**FableLink App**

**Backend Codes**

**// /backend/config/db.js**

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

**// Function to connect to MongoDB**

**const connectDB = async () => {**

**try {**

**const conn = await mongoose.connect(process.env.MONGO\_URI );**

**console.log(`MongoDB Connected: ${conn.connection.host}`);**

**} catch (error) {**

**console.error(`Error: ${error.message}`);**

**process.exit(1); // Exit with failure**

**}**

**};**

**module.exports = connectDB;**

**Controllers**

**// /backend/controllers/agentController.js**

**const Agent = require('../models/Agent');**

**// Get AI agent for a user**

**const getAgent = async (req, res) => {**

**const agent = await Agent.findOne({ userId: req.params.userId });**

**if (agent) {**

**res.json(agent);**

**} else {**

**res.status(404).json({ message: 'Agent not found' });**

**}**

**};**

**// Update AI agent's personality**

**const updateAgent = async (req, res) => {**

**const agent = await Agent.findOne({ userId: req.params.userId });**

**if (agent) {**

**agent.agentPersonality = { ...agent.agentPersonality, ...req.body.agentPersonality };**

**const updatedAgent = await agent.save();**

**res.json(updatedAgent);**

**} else {**

**res.status(404).json({ message: 'Agent not found' });**

**}**

**};**

**// Get AI agent's interaction history**

**const getAgentInteractions = async (req, res) => {**

**const agent = await Agent.findOne({ userId: req.params.userId });**

**if (agent) {**

**res.json(agent.trainingData.interactionHistory);**

**} else {**

**res.status(404).json({ message: 'Agent not found' });**

**}**

**};**

**// Add an interaction to the AI agent's history**

**const addInteraction = async (req, res) => {**

**const agent = await Agent.findOne({ userId: req.params.userId });**

**if (agent) {**

**agent.trainingData.interactionHistory.push(req.body);**

**const updatedAgent = await agent.save();**

**res.json(updatedAgent);**

**} else {**

**res.status(404).json({ message: 'Agent not found' });**

**}**

**};**

**module.exports = { getAgent, updateAgent, getAgentInteractions, addInteraction };**

**// /backend/controllers/connectionController.js**

**const Connection = require('../models/Connection');**

**// Create a new connection**

**const createConnection = async (req, res) => {**

**const { userOneId, userTwoId, initiatedBy } = req.body;**

**try {**

**const connection = new Connection({ userOneId, userTwoId, initiatedBy });**

**const newConnection = await connection.save();**

**res.status(201).json(newConnection);**

**} catch (error) {**

**res.status(500).json({ message: error.message });**

**}**

**};**

**// Get all connections for a user**

**const getConnections = async (req, res) => {**

**const connections = await Connection.find({ $or: [{ userOneId: req.params.userId }, { userTwoId: req.params.userId }] });**

**res.json(connections);**

**};**

**// Update connection status**

**const updateConnectionStatus = async (req, res) => {**

**const connection = await Connection.findById(req.params.connectionId);**

**if (connection) {**

**connection.status = req.body.status || connection.status;**

**const updatedConnection = await connection.save();**

**res.json(updatedConnection);**

**} else {**

**res.status(404).json({ message: 'Connection not found' });**

**}**

**};**

**module.exports = { createConnection, getConnections, updateConnectionStatus };**

**// /backend/controllers/messageController.js**

**const Message = require('../models/Message');**

**// Send a message between users**

**const sendMessage = async (req, res) => {**

**const { senderId, receiverId, message } = req.body;**

**try {**

**const newMessage = new Message({ senderId, receiverId, message });**

**const savedMessage = await newMessage.save();**

**res.status(201).json(savedMessage);**

**} catch (error) {**

**res.status(500).json({ message: error.message });**

**}**

**};**

**// Get all messages between users**

**const getMessages = async (req, res) => {**

**const messages = await Message.find({**

**$or: [**

**{ senderId: req.params.userId, receiverId: req.params.receiverId },**

**{ senderId: req.params.receiverId, receiverId: req.params.userId }**

**]**

**});**

**res.json(messages);**

**};**

**module.exports = { sendMessage, getMessages };**

**// /backend/controllers/userController.js**

**const User = require('../models/User');**

**const bcrypt = require('bcryptjs');**

**const jwt = require('jsonwebtoken');**

**// Register a new user**

**const registerUser = async (req, res) => {**

**const { username, email, password } = req.body;**

**try {**

**const userExists = await User.findOne({ email });**

**if (userExists) {**

**return res.status(400).json({ message: 'User already exists' });**

**}**

**const hashedPassword = await bcrypt.hash(password, 10);**

**const user = new User({**

**username,**

**email,**

**password: hashedPassword**

**});**

**const newUser = await user.save();**

**res.status(201).json(newUser);**

**} catch (error) {**

**res.status(500).json({ message: error.message });**

**}**

**};**

**// Login a user and generate a token**

**const loginUser = async (req, res) => {**

**const { email, password } = req.body;**

**try {**

**const user = await User.findOne({ email });**

**if (user && (await bcrypt.compare(password, user.password))) {**

**const token = jwt.sign({ id: user.\_id }, process.env.JWT\_SECRET, {**

**expiresIn: '30d'**

**});**

**res.json({ token, user });**

**} else {**

**res.status(401).json({ message: 'Invalid credentials' });**

**}**

**} catch (error) {**

**res.status(500).json({ message: error.message });**

**}**

**};**

**// Get user profile**

**const getUserProfile = async (req, res) => {**

**const user = await User.findById(req.user.id);**

**if (user) {**

**res.json(user);**

**} else {**

**res.status(404).json({ message: 'User not found' });**

**}**

**};**

**// Update user profile**

**const updateUserProfile = async (req, res) => {**

**const user = await User.findById(req.user.id);**

**if (user) {**

**user.profile = { ...user.profile, ...req.body.profile };**

**user.preferences = { ...user.preferences, ...req.body.preferences };**

**const updatedUser = await user.save();**

**res.json(updatedUser);**

**} else {**

**res.status(404).json({ message: 'User not found' });**

**}**

**};**

**// Get all users**

**const getAllUsers = async (req, res) => {**

**const users = await User.find({});**

**res.json(users);**

**};**

**module.exports = { registerUser, loginUser, getUserProfile, updateUserProfile, getAllUsers };**

**// /backend/controllers/userController.js**

**const User = require('../models/User');**

**const bcrypt = require('bcryptjs');**

**const jwt = require('jsonwebtoken');**

**// Register a new user**

**const registerUser = async (req, res) => {**

**const { username, email, password } = req.body;**

**try {**

**const userExists = await User.findOne({ email });**

**if (userExists) {**

**return res.status(400).json({ message: 'User already exists' });**

**}**

**const hashedPassword = await bcrypt.hash(password, 10);**

**const user = new User({**

**username,**

**email,**

**password: hashedPassword**

**});**

**const newUser = await user.save();**

**res.status(201).json(newUser);**

**} catch (error) {**

**res.status(500).json({ message: error.message });**

**}**

**};**

**// Login a user and generate a token**

**const loginUser = async (req, res) => {**

**const { email, password } = req.body;**

**try {**

**const user = await User.findOne({ email });**

**if (user && (await bcrypt.compare(password, user.password))) {**

**const token = jwt.sign({ id: user.\_id }, process.env.JWT\_SECRET, {**

**expiresIn: '30d'**

**});**

**res.json({ token, user });**

**} else {**

**res.status(401).json({ message: 'Invalid credentials' });**

**}**

**} catch (error) {**

**res.status(500).json({ message: error.message });**

**}**

**};**

**// Get user profile**

**const getUserProfile = async (req, res) => {**

**const user = await User.findById(req.user.id);**

**if (user) {**

**res.json(user);**

**} else {**

**res.status(404).json({ message: 'User not found' });**

**}**

**};**

**// Update user profile**

**const updateUserProfile = async (req, res) => {**

**const user = await User.findById(req.user.id);**

**if (user) {**

**user.profile = { ...user.profile, ...req.body.profile };**

**user.preferences = { ...user.preferences, ...req.body.preferences };**

**const updatedUser = await user.save();**

**res.json(updatedUser);**

**} else {**

**res.status(404).json({ message: 'User not found' });**

**}**

**};**

**// Get all users**

**const getAllUsers = async (req, res) => {**

**const users = await User.find({});**

**res.json(users);**

**};**

**module.exports = { registerUser, loginUser, getUserProfile, updateUserProfile, getAllUsers };**

**Middleware**

**// /backend/controllers/userController.js**

**const User = require('../models/User');**

**const bcrypt = require('bcryptjs');**

**const jwt = require('jsonwebtoken');**

**// Register a new user**

**const registerUser = async (req, res) => {**

**const { username, email, password } = req.body;**

**try {**

**const userExists = await User.findOne({ email });**

**if (userExists) {**

**return res.status(400).json({ message: 'User already exists' });**

**}**

**const hashedPassword = await bcrypt.hash(password, 10);**

**const user = new User({**

**username,**

**email,**

**password: hashedPassword**

**});**

**const newUser = await user.save();**

**res.status(201).json(newUser);**

**} catch (error) {**

**res.status(500).json({ message: error.message });**

**}**

**};**

**// Login a user and generate a token**

**const loginUser = async (req, res) => {**

**const { email, password } = req.body;**

**try {**

**const user = await User.findOne({ email });**

**if (user && (await bcrypt.compare(password, user.password))) {**

**const token = jwt.sign({ id: user.\_id }, process.env.JWT\_SECRET, {**

**expiresIn: '30d'**

**});**

**res.json({ token, user });**

**} else {**

**res.status(401).json({ message: 'Invalid credentials' });**

**}**

**} catch (error) {**

**res.status(500).json({ message: error.message });**

**}**

**};**

**// Get user profile**

**const getUserProfile = async (req, res) => {**

**const user = await User.findById(req.user.id);**

**if (user) {**

**res.json(user);**

**} else {**

**res.status(404).json({ message: 'User not found' });**

**}**

**};**

**// Update user profile**

**const updateUserProfile = async (req, res) => {**

**const user = await User.findById(req.user.id);**

**if (user) {**

**user.profile = { ...user.profile, ...req.body.profile };**

**user.preferences = { ...user.preferences, ...req.body.preferences };**

**const updatedUser = await user.save();**

**res.json(updatedUser);**

**} else {**

**res.status(404).json({ message: 'User not found' });**

**}**

**};**

**// Get all users**

**const getAllUsers = async (req, res) => {**

**const users = await User.find({});**

**res.json(users);**

**};**

**module.exports = { registerUser, loginUser, getUserProfile, updateUserProfile, getAllUsers };**

**// /backend/middleware/errorMiddleware.js**

**// Middleware for handling 404 Not Found**

**const notFound = (req, res, next) => {**

**const error = new Error(`Not Found - ${req.originalUrl}`);**

**res.status(404);**

**next(error);**

**};**

**// Middleware for handling errors**

**const errorHandler = (err, req, res, next) => {**

**const statusCode = res.statusCode === 200 ? 500 : res.statusCode;**

**res.status(statusCode);**

**res.json({**

**message: err.message,**

**stack: process.env.NODE\_ENV === 'production' ? null : err.stack,**

**});**

**};**

**module.exports = { notFound, errorHandler };**

**Models**

**// /backend/models/Agent.js**

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

**const interactionSchema = new mongoose.Schema({**

**userInput: { type: String, required: true },**

**agentResponse: { type: String, required: true },**

**timestamp: { type: Date, default: Date.now }**

**});**

**const agentSchema = new mongoose.Schema({**

**userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },**

**agentName: { type: String, default: 'AI Agent' },**

**agentPersonality: {**

**type: { type: String, enum: ['empathetic', 'professional', 'casual'], default: 'professional' },**

**tone: { type: String, default: 'friendly' },**

**responseSpeed: { type: String, default: 'fast' }**

**},**

**trainingData: {**

**interactionHistory: [interactionSchema]**

**},**

**createdAt: { type: Date, default: Date.now },**

**updatedAt: { type: Date, default: Date.now }**

**});**

**const Agent = mongoose.model('Agent', agentSchema);**

**module.exports = Agent;**

**// /backend/models/Connection.js**

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

**const connectionSchema = new mongoose.Schema({**

**userOneId: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },**

**userTwoId: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },**

**status: { type: String, enum: ['pending', 'connected', 'blocked'], default: 'pending' },**

**initiatedBy: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },**

**connectedAt: { type: Date, default: Date.now },**

**updatedAt: { type: Date, default: Date.now }**

**});**

**const Connection = mongoose.model('Connection', connectionSchema);**

**module.exports = Connection;**

**// /backend/models/Interest.js**

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

**const interestSchema = new mongoose.Schema({**

**interest: { type: String, required: true },**

**description: { type: String }**

**});**

**const Interest = mongoose.model('Interest', interestSchema);**

**module.exports = Interest;**

**// /backend/models/Message.js**

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

**const messageSchema = new mongoose.Schema({**

**senderId: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },  // Could also be an AI agent**

**receiverId: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },**

**message: { type: String, required: true },**

**messageType: { type: String, enum: ['text', 'image', 'file'], default: 'text' },**

**timestamp: { type: Date, default: Date.now }**

**});**

**const Message = mongoose.model('Message', messageSchema);**

**module.exports = Message;**

**// /backend/models/User.js**

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

**const connectionSchema = new mongoose.Schema({**

**userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },**

**status: { type: String, enum: ['pending', 'connected', 'blocked'], default: 'pending' },**

**connectedAt: { type: Date, default: Date.now },**

**});**

**const userSchema = new mongoose.Schema({**

**username: { type: String, required: true, unique: true },**

**email: { type: String, required: true, unique: true },**

**password: { type: String, required: true },**

**profile: {**

**name: { type: String },**

**bio: { type: String },**

**avatar: { type: String },**

**location: { type: String }**

**},**

**preferences: {**

**interests: { type: [String] },**

**language: { type: String, default: 'en' },**

**communicationStyle: { type: String, default: 'formal' }**

**},**

**connections: [connectionSchema],**

**createdAt: { type: Date, default: Date.now },**

**updatedAt: { type: Date, default: Date.now }**

**});**

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

**module.exports = User;**

**Routes**

**// /backend/routes/agentRoutes.js**

**const express = require('express');**

**const router = express.Router();**

**const { getAgent, updateAgent, getAgentInteractions, addInteraction } = require('../controllers/agentController');**

**const { protect } = require('../middleware/authMiddleware');**

**// Get the AI agent for a user (protected route)**

**router.get('/:userId', protect, getAgent);**

**// Update the AI agent's personality or other attributes**

**router.put('/:userId', protect, updateAgent);**

**// Get interaction history for an AI agent**

**router.get('/:userId/interactions', protect, getAgentInteractions);**

**// Add an interaction to the AI agent's history**

**router.post('/:userId/interactions', protect, addInteraction);**

**module.exports = router;**

**// /backend/routes/connectionRoutes.js**

**const express = require('express');**

**const router = express.Router();**

**const { createConnection, getConnections, updateConnectionStatus } = require('../controllers/connectionController');**

**const { protect } = require('../middleware/authMiddleware');**

**// Create a new connection between two users**

**router.post('/', protect, createConnection);**

**// Get all connections for a user**

**router.get('/:userId', protect, getConnections);**

**// Update connection status (e.g., pending to connected)**

**router.put('/:connectionId', protect, updateConnectionStatus);**

**module.exports = router;**

**// /backend/routes/messageRoutes.js**

**const express = require('express');**

**const router = express.Router();**

**const { sendMessage, getMessages } = require('../controllers/messageController');**

**const { protect } = require('../middleware/authMiddleware');**

**// Send a message between two users**

**router.post('/', protect, sendMessage);**

**// Get all messages between two users**

**router.get('/:userId/:receiverId', protect, getMessages);**

**module.exports = router;**

**// /backend/routes/messageRoutes.js**

**const express = require('express');**

**const router = express.Router();**

**const { sendMessage, getMessages } = require('../controllers/messageController');**

**const { protect } = require('../middleware/authMiddleware');**

**// Send a message between two users**

**router.post('/', protect, sendMessage);**

**// Get all messages between two users**

**router.get('/:userId/:receiverId', protect, getMessages);**

**module.exports = router;**

**// /backend/routes/testRoutes.js**

**const express = require('express');**

**const router = express.Router();**

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

**// Test MongoDB connection route**

**router.get('/check-db', async (req, res) => {**

**try {**

**const state = mongoose.connection.readyState;**

**const states = ['disconnected', 'connected', 'connecting', 'disconnecting'];**

**res.json({ status: states[state] });**

**} catch (error) {**

**res.status(500).json({ message: 'Error checking database connection', error: error.message });**

**}**

**});**

**module.exports = router;**

**// /backend/routes/userRoutes.js**

**const express = require('express');**

**const router = express.Router();**

**const { registerUser, loginUser, getUserProfile, updateUserProfile, getAllUsers } = require('../controllers/userController');**

**const { protect } = require('../middleware/authMiddleware');**

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

**// User registration route**

**router.post('/register', registerUser);**

**// User login route**

**router.post('/login', loginUser);**

**// Get user profile (protected route)**

**router.get('/profile', protect, getUserProfile);**

**// Update user profile (protected route)**

**router.put('/profile', protect, updateUserProfile);**

**// Get all users (this could be for exploring users, admin functions)**

**router.get('/', protect, getAllUsers);**

**module.exports = router;**

**Tests**

**// /backend/tests/protectedRoute.test.js**

**const request = require('supertest');**

**const app = require('../server');**

**const User = require('../models/User');**

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

**afterEach(async () => {**

**await User.deleteMany();**

**});**

**afterAll(async () => {**

**await mongoose.connection.close();**

**});**

**describe('Protected Route Access', () => {**

**it('should not allow access to a protected route without token', async () => {**

**const res = await request(app).get('/api/users/profile');**

**expect(res.statusCode).toEqual(401);**

**expect(res.body).toHaveProperty('message', 'Not authorized, no token');**

**});**

**});**

**// /backend/tests/user.test.js**

**const request = require('supertest');**

**const app = require('../server');**

**const User = require('../models/User');**

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

**// Clear the test database after each test**

**afterEach(async () => {**

**await User.deleteMany();**

**});**

**afterAll(async () => {**

**await mongoose.connection.close();**

**});**

**describe('User Registration and Login', () => {**

**it('should register a new user', async () => {**

**const res = await request(app)**

**.post('/api/users/register')**

**.send({**

**username: 'testuser',**

**email: 'test@example.com',**

**password: 'password123',**

**});**

**expect(res.statusCode).toEqual(201);**

**expect(res.body).toHaveProperty('\_id');**

**expect(res.body).toHaveProperty('username', 'testuser');**

**});**

**it('should login the registered user', async () => {**

**// First register the user**

**await request(app)**

**.post('/api/users/register')**

**.send({**

**username: 'testuser',**

**email: 'test@example.com',**

**password: 'password123',**

**});**

**// Then login**

**const res = await request(app)**

**.post('/api/users/login')**

**.send({**

**email: 'test@example.com',**

**password: 'password123',**

**});**

**expect(res.statusCode).toEqual(200);**

**expect(res.body).toHaveProperty('token');**

**});**

**it('should not allow registration with existing email', async () => {**

**// Register the user**

**await request(app)**

**.post('/api/users/register')**

**.send({**

**username: 'testuser',**

**email: 'test@example.com',**

**password: 'password123',**

**});**

**// Try to register again with the same email**

**const res = await request(app)**

**.post('/api/users/register')**

**.send({**

**username: 'testuser2',**

**email: 'test@example.com',**

**password: 'password456',**

**});**

**expect(res.statusCode).toEqual(400);**

**expect(res.body).toHaveProperty('message', 'User already exists');**

**});**

**});**

**Utils**

**// /backend/utils/generateToken.js**

**const jwt = require('jsonwebtoken');**

**// Function to generate a JWT token for a user**

**const generateToken = (id) => {**

**return jwt.sign({ id }, process.env.JWT\_SECRET, {**

**expiresIn: '30d', // Token valid for 30 days**

**});**

**};**

**module.exports = generateToken;**

**// /backend/utils/hashPassword.js**

**const bcrypt = require('bcryptjs');**

**// Function to hash a password**

**const hashPassword = async (password) => {**

**const salt = await bcrypt.genSalt(10);**

**const hashedPassword = await bcrypt.hash(password, salt);**

**return hashedPassword;**

**};**

**module.exports = hashPassword;**

**.env**

**MONGO\_URI=mongodb://127.0.0.1:27017/FableLink**

**JWT\_SECRET= 6794e3ffd46a1dfb6dd0c2c3eff124a420bdfb3e3e40bfcb14ac9dd22e203a9d**

**NODE\_ENV=development**

**Server.js**

**// /backend/server.js**

**const express = require('express');**

**const dotenv = require('dotenv');**

**const cors = require('cors');**

**const connectDB = require('./config/db'); // MongoDB connection setup**

**const { notFound, errorHandler } = require('./middleware/errorMiddleware'); // Custom error handlers**

**// Load environment variables**

**dotenv.config();**

**// Connect to MongoDB**

**connectDB();**

**const app = express();**

**// Middleware**

**app.use(express.json()); // Body parser for JSON data**

**app.use(cors()); // Allow cross-origin requests**

**// Import Routes**

**const userRoutes = require('./routes/userRoutes');**

**const agentRoutes = require('./routes/agentRoutes');**

**const connectionRoutes = require('./routes/connectionRoutes');**

**const messageRoutes = require('./routes/messageRoutes'); // Optional**

**const testRoutes = require('./routes/testRoutes');**

**// Use Routes**

**app.use('/api/users', userRoutes); // User routes**

**app.use('/api/agents', agentRoutes); // AI Agent routes**

**app.use('/api/connections', connectionRoutes); // User connection routes**

**app.use('/api/messages', messageRoutes); // Optional: Messaging routes**

**app.use('/api/test', testRoutes);**

**// Error Handling Middleware**

**app.use(notFound); // Handle 404 errors**

**app.use(errorHandler); // Custom error handling**

**// Server Port**

**const PORT = process.env.PORT || 5000;**

**// Start Server**

**app.listen(PORT, () => {**

**console.log(`Server is running on port ${PORT}`);**

**});**

**Package.json**

**{**

**"name": "backend",**

**"version": "1.0.0",**

**"description": "",**

**"main": "index.js",**

**"scripts": {**

**"start": "react-scripts start",**

**"build": "react-scripts build",**

**"test": "jest"**

**},**

**"keywords": [],**

**"author": "",**

**"license": "ISC",**

**"dependencies": {**

**"bcryptjs": "^2.4.3",**

**"body-parser": "^1.20.3",**

**"cors": "^2.8.5",**

**"dotenv": "^16.4.5",**

**"express": "^4.20.0",**

**"jsonwebtoken": "^9.0.2",**

**"mongoose": "^8.6.1"**

**},**

**"devDependencies": {**

**"jest": "^29.7.0",**

**"nodemon": "^3.1.4",**

**"supertest": "^7.0.0"**

**}**

**}**

**Structure for frontend**

**/frontend**

**├── /public # Static files (e.g., index.html, images)**

**├── /src**

**│ ├── /components # Reusable components (e.g., ChatBox, AgentInteraction)**

**│ ├── /pages # Page components (e.g., Dashboard, Login, Register)**

**│ ├── /services # API service calls to interact with backend (e.g., userService.js)**

**│ ├── /context # Context for global state management (e.g., AuthContext)**

**│ ├── /styles # Global styles (CSS or SCSS)**

**│ ├── App.js # Main App component**

**│ ├── index.js # Entry point for React application**

**└── package.json # Project metadata and dependencies**

**Codes for frontend**

**Components**

**AgentInteraction.js**

**import React, { useState, useEffect, useContext } from 'react';**

**import { Form, Button, ListGroup } from 'react-bootstrap';**

**import { agentService } from '../services/agentService';**

**import { AuthContext } from '../context/AuthContext';**

**const AgentInteraction = () => {**

**const { user } = useContext(AuthContext);**

**const [interactionHistory, setInteractionHistory] = useState([]);**

**const [userInput, setUserInput] = useState('');**

**// Fetch interaction history when the component mounts**

**useEffect(() => {**

**if (user) {**

**agentService.getAgent(user.id).then(agent => {**

**setInteractionHistory(agent.trainingData.interactionHistory);**

**});**

**}**

**}, [user]);**

**// Handle sending new input to the agent**

**const handleSendInteraction = async (e) => {**

**e.preventDefault();**

**if (userInput.trim()) {**

**const newInteraction = {**

**userInput: userInput,**

**agentResponse: "Generating response...",  // Simulating response, this should come from the backend**

**};**

**await agentService.updateAgent(user.id, { agentPersonality: newInteraction });**

**setInteractionHistory([...interactionHistory, newInteraction]);  // Add the new interaction to the list**

**setUserInput('');  // Clear the input field**

**}**

**};**

**return (**

**<div>**

**<h5>Agent Interaction</h5>**

**<ListGroup>**

**{interactionHistory.map((interaction, index) => (**

**<ListGroup.Item key={index}>**

**<strong>You:</strong> {interaction.userInput} <br />**

**<strong>Agent:</strong> {interaction.agentResponse}**

**</ListGroup.Item>**

**))}**

**</ListGroup>**

**<Form onSubmit={handleSendInteraction} className="mt-3">**

**<Form.Group controlId="userInput">**

**<Form.Control**

**type="text"**

**placeholder="Ask the agent..."**

**value={userInput}**

**onChange={(e) => setUserInput(e.target.value)}**

**required**

**/>**

**</Form.Group>**

**<Button variant="primary" type="submit">Send</Button>**

**</Form>**

**</div>**

**);**

**};**

**export default AgentInteraction;**

**ChatBox.js**

**import React, { useState, useContext } from 'react';**

**import { Form, Button, ListGroup } from 'react-bootstrap';**

**import { AuthContext } from '../context/AuthContext';**

**import { messageService } from '../services/messageService';**

**const ChatBox = ({ recipientId }) => {**

**const { user } = useContext(AuthContext);**

**const [messages, setMessages] = useState([]);**

**const [messageText, setMessageText] = useState('');**

**// Fetch messages between user and recipient when the component mounts**

**React.useEffect(() => {**

**if (user && recipientId) {**

**messageService.getMessages(user.id, recipientId).then(setMessages);**

**}**

**}, [user, recipientId]);**

**// Handle sending a new message**

**const handleSendMessage = async (e) => {**

**e.preventDefault();**

**if (messageText.trim()) {**

**const newMessage = {**

**senderId: user.id,**

**receiverId: recipientId,**

**message: messageText**

**};**

**await messageService.sendMessage(newMessage);**

**setMessages([...messages, newMessage]);  // Add the new message to the message list**

**setMessageText('');  // Clear the input field**

**}**

**};**

**return (**

**<div>**

**<ListGroup>**

**{messages.map((msg, index) => (**

**<ListGroup.Item key={index}>**

**<strong>{msg.senderId === user.id ? 'You' : 'Them'}:</strong> {msg.message}**

**</ListGroup.Item>**

**))}**

**</ListGroup>**

**<Form onSubmit={handleSendMessage} className="mt-3">**

**<Form.Group controlId="messageText">**

**<Form.Control**

**type="text"**

**placeholder="Type your message..."**

**value={messageText}**

**onChange={(e) => setMessageText(e.target.value)}**

**required**

**/>**

**</Form.Group>**

**<Button variant="primary" type="submit">Send</Button>**

**</Form>**

**</div>**

**);**

**};**

**export default ChatBox;**

**ConnectionList.js**

**import React, { useEffect, useState, useContext } from 'react';**

**import { ListGroup } from 'react-bootstrap';**

**import { connectionService } from '../services/connectionService';**

**import { AuthContext } from '../context/AuthContext';**

**const ConnectionList = () => {**

**const { user } = useContext(AuthContext);**

**const [connections, setConnections] = useState([]);**

**// Fetch connections when the component mounts**

**useEffect(() => {**

**if (user) {**

**connectionService.getConnections(user.id).then(setConnections);**

**}**

**}, [user]);**

**return (**

**<div>**

**<h5>Your Connections</h5>**

**<ListGroup>**

**{connections.map((connection, index) => (**

**<ListGroup.Item key={index}>**

**Connected with: {connection.userTwoId === user.id ? connection.userOneId : connection.userTwoId}**

**<br />**

**Status: {connection.status}**

**</ListGroup.Item>**

**))}**

**</ListGroup>**

**</div>**

**);**

**};**

**export default ConnectionList;**

**MessageList.js**

**import React, { useEffect, useState, useContext } from 'react';**

**import { ListGroup } from 'react-bootstrap';**

**import { messageService } from '../services/messageService';**

**import { AuthContext } from '../context/AuthContext';**

**const MessageList = ({ recipientId }) => {**

**const { user } = useContext(AuthContext);**

**const [messages, setMessages] = useState([]);**

**// Fetch messages when the component mounts**

**useEffect(() => {**

**if (user && recipientId) {**

**messageService.getMessages(user.id, recipientId).then(setMessages);**

**}**

**}, [user, recipientId]);**

**return (**

**<div>**

**<h5>Message History</h5>**

**<ListGroup>**

**{messages.map((msg, index) => (**

**<ListGroup.Item key={index}>**

**<strong>{msg.senderId === user.id ? 'You' : 'Them'}:</strong> {msg.message}**

**</ListGroup.Item>**

**))}**

**</ListGroup>**

**</div>**

**);**

**};**

**export default MessageList;**

**Context**

**AgentContext.js**

**import React, { createContext, useState, useContext, useEffect } from 'react';**

**import { agentService } from '../services/agentService';**

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

**// Create the context**

**export const AgentContext = createContext();**

**export const AgentProvider = ({ children }) => {**

**const { user } = useContext(AuthContext);  // Get the current user from AuthContext**

**const [agent, setAgent] = useState(null);**

**const [loading, setLoading] = useState(true);**

**// Fetch the agent when the user is logged in**

**useEffect(() => {**

**if (user) {**

**fetchAgent(user.id);**

**}**

**}, [user]);**

**// Fetch agent details**

**const fetchAgent = async (userId) => {**

**try {**

**const agentData = await agentService.getAgent(userId);**

**setAgent(agentData);**

**} catch (error) {**

**console.error('Error fetching agent data', error);**

**} finally {**

**setLoading(false);**

**}**

**};**

**// Update agent's personality or other data**

**const updateAgent = async (userId, agentData) => {**

**try {**

**const updatedAgent = await agentService.updateAgent(userId, agentData);**

**setAgent(updatedAgent);**

**} catch (error) {**

**console.error('Error updating agent', error);**

**}**

**};**

**return (**

**<AgentContext.Provider value={{ agent, fetchAgent, updateAgent, loading }}>**

**{children}**

**</AgentContext.Provider>**

**);**

**};**

**AuthContext.js**

**import React, { createContext, useState, useEffect } from 'react';**

**import { userService } from '../services/userService';**

**// Create the context**

**export const AuthContext = createContext();**

**export const AuthProvider = ({ children }) => {**

**const [user, setUser] = useState(null);**

**const [loading, setLoading] = useState(true);**

**// Check if user is logged in when the app loads**

**useEffect(() => {**

**const storedUser = localStorage.getItem('user');**

**if (storedUser) {**

**setUser(JSON.parse(storedUser));**

**}**

**setLoading(false);**

**}, []);**

**// Login function**

**const login = async (email, password) => {**

**const response = await userService.loginUser({ email, password });**

**setUser(response.user);**

**localStorage.setItem('user', JSON.stringify(response.user));  // Save user to localStorage**

**return response;**

**};**

**// Logout function**

**const logout = () => {**

**setUser(null);**

**localStorage.removeItem('user');  // Remove user from localStorage**

**};**

**return (**

**<AuthContext.Provider value={{ user, login, logout, loading }}>**

**{children}**

**</AuthContext.Provider>**

**);**

**};**

**Pages/**

**Dashboard.js**

**import React, { useContext, useEffect, useState } from 'react';**

**import { AuthContext } from '../context/AuthContext';**

**import { connectionService } from '../services/connectionService';**

**import { messageService } from '../services/messageService';**

**import { agentService } from '../services/agentService';**

**import { Container, Row, Col, Card, Button } from 'react-bootstrap';**

**const Dashboard = () => {**

**const { user } = useContext(AuthContext);**

**const [connections, setConnections] = useState([]);**

**const [agent, setAgent] = useState(null);**

**const [messages, setMessages] = useState([]);**

**useEffect(() => {**

**if (user) {**

**// Fetch connections**

**connectionService.getConnections(user.id).then(setConnections);**

**// Fetch AI agent data**

**agentService.getAgent(user.id).then(setAgent);**

**// Fetch messages**

**messageService.getMessages(user.id).then(setMessages);**

**}**

**}, [user]);**

**if (!user) {**

**return <h4>Please log in to view your dashboard</h4>;**

**}**

**return (**

**<Container>**

**<h1 className="my-4">Welcome, {user.username}!</h1>**

**<Row>**

**<Col md={4}>**

**<Card className="mb-4">**

**<Card.Body>**

**<Card.Title>Connections</Card.Title>**

**<Card.Text>**

**You have {connections.length} connections.**

**</Card.Text>**

**<Button variant="primary">View Connections</Button>**

**</Card.Body>**

**</Card>**

**</Col>**

**<Col md={4}>**

**<Card className="mb-4">**

**<Card.Body>**

**<Card.Title>AI Agent</Card.Title>**

**{agent ? (**

**<>**

**<Card.Text>**

**Agent Name: {agent.agentName}**

**</Card.Text>**

**<Button variant="primary">View Agent</Button>**

**</>**

**) : (**

**<Card.Text>No agent data available.</Card.Text>**

**)}**

**</Card.Body>**

**</Card>**

**</Col>**

**<Col md={4}>**

**<Card className="mb-4">**

**<Card.Body>**

**<Card.Title>Messages</Card.Title>**

**<Card.Text>**

**You have {messages.length} messages.**

**</Card.Text>**

**<Button variant="primary">View Messages</Button>**

**</Card.Body>**

**</Card>**

**</Col>**

**</Row>**

**</Container>**

**);**

**};**

**export default Dashboard;**

**Login.js**

**import React, { useState, useContext } from 'react';**

**import { AuthContext } from '../context/AuthContext';**

**import { userService } from '../services/userService';**

**import { Form, Button, Container, Row, Col, Alert } from 'react-bootstrap';**

**import { useNavigate } from 'react-router-dom';**

**const Login = () => {**

**const [email, setEmail] = useState('');**

**const [password, setPassword] = useState('');**

**const [error, setError] = useState('');**

**const { setUser } = useContext(AuthContext);**

**const Navigate = useNavigate();**

**const handleLogin = async (e) => {**

**e.preventDefault();**

**try {**

**const response = await userService.loginUser({ email, password });**

**setUser(response.user);**

**Navigate.push('/dashboard');**

**} catch (err) {**

**setError('Invalid email or password');**

**}**

**};**

**return (**

**<Container>**

**<Row className="justify-content-md-center">**

**<Col md={6}>**

**<h2 className="my-4">Login</h2>**

**{error && <Alert variant="danger">{error}</Alert>}**

**<Form onSubmit={handleLogin}>**

**<Form.Group className="mb-3" controlId="formBasicEmail">**

**<Form.Label>Email address</Form.Label>**

**<Form.Control**

**type="email"**

**placeholder="Enter email"**

**value={email}**

**onChange={(e) => setEmail(e.target.value)}**

**required**

**/>**

**</Form.Group>**

**<Form.Group className="mb-3" controlId="formBasicPassword">**

**<Form.Label>Password</Form.Label>**

**<Form.Control**

**type="password"**

**placeholder="Password"**

**value={password}**

**onChange={(e) => setPassword(e.target.value)}**

**required**

**/>**

**</Form.Group>**

**<Button variant="primary" type="submit">**

**Login**

**</Button>**

**</Form>**

**</Col>**

**</Row>**

**</Container>**

**);**

**};**

**export default Login;**

**Profile.js**

**import React, { useState, useEffect, useContext } from 'react';**

**import { AuthContext } from '../context/AuthContext';**

**import { Form, Button, Container } from 'react-bootstrap';**

**import { userService } from '../services/userService';**

**const Profile = () => {**

**const { user, setUser } = useContext(AuthContext);**

**const [profile, setProfile] = useState({**

**name: '',**

**bio: '',**

**location: '',**

**});**

**useEffect(() => {**

**if (user) {**

**setProfile(user.profile);  // Populate form with user data**

**}**

**}, [user]);**

**const handleInputChange = (e) => {**

**const { name, value } = e.target;**

**setProfile({ ...profile, [name]: value });**

**};**

**const handleSubmit = async (e) => {**

**e.preventDefault();**

**try {**

**const updatedUser = await userService.updateUserProfile(profile);**

**setUser({ ...user, profile: updatedUser.profile });  // Update context**

**alert('Profile updated successfully');**

**} catch (error) {**

**console.error('Error updating profile', error);**

**}**

**};**

**return (**

**<Container>**

**<h1 className="my-4">Profile</h1>**

**<Form onSubmit={handleSubmit}>**

**<Form.Group controlId="formName">**

**<Form.Label>Name</Form.Label>**

**<Form.Control**

**type="text"**

**name="name"**

**value={profile.name}**

**onChange={handleInputChange}**

**/>**

**</Form.Group>**

**<Form.Group controlId="formBio">**

**<Form.Label>Bio</Form.Label>**

**<Form.Control**

**as="textarea"**

**rows={3}**

**name="bio"**

**value={profile.bio}**

**onChange={handleInputChange}**

**/>**

**</Form.Group>**

**<Form.Group controlId="formLocation">**

**<Form.Label>Location</Form.Label>**

**<Form.Control**

**type="text"**

**name="location"**

**value={profile.location}**

**onChange={handleInputChange}**

**/>**

**</Form.Group>**

**<Button variant="primary" type="submit">**

**Save Changes**

**</Button>**

**</Form>**

**</Container>**

**);**

**};**

**export default Profile;**

**services**

**agentServices.js**

**import axios from 'axios';**

**const API\_URL = '/api/agents'; // Assuming this is the backend route**

**// Get agent details for a user**

**const getAgent = async (userId) => {**

**const response = await axios.get(`${API\_URL}/${userId}`);**

**return response.data;**

**};**

**// Update agent details for a user**

**const updateAgent = async (userId, agentData) => {**

**const response = await axios.put(`${API\_URL}/${userId}`, agentData);**

**return response.data;**

**};**

**export const agentService = {**

**getAgent,**

**updateAgent**

**};**

**connectionService.js**

**import axios from 'axios';**

**const API\_URL = '/api/connections'; // Assuming this is the base URL for connection routes**

**// Create a new connection between two users**

**const createConnection = async (connectionData) => {**

**const response = await axios.post(API\_URL, connectionData);**

**return response.data;**

**};**

**// Get all connections for a user**

**const getConnections = async (userId) => {**

**const response = await axios.get(`${API\_URL}/${userId}`);**

**return response.data;**

**};**

**// Update the status of a connection (e.g., from pending to connected)**

**const updateConnectionStatus = async (connectionId, status) => {**

**const response = await axios.put(`${API\_URL}/${connectionId}`, { status });**

**return response.data;**

**};**

**export const connectionService = {**

**createConnection,**

**getConnections,**

**updateConnectionStatus,**

**};**

**messageService.js**

**import axios from 'axios';**

**const API\_URL = '/api/messages'; // Assuming this is the base URL for message routes**

**// Send a message between users**

**const sendMessage = async (messageData) => {**

**const response = await axios.post(API\_URL, messageData);**

**return response.data;**

**};**

**// Get all messages between two users**

**const getMessages = async (userId, receiverId) => {**

**const response = await axios.get(`${API\_URL}/${userId}/${receiverId}`);**

**return response.data;**

**};**

**export const messageService = {**

**sendMessage,**

**getMessages,**

**};**

**userService.js**

**import axios from 'axios';**

**const API\_URL = '/api/users'; // Assuming this is the backend route**

**// Login user**

**const loginUser = async (userData) => {**

**const response = await axios.post(`${API\_URL}/login`, userData);**

**return response.data;**

**};**

**// Register user (if needed for registration)**

**const registerUser = async (userData) => {**

**const response = await axios.post(`${API\_URL}/register`, userData);**

**return response.data;**

**};**

**export const userService = {**

**loginUser,**

**registerUser**

**};**

**App.js**

**import React from 'react';**

**import { BrowserRouter as Router, Route, Routes, Navigate } from 'react-router-dom';**

**import { AuthProvider, AuthContext } from './context/AuthContext';**

**import { AgentProvider } from './context/AgentContext';**

**import Dashboard from './pages/Dashboard';**

**import Profile from './pages/Profile';**

**import Login from './pages/Login';**

**import { Navbar, Nav } from 'react-bootstrap';**

**import 'bootstrap/dist/css/bootstrap.min.css';  // Bootstrap styles**

**const App = () => {**

**return (**

**<AuthProvider>**

**<AgentProvider>**

**<Router>**

**<AppNavbar />**

**<Routes>**

**<Route path="/login" element={<Login />} />**

**<Route path="/dashboard" element={<PrivateRoute component={<Dashboard />} />} />**

**<Route path="/profile" element={<PrivateRoute component={<Profile />} />} />**

**<Route path="\*" element={<Navigate to="/login" />} />**

**</Routes>**

**</Router>**

**</AgentProvider>**

**</AuthProvider>**

**);**

**};**

**// Navbar for the app**

**const AppNavbar = () => {**

**const { user, logout } = React.useContext(AuthContext);**

**return (**

**<Navbar bg="dark" variant="dark" expand="lg">**

**<Navbar.Brand href="/">FableLink</Navbar.Brand>**

**<Navbar.Toggle aria-controls="basic-navbar-nav" />**

**<Navbar.Collapse id="basic-navbar-nav">**

**<Nav className="ml-auto">**

**{user ? (**

**<>**

**<Nav.Link href="/dashboard">Dashboard</Nav.Link>**

**<Nav.Link href="/profile">Profile</Nav.Link>**

**<Nav.Link href="/" onClick={logout}>Logout</Nav.Link>**

**</>**

**) : (**

**<Nav.Link href="/login">Login</Nav.Link>**

**)}**

**</Nav>**

**</Navbar.Collapse>**

**</Navbar>**

**);**

**};**

**// Private route component to protect routes**

**const PrivateRoute = ({ component }) => {**

**const { user } = React.useContext(AuthContext);**

**if (!user) {**

**return <Navigate to="/login" replace />;**

**}**

**return component;**

**};**

**export default App;**