,  
 });  
 const [isLoading, setIsLoading] = useState(true);  
  
 useEffect(() => {  
 fetchRequestHistory();  
 }, []);  
  
 const fetchRequestHistory = async () => {  
 try {  
 const response = await fetch('/api/requests/my-history', {  
 headers: {  
 Authorization: `Bearer ${localStorage.getItem('token')}`,  
 },  
 });  
  
 if (response.ok) {  
 const history = await response.json();  
 setRequestHistory(history);  
   
 // Calculate stats  
 const fulfilled = history.filter(req => req.status === 'fulfilled');  
 setStats({  
 totalRequests: history.length,  
 fulfilledRequests: fulfilled.length,  
 averageResponseTime: calculateAverageResponseTime(history),  
 });  
 }  
 } catch (error) {  
 console.error('Failed to fetch request history:', error);  
 } finally {  
 setIsLoading(false);  
 }  
 };  
  
 const calculateAverageResponseTime = (history) => {  
 const respondedRequests = history.filter(req =>   
 req.status === 'matched' || req.status === 'fulfilled'  
 );  
   
 if (respondedRequests.length === 0) return 0;  
   
 // Mock calculation - in real app, track actual response times  
 return 4.5; // hours  
 };  
  
 const getStatusColor = (status) => {  
 switch (status) {  
 case 'pending': return 'yellow';  
 case 'matched': return 'blue';  
 case 'fulfilled': return 'green';  
 case 'cancelled': return 'red';  
 default: return 'gray';  
 }  
 };  
  
 const getStatusProgress = (status) => {  
 switch (status) {  
 case 'pending': return 25;  
 case 'matched': return 75;  
 case 'fulfilled': return 100;  
 case 'cancelled': return 0;  
 default: return 0;  
 }  
 };  
  
 const getUrgencyColor = (urgency) => {  
 switch (urgency) {  
 case 'critical': return 'red';  
 case 'high': return 'orange';  
 case 'medium': return 'yellow';  
 case 'low': return 'green';  
 default: return 'gray';  
 }  
 };  
  
 if (isLoading) {  
 return (  
 <Container maxW="6xl" py={8}>  
 <Text>Loading request history...</Text>  
 </Container>  
 );  
 }  
  
 return (  
 <Container maxW="6xl" py={8}>  
 <VStack spacing={8} align="stretch">  
 <HStack justify="space-between" align="start">  
 <Box>  
 <Heading size="lg" mb={2}>Request History</Heading>  
 <Text color="gray.600">  
 View all your blood requests and their outcomes  
 </Text>  
 </Box>  
 <Button  
 as={RouterLink}  
 to="/request-blood"  
 colorScheme="primary"  
 leftIcon={<Icon as={FiRefreshCw} />}  
 >  
 New Request  
 </Button>  
 </HStack>  
  
 {/\* Stats Cards \*/}  
 <Grid templateColumns={{ base: '1fr', md: 'repeat(3, 1fr)' }} gap={6}>  
 <Card>  
 <CardBody textAlign="center" p={8}>  
 <VStack spacing={4}>  
 <Icon as={FiRefreshCw} w={12} h={12} color="blue.500" />  
 <Box>  
 <Text fontSize="3xl" fontWeight="bold" color="blue.500">  
 {stats.totalRequests}  
 </Text>  
 <Text color="gray.600">Total Requests</Text>  
 </Box>  
 </VStack>  
 </CardBody>  
 </Card>  
  
 <Card>  
 <CardBody textAlign="center" p={8}>  
 <VStack spacing={4}>  
 <Icon as={FiCalendar} w={12} h={12} color="green.500" />  
 <Box>  
 <Text fontSize="3xl" fontWeight="bold" color="green.500">  
 {stats.fulfilledRequests}  
 </Text>  
 <Text color="gray.600">Fulfilled Requests</Text>  
 </Box>  
 </VStack>  
 </CardBody>  
 </Card>  
  
 <Card>  
 <CardBody textAlign="center" p={8}>  
 <VStack spacing={4}>  
 <Icon as={FiClock} w={12} h={12} color="orange.500" />  
 <Box>  
 <Text fontSize="3xl" fontWeight="bold" color="orange.500">  
 {stats.averageResponseTime}h  
 </Text>  
 <Text color="gray.600">Avg. Response Time</Text>  
 </Box>  
 </VStack>  
 </CardBody>  
 </Card>  
 </Grid>  
  
 {/\* Request History \*/}  
 <Card>  
 <CardBody>  
 <VStack spacing={6} align="stretch">  
 <Heading size="md">Your Blood Requests</Heading>  
   
 {requestHistory.length > 0 ? (  
 <VStack spacing={4}>  
 {requestHistory.map((request) => (  
 <Card key={request.\_id} variant="outline">  
 <CardBody>  
 <Stack spacing={4}>  
 <HStack justify="space-between" wrap="wrap">  
 <HStack spacing={3}>  
 <Badge colorScheme="primary" fontSize="md" px={3} py={1}>  
 {request.bloodGroup}  
 </Badge>  
 <Badge colorScheme={getUrgencyColor(request.urgencyLevel)}>  
 {request.urgencyLevel}  
 </Badge>  
 </HStack>  
 <Badge colorScheme={getStatusColor(request.status)} fontSize="md">  
 {request.status}  
 </Badge>  
 </HStack>  
   
 <Grid templateColumns={{ base: '1fr', md: '2fr 1fr' }} gap={6}>  
 <Box>  
 <VStack spacing={3} align="start">  
 <Box>  
 <Text fontWeight="medium" fontSize="lg" mb={1}>  
 {request.hospital}  
 </Text>  
 <Text color="gray.600">  
 {request.units} units requested  
 </Text>  
 </Box>  
   
 <HStack>  
 <Icon as={FiCalendar} color="gray.500" w={4} h={4} />  
 <Text fontSize="sm" color="gray.600">  
 Requested: {new Date(request.createdAt).toLocaleDateString()}  
 </Text>  
 </HStack>  
   
 <HStack>  
 <Icon as={FiMapPin} color="gray.500" w={4} h={4} />  
 <Text fontSize="sm" color="gray.600">  
 Contact: {request.contactNumber}  
 </Text>  
 </HStack>  
  
 {request.additionalNotes && (  
 <Box>  
 <Text fontSize="sm" fontWeight="medium" color="gray.700">  
 Notes:  
 </Text>  
 <Text fontSize="sm" color="gray.600">  
 {request.additionalNotes}  
 </Text>  
 </Box>  
 )}  
 </VStack>  
 </Box>  
  
 <Box>  
 <VStack spacing={4} align="stretch">  
 <Box>  
 <HStack justify="space-between" mb={2}>  
 <Text fontSize="sm" fontWeight="medium">  
 Progress  
 </Text>  
 <Text fontSize="sm" color="gray.600">  
 {request.status}  
 </Text>  
 </HStack>  
 <Progress  
 value={getStatusProgress(request.status)}  
 colorScheme={getStatusColor(request.status)}  
 size="sm"  
 rounded="full"  
 />  
 </Box>  
  
 {request.status === 'fulfilled' && (  
 <Box p={3} bg="green.50" rounded="md">  
 <Text fontSize="sm" color="green.700" textAlign="center">  
 ✓ Request fulfilled successfully  
 </Text>  
 </Box>  
 )}  
  
 {request.status === 'cancelled' && (  
 <Box p={3} bg="red.50" rounded="md">  
 <Text fontSize="sm" color="red.700" textAlign="center">  
 Request was cancelled  
 </Text>  
 </Box>  
 )}  
  
 {(request.status === 'fulfilled' || request.status === 'cancelled') && (  
 <Button  
 size="sm"  
 colorScheme="primary"  
 variant="outline"  
 leftIcon={<Icon as={FiRefreshCw} />}  
 as={RouterLink}  
 to="/request-blood"  
 >  
 Repeat Request  
 </Button>  
 )}  
 </VStack>  
 </Box>  
 </Grid>  
 </Stack>  
 </CardBody>  
 </Card>  
 ))}  
 </VStack>  
 ) : (  
 <Alert status="info">  
 <AlertIcon />  
 <Box>  
 <Text fontWeight="medium">No request history</Text>  
 <Text fontSize="sm">  
 You haven't made any blood requests yet. Click "New Request" to get started.  
 </Text>  
 </Box>  
 </Alert>  
 )}  
 </VStack>  
 </CardBody>  
 </Card>  
 </VStack>  
 </Container>  
 );  
}  
  
export default RecipientHistoryPage;

================================================================================

## 51. Homepage

📂 File Path: C:\Users\Admin\React\sai\project(hemo\_connect)\src\pages\Home\HomePage.jsx

import {  
 Box,  
 Button,  
 Container,  
 Heading,  
 Text,  
 VStack,  
 HStack,  
 Grid,  
 GridItem,  
 Card,  
 CardBody,  
 Icon,  
 Badge,  
 Flex,  
} from '@chakra-ui/react';  
import { FiHeart, FiUsers, FiMapPin, FiClock } from 'react-icons/fi';  
import { Link as RouterLink, Navigate } from 'react-router-dom';  
import { useAuth } from '../../context/AuthContext';  
import { useState, useEffect } from 'react';  
  
function HomePage() {  
 const { isAuthenticated, user } = useAuth();  
 const [activeRequests, setActiveRequests] = useState([]);  
  
 useEffect(() => {  
 // Fetch active blood requests for the ticker  
 fetchActiveRequests();  
 }, []);  
  
 const fetchActiveRequests = async () => {  
 try {  
 const response = await fetch('/api/blood-requests/active');  
 if (response.ok) {  
 const data = await response.json();  
 setActiveRequests(data);  
 }  
 } catch (error) {  
 console.error('Failed to fetch active requests:', error);  
 }  
 };  
  
 if (isAuthenticated && user?.profileComplete) {  
 const dashboardPath = user.role === 'donor' ? '/donor-dashboard' :   
 user.role === 'recipient' ? '/recipient-dashboard' :   
 user.role === 'admin' ? '/admin-dashboard' : '/dashboard';  
 return <Navigate to={dashboardPath} replace />;  
 }  
  
 return (  
 <Box>  
 {/\* Hero Section \*/}  
 <Box bg="gradient-to-r from-primary.500 to-primary.600" color="red" py={20}>  
 <Container maxW="6xl">  
 <VStack spacing={8} textAlign="center">  
 <Icon as={FiHeart} w={16} h={16} />  
 <Heading size="2xl" fontWeight="bold">  
 Save Lives Through Blood Donation  
 </Heading>  
 <Text fontSize="xl" maxW="3xl">  
 Connect donors with recipients in real-time. Join our community of   
 life-savers and help make blood donation accessible to everyone.  
 </Text>  
   
 <HStack spacing={4} pt={4}>  
 <Button  
 as={RouterLink}  
 to="/find-donors"  
 size="lg"  
 bg="white"  
 color="primary.500"  
 \_hover={{ transform: 'translateY(-2px)', boxShadow: 'lg' }}  
 >  
 Find Donor  
 </Button>  
 <Button  
 as={RouterLink}  
 to="/signup"  
 size="lg"  
 bg="white"  
 color="primary.500"  
 borderColor="white"  
 variant="solid"  
 \_hover={{ transform: 'translateY(-2px)', boxShadow: 'lg' }}  
 >  
 Become Donor  
 </Button>  
 </HStack>  
 </VStack>  
 </Container>  
 </Box>  
  
 {/\* Real-time Blood Request Ticker \*/}  
 <Box bg="gray.50" py={4} borderY="1px" borderColor="gray.200">  
 <Container maxW="6xl">  
 <Flex align="center" gap={4}>  
 <Badge colorScheme="red" fontSize="sm" px={3} py={1} rounded="full">  
 URGENT  
 </Badge>  
 <Box flex={1} overflow="hidden">  
 <Text fontSize="sm" fontWeight="medium" color="gray.700">  
 {activeRequests.length > 0 ? (  
 `${activeRequests[0].bloodGroup} blood needed at ${activeRequests[0].hospital} - ${activeRequests[0].units} units required`  
 ) : (  
 'No urgent requests at the moment'  
 )}  
 </Text>  
 </Box>  
 <Button  
 as={RouterLink}  
 to="/find-donors"  
 size="sm"  
 colorScheme="primary"  
 variant="outline"  
 >  
 Help Now  
 </Button>  
 </Flex>  
 </Container>  
 </Box>  
  
 {/\* Features Section \*/}  
 <Container maxW="6xl" py={20}>  
 <VStack spacing={16}>  
 <VStack spacing={4} textAlign="center">  
 <Heading size="xl">How Hemo Connect Works</Heading>  
 <Text fontSize="lg" color="gray.600" maxW="2xl">  
 Our platform connects blood donors with recipients seamlessly,   
 making life-saving donations more accessible than ever.  
 </Text>  
 </VStack>  
  
 <Grid templateColumns={{ base: '1fr', md: 'repeat(3, 1fr)' }} gap={8}>  
 <GridItem>  
 <Card h="full" textAlign="center" p={6}>  
 <CardBody>  
 <VStack spacing={4}>  
 <Icon as={FiUsers} w={12} h={12} color="primary.500" />  
 <Heading size="md">Register as Donor</Heading>  
 <Text color="gray.600">  
 Sign up and complete your donor profile with medical   
 information and availability preferences.  
 </Text>  
 </VStack>  
 </CardBody>  
 </Card>  
 </GridItem>  
   
 <GridItem>  
 <Card h="full" textAlign="center" p={6}>  
 <CardBody>  
 <VStack spacing={4}>  
 <Icon as={FiMapPin} w={12} h={12} color="primary.500" />  
 <Heading size="md">Get Matched</Heading>  
 <Text color="gray.600">  
 Receive notifications when someone nearby needs your   
 blood type for emergency or planned donations.  
 </Text>  
 </VStack>  
 </CardBody>  
 </Card>  
 </GridItem>  
   
 <GridItem>  
 <Card h="full" textAlign="center" p={6}>  
 <CardBody>  
 <VStack spacing={4}>  
 <Icon as={FiHeart} w={12} h={12} color="primary.500" />  
 <Heading size="md">Save Lives</Heading>  
 <Text color="gray.600">  
 Connect with recipients, coordinate donation logistics,   
 and help save lives in your community.  
 </Text>  
 </VStack>  
 </CardBody>  
 </Card>  
 </GridItem>  
 </Grid>  
 </VStack>  
 </Container>  
  
 {/\* Stats Section \*/}  
 <Box bg="primary.500" color="white" py={16}>  
 <Container maxW="6xl">  
 <Grid templateColumns={{ base: '1fr', md: 'repeat(4, 1fr)' }} gap={8}>  
 <VStack>  
 <Text fontSize="4xl" fontWeight="bold">2,547</Text>  
 <Text>Lives Saved</Text>  
 </VStack>  
 <VStack>  
 <Text fontSize="4xl" fontWeight="bold">1,234</Text>  
 <Text>Active Donors</Text>  
 </VStack>  
 <VStack>  
 <Text fontSize="4xl" fontWeight="bold">856</Text>  
 <Text>Successful Matches</Text>  
 </VStack>  
 <VStack>  
 <Text fontSize="4xl" fontWeight="bold">24/7</Text>  
 <Text>Emergency Support</Text>  
 </VStack>  
 </Grid>  
 </Container>  
 </Box>  
  
 {/\* Call to Action \*/}  
 <Container maxW="6xl" py={20}>  
 <VStack spacing={8} textAlign="center">  
 <Heading size="xl">Ready to Make a Difference?</Heading>  
 <Text fontSize="lg" color="gray.600" maxW="2xl">  
 Join thousands of donors who are already helping save lives.   
 Your donation can make the difference between life and death.  
 </Text>  
 <HStack spacing={4}>  
 <Button  
 as={RouterLink}  
 to="/signup"  
 size="lg"  
 colorScheme="primary"  
 >  
 Join as Donor  
 </Button>  
 <Button  
 as={RouterLink}  
 to="/request-blood"  
 size="lg"  
 variant="outline"  
 colorScheme="primary"  
 >  
 Request Blood  
 </Button>  
 </HStack>  
 </VStack>  
 </Container>  
 </Box>  
 );  
}  
  
export default HomePage;

================================================================================

## 52. Donormatchingpage

📂 File Path: C:\Users\Admin\React\sai\project(hemo\_connect)\src\pages\Matching\DonorMatchingPage.jsx

import {  
 Container,  
 VStack,  
 Heading,  
 Text,  
 Card,  
 CardBody,  
 Badge,  
 HStack,  
 Icon,  
 Box,  
 Alert,  
 AlertIcon,  
 Grid,  
 GridItem,  
 Stack,  
 Button,  
 Select,  
 Input,  
 FormControl,  
 FormLabel,  
 useToast,  
} from '@chakra-ui/react';  
import { FiMapPin, FiPhone, FiMail, FiFilter, FiUser } from 'react-icons/fi';  
import { useState, useEffect } from 'react';  
import { useNavigate } from 'react-router-dom';  
  
function DonorMatchingPage() {  
 const [donors, setDonors] = useState([]);  
 const [filteredDonors, setFilteredDonors] = useState([]);  
 const [filters, setFilters] = useState({  
 bloodGroup: '',  
 location: '',  
 });  
 const [isLoading, setIsLoading] = useState(true);  
 const toast = useToast();  
 const navigate = useNavigate();  
  
 const bloodGroups = ['A+', 'A-', 'B+', 'B-', 'AB+', 'AB-', 'O+', 'O-'];  
  
 useEffect(() => {  
 fetchDonors();  
 }, []);  
  
 useEffect(() => {  
 applyFilters();  
 }, [donors, filters]);  
  
 const fetchDonors = async () => {  
 try {  
 const response = await fetch('/api/donors/nearby', {  
 headers: {  
 Authorization: `Bearer ${localStorage.getItem('token')}`,  
 },  
 });  
  
 if (response.ok) {  
 const donorsData = await response.json();  
 setDonors(donorsData);  
 }  
 } catch (error) {  
 console.error('Failed to fetch donors:', error);  
 toast({  
 title: 'Failed to load donors',  
 description: 'Please try again later',  
 status: 'error',  
 duration: 3000,  
 });  
 } finally {  
 setIsLoading(false);  
 }  
 };  
  
 const applyFilters = () => {  
 let filtered = [...donors];  
  
 if (filters.bloodGroup) {  
 filtered = filtered.filter(donor => donor.bloodGroup === filters.bloodGroup);  
 }  
  
 if (filters.location) {  
 filtered = filtered.filter(donor =>  
 donor.location.toLowerCase().includes(filters.location.toLowerCase())  
 );  
 }  
  
 setFilteredDonors(filtered);  
   
 // Redirect to no donors page if no results  
 if (filtered.length === 0 && donors.length > 0) {  
 navigate('/no-donors');  
 }  
 };  
  
 const handleFilterChange = (e) => {  
 const { name, value } = e.target;  
 setFilters(prev => ({  
 ...prev,  
 [name]: value,  
 }));  
 };  
  
 const clearFilters = () => {  
 setFilters({  
 bloodGroup: '',  
 location: '',  
 });  
 };  
  
 const contactDonor = (donor, method) => {  
 if (method === 'phone') {  
 window.open(`tel:${donor.phone}`, '\_blank');  
 } else if (method === 'email') {  
 window.open(`mailto:${donor.email}`, '\_blank');  
 }  
   
 toast({  
 title: 'Contact Initiated',  
 description: `Opening ${method} to contact ${donor.name}`,  
 status: 'success',  
 duration: 3000,  
 });  
 };  
  
 if (isLoading) {  
 return (  
 <Container maxW="6xl" py={8}>  
 <Text>Loading available donors...</Text>  
 </Container>  
 );  
 }  
  
 return (  
 <Container maxW="6xl" py={8}>  
 <VStack spacing={8} align="stretch">  
 <Box>  
 <Heading size="lg" mb={2}>Find Blood Donors</Heading>  
 <Text color="gray.600">  
 Connect with available donors in your area  
 </Text>  
 </Box>  
  
 {/\* Filters \*/}  
 <Card>  
 <CardBody>  
 <VStack spacing={4} align="stretch">  
 <HStack>  
 <Icon as={FiFilter} color="primary.500" />  
 <Heading size="md">Filter Donors</Heading>  
 </HStack>  
   
 <Grid templateColumns={{ base: '1fr', md: 'repeat(3, 1fr)' }} gap={4}>  
 <FormControl>  
 <FormLabel>Blood Group</FormLabel>  
 <Select  
 name="bloodGroup"  
 value={filters.bloodGroup}  
 onChange={handleFilterChange}  
 placeholder="All blood groups"  
 >  
 {bloodGroups.map(group => (  
 <option key={group} value={group}>{group}</option>  
 ))}  
 </Select>  
 </FormControl>  
  
 <FormControl>  
 <FormLabel>Location</FormLabel>  
 <Input  
 name="location"  
 value={filters.location}  
 onChange={handleFilterChange}  
 placeholder="Enter location"  
 />  
 </FormControl>  
  
 <FormControl display="flex" alignItems="end">  
 <Button onClick={clearFilters} variant="outline" w="full">  
 Clear Filters  
 </Button>  
 </FormControl>  
 </Grid>  
  
 <Text fontSize="sm" color="gray.600">  
 Found {filteredDonors.length} available donors  
 </Text>  
 </VStack>  
 </CardBody>  
 </Card>  
  
 {/\* Donors List \*/}  
 {filteredDonors.length > 0 ? (  
 <Grid templateColumns={{ base: '1fr', md: 'repeat(2, 1fr)', lg: 'repeat(3, 1fr)' }} gap={6}>  
 {filteredDonors.map((donor) => (  
 <Card key={donor.\_id} variant="outline" \_hover={{ boxShadow: 'md' }}>  
 <CardBody>  
 <Stack spacing={4}>  
 <HStack justify="space-between">  
 <HStack>  
 <Icon as={FiUser} color="primary.500" />  
 <Text fontWeight="medium">{donor.name}</Text>  
 </HStack>  
 <Badge colorScheme="primary" fontSize="md" px={3} py={1}>  
 {donor.bloodGroup}  
 </Badge>  
 </HStack>  
   
 <VStack spacing={2} align="start">  
 <HStack>  
 <Icon as={FiMapPin} color="gray.500" w={4} h={4} />  
 <Text fontSize="sm" color="gray.600">  
 {donor.location}  
 </Text>  
 </HStack>  
   
 <HStack>  
 <Badge colorScheme="green" size="sm">Available</Badge>  
 <Text fontSize="sm" color="gray.500">  
 Last active: {new Date(donor.lastActive).toLocaleDateString()}  
 </Text>  
 </HStack>  
 </VStack>  
  
 <Stack spacing={2}>  
 <Text fontSize="sm" fontWeight="medium" color="gray.700">  
 Contact Options:  
 </Text>  
 <HStack spacing={2}>  
 <Button  
 size="sm"  
 colorScheme="blue"  
 leftIcon={<Icon as={FiPhone} />}  
 onClick={() => contactDonor(donor, 'phone')}  
 flex={1}  
 >  
 Call  
 </Button>  
 <Button  
 size="sm"  
 colorScheme="green"  
 leftIcon={<Icon as={FiMail} />}  
 onClick={() => contactDonor(donor, 'email')}  
 flex={1}  
 >  
 Email  
 </Button>  
 </HStack>  
 </Stack>  
 </Stack>  
 </CardBody>  
 </Card>  
 ))}  
 </Grid>  
 ) : (  
 <Alert status="warning">  
 <AlertIcon />  
 <Box>  
 <Text fontWeight="medium">No donors found</Text>  
 <Text fontSize="sm">  
 Try adjusting your filters or check back later for new donors.  
 </Text>  
 </Box>  
 </Alert>  
 )}  
 </VStack>  
 </Container>  
 );  
}  
  
export default DonorMatchingPage;

================================================================================

## 53. Notificationspage

📂 File Path: C:\Users\Admin\React\sai\project(hemo\_connect)\src\pages\Notifications\NotificationsPage.jsx

import {  
 Container,  
 VStack,  
 Heading,  
 Text,  
 Card,  
 CardBody,  
 Badge,  
 HStack,  
 Icon,  
 Button,  
 Box,  
 Alert,  
 AlertIcon,  
} from '@chakra-ui/react';  
import { FiBell, FiTrash2, FiCheck } from 'react-icons/fi';  
import { useSocket } from '../../context/SocketContext';  
  
function NotificationsPage() {  
 const { notifications, setNotifications } = useSocket();  
  
 const markAsRead = (notificationId) => {  
 setNotifications(prev =>  
 prev.map(notification =>  
 notification.id === notificationId  
 ? { ...notification, read: true }  
 : notification  
 )  
 );  
 };  
  
 const deleteNotification = (notificationId) => {  
 setNotifications(prev =>  
 prev.filter(notification => notification.id !== notificationId)  
 );  
 };  
  
 const clearAll = () => {  
 setNotifications([]);  
 };  
  
 const getNotificationIcon = (type) => {  
 switch (type) {  
 case 'blood\_request':  
 return FiBell;  
 default:  
 return FiBell;  
 }  
 };  
  
 const getNotificationColor = (type) => {  
 switch (type) {  
 case 'blood\_request':  
 return 'red';  
 case 'donor\_response':  
 return 'blue';  
 case 'request\_fulfilled':  
 return 'green';  
 default:  
 return 'gray';  
 }  
 };  
  
 return (  
 <Container maxW="4xl" py={8}>  
 <VStack spacing={8} align="stretch">  
 <HStack justify="space-between" align="start">  
 <Box>  
 <Heading size="lg" mb={2}>Notifications</Heading>  
 <Text color="gray.600">  
 Stay updated with blood requests and responses  
 </Text>  
 </Box>  
 {notifications.length > 0 && (  
 <Button  
 size="sm"  
 variant="outline"  
 onClick={clearAll}  
 leftIcon={<Icon as={FiTrash2} />}  
 >  
 Clear All  
 </Button>  
 )}  
 </HStack>  
  
 {notifications.length > 0 ? (  
 <VStack spacing={4}>  
 {notifications.map((notification) => (  
 <Card  
 key={notification.id}  
 w="full"  
 variant={notification.read ? "outline" : "elevated"}  
 bg={notification.read ? "gray.50" : "white"}  
 >  
 <CardBody>  
 <HStack justify="space-between" align="start">  
 <HStack spacing={4} flex={1}>  
 <Icon  
 as={getNotificationIcon(notification.type)}  
 color={`${getNotificationColor(notification.type)}.500`}  
 w={6}  
 h={6}  
 />  
 <Box flex={1}>  
 <HStack spacing={2} mb={2}>  
 <Text fontWeight="medium">  
 {notification.title}  
 </Text>  
 {!notification.read && (  
 <Badge colorScheme="blue" size="sm">New</Badge>  
 )}  
 <Badge   
 colorScheme={getNotificationColor(notification.type)}  
 size="sm"  
 >  
 {notification.type.replace('\_', ' ')}  
 </Badge>  
 </HStack>  
 <Text color="gray.600" mb={2}>  
 {notification.message}  
 </Text>  
 <Text fontSize="sm" color="gray.500">  
 {new Date(notification.timestamp).toLocaleString()}  
 </Text>  
 </Box>  
 </HStack>  
   
 <VStack spacing={2}>  
 {!notification.read && (  
 <Button  
 size="sm"  
 variant="ghost"  
 onClick={() => markAsRead(notification.id)}  
 leftIcon={<Icon as={FiCheck} />}  
 >  
 Mark Read  
 </Button>  
 )}  
 <Button  
 size="sm"  
 variant="ghost"  
 colorScheme="red"  
 onClick={() => deleteNotification(notification.id)}  
 >  
 <Icon as={FiTrash2} />  
 </Button>  
 </VStack>  
 </HStack>  
 </CardBody>  
 </Card>  
 ))}  
 </VStack>  
 ) : (  
 <Alert status="info">  
 <AlertIcon />  
 <Box>  
 <Text fontWeight="medium">No notifications</Text>  
 <Text fontSize="sm">  
 You're all caught up! New notifications will appear here.  
 </Text>  
 </Box>  
 </Alert>  
 )}  
 </VStack>  
 </Container>  
 );  
}  
  
export default NotificationsPage;

================================================================================

## 54. Bloodrequestpage

📂 File Path: C:\Users\Admin\React\sai\project(hemo\_connect)\src\pages\Request\BloodRequestPage.jsx

import {  
 Box,  
 Button,  
 Container,  
 FormControl,  
 FormLabel,  
 Heading,  
 Input,  
 Stack,  
 Text,  
 useToast,  
 VStack,  
 Card,  
 CardBody,  
 Icon,  
 Select,  
 Textarea,  
 NumberInput,  
 NumberInputField,  
 NumberInputStepper,  
 NumberIncrementStepper,  
 NumberDecrementStepper,  
 Alert,  
 AlertIcon,  
} from '@chakra-ui/react';  
import { FiHeart } from 'react-icons/fi';  
import { useState } from 'react';  
import { useNavigate } from 'react-router-dom';  
import { useAuth } from '../../context/AuthContext';  
  
function BloodRequestPage() {  
 const { user } = useAuth();  
 const [formData, setFormData] = useState({  
 bloodGroup: '',  
 units: 1,  
 hospital: user?.hospital || '',  
 contactNumber: user?.phone || '',  
 urgencyLevel: 'medium',  
 additionalNotes: '',  
 requiredByDate: '',  
 });  
 const [isLoading, setIsLoading] = useState(false);  
 const toast = useToast();  
 const navigate = useNavigate();  
  
 const bloodGroups = ['A+', 'A-', 'B+', 'B-', 'AB+', 'AB-', 'O+', 'O-'];  
  
 const handleChange = (e) => {  
 const { name, value } = e.target;  
 setFormData(prev => ({  
 ...prev,  
 [name]: value,  
 }));  
 };  
  
 const handleNumberChange = (valueString) => {  
 setFormData(prev => ({  
 ...prev,  
 units: parseInt(valueString) || 1,  
 }));  
 };  
  
 const handleSubmit = async (e) => {  
 e.preventDefault();  
 setIsLoading(true);  
  
 try {  
 const response = await fetch('/api/requests', {  
 method: 'POST',  
 headers: {  
 'Content-Type': 'application/json',  
 Authorization: `Bearer ${localStorage.getItem('token')}`,  
 },  
 body: JSON.stringify(formData),  
 });  
  
 if (response.ok) {  
 const newRequest = await response.json();  
 toast({  
 title: 'Request Submitted',  
 description: 'Your blood request has been submitted successfully!',  
 status: 'success',  
 duration: 5000,  
 });  
   
 // Redirect to request success page or dashboard  
 navigate('/recipient-dashboard', {   
 state: { newRequestId: newRequest.\_id }   
 });  
 } else {  
 const data = await response.json();  
 throw new Error(data.message);  
 }  
 } catch (error) {  
 toast({  
 title: 'Submission Failed',  
 description: error.message || 'Failed to submit blood request',  
 status: 'error',  
 duration: 5000,  
 });  
 }  
  
 setIsLoading(false);  
 };  
  
 return (  
 <Container maxW="2xl" py={12}>  
 <VStack spacing={8}>  
 <VStack spacing={4} textAlign="center">  
 <Icon as={FiHeart} w={12} h={12} color="primary.500" />  
 <Heading size="lg">Request Blood</Heading>  
 <Text color="gray.600">  
 Submit your blood request and we'll notify nearby donors  
 </Text>  
 </VStack>  
  
 <Alert status="info">  
 <AlertIcon />  
 <Box>  
 <Text fontWeight="medium">Important:</Text>  
 <Text fontSize="sm">  
 Please ensure all information is accurate. Emergency requests will be prioritized.  
 </Text>  
 </Box>  
 </Alert>  
  
 <Card w="full">  
 <CardBody p={8}>  
 <form onSubmit={handleSubmit}>  
 <Stack spacing={6}>  
 <FormControl isRequired>  
 <FormLabel>Blood Group Needed</FormLabel>  
 <Select  
 name="bloodGroup"  
 value={formData.bloodGroup}  
 onChange={handleChange}  
 placeholder="Select blood group"  
 >  
 {bloodGroups.map(group => (  
 <option key={group} value={group}>{group}</option>  
 ))}  
 </Select>  
 </FormControl>  
  
 <FormControl isRequired>  
 <FormLabel>Units Required</FormLabel>  
 <NumberInput  
 value={formData.units}  
 onChange={handleNumberChange}  
 min={1}  
 max={10}  
 >  
 <NumberInputField />  
 <NumberInputStepper>  
 <NumberIncrementStepper />  
 <NumberDecrementStepper />  
 </NumberInputStepper>  
 </NumberInput>  
 </FormControl>  
  
 <FormControl isRequired>  
 <FormLabel>Hospital/Medical Center</FormLabel>  
 <Input  
 name="hospital"  
 value={formData.hospital}  
 onChange={handleChange}  
 placeholder="Enter hospital name"  
 />  
 </FormControl>  
  
 <FormControl isRequired>  
 <FormLabel>Contact Number</FormLabel>  
 <Input  
 name="contactNumber"  
 value={formData.contactNumber}  
 onChange={handleChange}  
 placeholder="Enter contact number"  
 />  
 </FormControl>  
  
 <FormControl isRequired>  
 <FormLabel>Urgency Level</FormLabel>  
 <Select  
 name="urgencyLevel"  
 value={formData.urgencyLevel}  
 onChange={handleChange}  
 >  
 <option value="low">Low - Planned Surgery (1+ weeks)</option>  
 <option value="medium">Medium - Within 48 hours</option>  
 <option value="high">High - Within 24 hours</option>  
 <option value="critical">Critical - Immediate</option>  
 </Select>  
 </FormControl>  
  
 <FormControl>  
 <FormLabel>Required By Date</FormLabel>  
 <Input  
 name="requiredByDate"  
 type="datetime-local"  
 value={formData.requiredByDate}  
 onChange={handleChange}  
 />  
 </FormControl>  
  
 <FormControl>  
 <FormLabel>Additional Notes</FormLabel>  
 <Textarea  
 name="additionalNotes"  
 value={formData.additionalNotes}  
 onChange={handleChange}  
 placeholder="Any additional information for donors"  
 rows={4}  
 />  
 </FormControl>  
  
 <Button  
 type="submit"  
 colorScheme="primary"  
 size="lg"  
 fontSize="md"  
 isLoading={isLoading}  
 loadingText="Submitting request..."  
 >  
 Submit Blood Request  
 </Button>  
 </Stack>  
 </form>  
 </CardBody>  
 </Card>  
  
 <Text textAlign="center" fontSize="sm" color="gray.500">  
 Your request will be shared with verified donors in your area.  
 You'll receive notifications as soon as donors respond.  
 </Text>  
 </VStack>  
 </Container>  
 );  
}  
  
export default BloodRequestPage;

================================================================================

## 55. Index

📂 File Path: C:\Users\Admin\React\sai\project(hemo\_connect)\src\theme\index.jsx

import { extendTheme } from '@chakra-ui/react';  
  
const theme = extendTheme({  
 colors: {  
 brand: {  
 50: '#fef2f2',  
 100: '#fee2e2',  
 200: '#fecaca',  
 300: '#fca5a5',  
 400: '#f87171',  
 500: '#ef4444',  
 600: '#dc2626',  
 700: '#b91c1c',  
 800: '#991b1b',  
 900: '#7f1d1d',  
 },  
 primary: {  
 50: '#fef2f2',  
 100: '#fee2e2',  
 200: '#fecaca',  
 300: '#fca5a5',  
 400: '#f87171',  
 500: '#E53E3E',  
 600: '#dc2626',  
 700: '#b91c1c',  
 800: '#991b1b',  
 900: '#7f1d1d',  
 },  
 secondary: {  
 50: '#f0fdf4',  
 100: '#dcfce7',  
 200: '#bbf7d0',  
 300: '#86efac',  
 400: '#4ade80',  
 500: '#22c55e',  
 600: '#16a34a',  
 700: '#15803d',  
 800: '#166534',  
 900: '#14532d',  
 }  
 },  
 fonts: {  
 heading: '"Inter", -apple-system, BlinkMacSystemFont, "Segoe UI", Helvetica, Arial, sans-serif',  
 body: '"Inter", -apple-system, BlinkMacSystemFont, "Segoe UI", Helvetica, Arial, sans-serif',  
 },  
 components: {  
 Button: {  
 defaultProps: {  
 colorScheme: 'primary',  
 },  
 variants: {  
 solid: {  
 \_hover: {  
 transform: 'translateY(-2px)',  
 boxShadow: 'lg',  
 },  
 transition: 'all 0.2s',  
 },  
 },  
 },  
 Card: {  
 baseStyle: {  
 container: {  
 borderRadius: 'xl',  
 boxShadow: 'sm',  
 \_hover: {  
 boxShadow: 'md',  
 },  
 transition: 'all 0.2s',  
 },  
 },  
 },  
 },  
 config: {  
 initialColorMode: 'light',  
 useSystemColorMode: false,  
 },  
});  
  
export default theme;

================================================================================