// =================================================================  
// Prompts 14-19: E-commerce, Security, Support, Ratings, i18n, & Deployment  
// =================================================================  
  
// This capstone section adds enterprise-level features.  
// It requires new dependencies:  
// npm install i18next react-i18next @node-rs/argon2 crypto-js jest supertest  
// devDependencies: @babel/core @babel/preset-env @babel/preset-react babel-jest  
  
// --- /server/models/AuditLog.model.js (Security & Compliance) ---  
/\*  
const mongoose = require('mongoose');  
const { Schema } = mongoose;  
  
const auditLogSchema = new Schema({  
 userId: { type: Schema.Types.ObjectId, ref: 'User' },  
 action: { type: String, required: true, enum: ['login', 'bid\_submitted', 'deal\_finalized', 'kyc\_approved'] },  
 details: { type: Object }, // e.g., { requestId: '...', bidId: '...' }  
 ipAddress: { type: String },  
}, { timestamps: true });  
  
const AuditLog = mongoose.model('AuditLog', auditLogSchema);  
module.exports = AuditLog;  
\*/  
  
// --- /server/models/SupportTicket.model.js (Customer Support) ---  
/\*  
const mongoose = require('mongoose');  
const { Schema } = mongoose;  
  
const supportTicketSchema = new Schema({  
 userId: { type: Schema.Types.ObjectId, ref: 'User', required: true },  
 requestId: { type: Schema.Types.ObjectId, ref: 'InsuranceRequest' },  
 subject: { type: String, required: true },  
 description: { type: String, required: true },  
 status: { type: String, enum: ['open', 'in\_progress', 'resolved'], default: 'open' },  
 messages: [{  
 senderId: { type: Schema.Types.ObjectId, ref: 'User' },  
 text: { type: String },  
 timestamp: { type: Date, default: Date.now }  
 }]  
}, { timestamps: true });  
  
const SupportTicket = mongoose.model('SupportTicket', supportTicketSchema);  
module.exports = SupportTicket;  
\*/  
  
  
// --- /server/middleware/audit.middleware.js (Logging Actions) ---  
/\*  
const AuditLog = require('../models/AuditLog.model');  
  
const logAction = (action) => {  
 return async (req, res, next) => {  
 // Let the route handler run first  
 res.on('finish', async () => {  
 if (res.statusCode < 400) { // Log only successful actions  
 const log = new AuditLog({  
 userId: req.user ? req.user.id : null,  
 action,  
 details: {  
 route: req.originalUrl,  
 method: req.method,  
 params: req.params,  
 body: req.body // Be careful not to log sensitive data like passwords  
 },  
 ipAddress: req.ip  
 });  
 await log.save();  
 }  
 });  
 next();  
 };  
};  
  
module.exports = logAction;  
\*/  
  
// --- /server/utils/encryption.js (AES-256 Encryption for Sensitive Data) ---  
/\*  
const crypto = require('crypto');  
const ENCRYPTION\_KEY = process.env.ENCRYPTION\_KEY; // Must be 256 bits (32 characters)  
const IV\_LENGTH = 16; // For AES, this is always 16  
  
function encrypt(text) {  
 let iv = crypto.randomBytes(IV\_LENGTH);  
 let cipher = crypto.createCipheriv('aes-256-cbc', Buffer.from(ENCRYPTION\_KEY), iv);  
 let encrypted = cipher.update(text);  
 encrypted = Buffer.concat([encrypted, cipher.final()]);  
 return iv.toString('hex') + ':' + encrypted.toString('hex');  
}  
  
function decrypt(text) {  
 let textParts = text.split(':');  
 let iv = Buffer.from(textParts.shift(), 'hex');  
 let encryptedText = Buffer.from(textParts.join(':'), 'hex');  
 let decipher = crypto.createDecipheriv('aes-256-cbc', Buffer.from(ENCRYPTION\_KEY), iv);  
 let decrypted = decipher.update(encryptedText);  
 decrypted = Buffer.concat([decrypted, decipher.final()]);  
 return decrypted.toString();  
}  
  
module.exports = { encrypt, decrypt };  
\*/  
  
  
// --- /server/routes/providers.routes.js (Provider Profiles & Search) ---  
/\*  
const express = require('express');  
const router = express.Router();  
const User = require('../models/User.model');  
  
// @route GET /api/providers  
// @desc Search/filter for providers  
// @access Public  
router.get('/', async (req, res) => {  
 const { expertise, minRating } = req.query;  
 const filter = { role: 'provider', kycStatus: 'verified' };  
  
 if (expertise) filter['profile.expertise'] = expertise;  
 if (minRating) filter['profile.avgRating'] = { $gte: Number(minRating) };  
  
 try {  
 const providers = await User.find(filter).select('-password -emailVerificationToken');  
 res.json(providers);  
 } catch (error) {  
 res.status(500).json({ message: 'Server error' });  
 }  
});  
  
// @route GET /api/providers/:id  
// @desc Get a provider's public profile  
// @access Public  
router.get('/:id', async (req, res) => {  
 try {  
 const provider = await User.findById(req.params.id)  
 .select('profile.companyName profile.expertise profile.avgRating');  
 // You would also populate completed deals here from the Bid/Request models  
 if (!provider || provider.role !== 'provider') {  
 return res.status(404).json({ message: 'Provider not found.' });  
 }  
 res.json(provider);  
 } catch (error) {  
 res.status(500).json({ message: 'Server error' });  
 }  
});  
  
module.exports = router;  
\*/  
  
  
// --- /server/routes/reviews.routes.js (Feedback and Ratings) ---  
/\*  
const express = require('express');  
const router = express.Router();  
const { protect, authorize } = require('../middleware/auth.middleware');  
const User = require('../models/User.model');  
const InsuranceRequest = require('../models/InsuranceRequest.model');  
  
// @route POST /api/reviews/:providerId  
// @desc Client leaves a review for a provider after a deal  
// @access Private (Client)  
router.post('/:providerId', protect, authorize('client'), async (req, res) => {  
 const { rating, comment, requestId } = req.body;  
  
 try {  
 // Verify that the client was part of this completed deal  
 const deal = await InsuranceRequest.findOne({  
 \_id: requestId,  
 clientId: req.user.id,  
 status: 'paid' // Or whatever the final status is  
 });  
 if (!deal) {  
 return res.status(403).json({ message: 'You can only review providers from completed deals.' });  
 }  
  
 const provider = await User.findById(req.params.providerId);  
 provider.profile.ratings.push({ raterId: req.user.id, rating, comment });  
  
 // Recalculate average rating  
 const totalRating = provider.profile.ratings.reduce((acc, r) => acc + r.rating, 0);  
 provider.profile.avgRating = totalRating / provider.profile.ratings.length;  
  
 await provider.save();  
 res.status(201).json(provider.profile);  
 } catch (error) {  
 res.status(500).json({ message: 'Server error', error: error.message });  
 }  
});  
\*/  
  
  
// --- /client/src/i18n.js (Localization Setup) ---  
/\*  
import i18n from 'i18next';  
import { initReactI18next } from 'react-i18next';  
  
const resources = {  
 en: {  
 translation: {  
 "dashboard\_title": "My Insurance Requests",  
 "total\_coverage": "Total Coverage: "  
 }  
 },  
 fr: {  
 translation: {  
 "dashboard\_title": "Mes Demandes d'Assurance",  
 "total\_coverage": "Couverture Totale: "  
 }  
 }  
};  
  
i18n  
 .use(initReactI18next)  
 .init({  
 resources,  
 lng: "en", // default language  
 interpolation: {  
 escapeValue: false  
 }  
 });  
  
export default i18n;  
\*/  
  
  
// --- /tests/bids.test.js (Example Integration Test) ---  
/\*  
const request = require('supertest');  
const app = require('../server/server'); // Assuming server.js exports the app  
const mongoose = require('mongoose');  
  
describe('Bidding API', () => {  
 beforeAll(async () => {  
 await mongoose.connect(process.env.MONGO\_URI\_TEST);  
 });  
  
 afterAll(async () => {  
 await mongoose.connection.close();  
 });  
  
 it('should not allow an unverified provider to bid', async () => {  
 // 1. Create an unverified provider and get a token  
 // 2. Create an insurance request  
 // 3. Make a POST request to /api/requests/:requestId/bids with the token  
 const res = await request(app)  
 .post('/api/requests/someRequestId/bids')  
 .set('Authorization', `Bearer some\_unverified\_provider\_token`)  
 .send({ bidAmount: 10000 });  
   
 expect(res.statusCode).toEqual(403);  
 expect(res.body.message).toContain('KYC verification');  
 });  
});  
\*/  
  
  
// --- /.github/workflows/deploy.yml (CI/CD Pipeline) ---  
/\*  
name: Deploy to AWS  
on:  
 push:  
 branches:  
 - main  
jobs:  
 build-and-deploy:  
 runs-on: ubuntu-latest  
 steps:  
 - name: Checkout code  
 uses: actions/checkout@v2  
 - name: Setup Node.js  
 uses: actions/setup-node@v2  
 with:  
 node-version: '18'  
 - name: Install dependencies  
 run: |  
 npm install  
 npm install --prefix server  
 npm install --prefix client  
 - name: Build client  
 run: npm run build --prefix client  
 - name: Run tests  
 run: npm test  
 - name: Deploy to AWS  
 uses: aws-actions/configure-aws-credentials@v1  
 with:  
 aws-access-key-id: ${{ secrets.AWS\_ACCESS\_KEY\_ID }}  
 aws-secret-access-key: ${{ secrets.AWS\_SECRET\_ACCESS\_KEY }}  
 aws-region: us-east-1  
 - name: Deploy to S3 and EC2  
 run: |  
 # Example deployment script  
 # aws s3 sync client/build s3://your-s3-bucket-name  
 # ssh -i your-key.pem ec2-user@your-ec2-ip 'cd /path/to/app && git pull && npm install --prefix server && pm2 restart server'  
\*/  
  
  
// =================================================================  
// Prompts 10, 11, & 12: Bid Management, Payments, & Analytics  
// =================================================================  
  
// (Code from previous turn remains here)  
  
// =================================================================  
// Prompts 7, 8, & 9: Chat, Asset Management, & Dashboards  
// =================================================================  
  
// (Code from previous turn remains here)  
  
// =================================================================  
// Prompt 5 & 6: Insurance Request Posting & Flexible Bidding  
// =================================================================  
  
// (Code from previous turn remains here)  
  
// =================================================================  
// Prompt 4: KYC Verification for Providers  
// =================================================================  
  
// (Code from previous turn remains here)  
  
// =================================================================  
// Prompt 3: User Authentication (JWT, RBAC, Email Verification)  
// =================================================================  
  
// (Code from previous turn remains here)  
  
// =================================================================  
// Prompt 2: Database Schema Design (MongoDB with Mongoose)  
// =================================================================  
  
// (Code from previous turn remains here)  
  
// =================================================================  
// Prompt 1: Project Setup  
// =================================================================  
  
// (Code from previous turn remains here)