Senior Fullstack Developer interview tasks

Description: You will be working on a microservice-based system that integrates local LLMs fine-tuned with contextual data. These tasks are designed to evaluate your backend architecture skills, fullstack fluency, and ability to work with local AI models.

All 3 tasks are mandatory.

Format: Complete the tasks at home and record a video explaining your implementation, architecture, and key decisions.

Please include a screen recording of yourself implementing at least one of the tasks.

Task 1: Microservice Design and Implementation

Goal: Design and implement a TypeScript-based microservice that exposes a REST API for managing product metadata.

Tech Stack

• Node.js + TypeScript
• Express.js or Fastify or other framework
• SQLite or Postgres or other database
• Jest or equivalent test framework

Requirements

• Create a REST API with full CRUD for Product (ID, name, description, tags, price).
• Expose API documentation using Swagger/OpenAPI.
• Implement validation, error handling, and basic logging.
• Include: docker-compose.yml, tests, and README.md

Service Description

The Product Metadata Microservice manages structured data about products, enabling integration with search, recommendation, and AI enrichment services.

Entity: Product

Fields:
- id: UUID (auto-generated)
- name: string (max 255, required)
- description: string (max 2000, required)
- tags: string[]
- price: number (positive, required)
- createdAt: timestamp
- updatedAt: timestamp

Endpoints:
- POST /products
- GET /products/:id
- PUT /products/:id
- DELETE /products/:id
- GET /products

Bonus Ideas:
- Soft deletes
- Category or brand field
- Cursor-based pagination
- Audit logging

Task 2: Fullstack Feature Build — Admin Dashboard

Goal: Build a an admin panel to manage products using the backend from Task 1.

Tech Stack

• Node.js + TypeScript
• TailwindCSS
• React Query or SWR
• Token-based auth

Requirements

• Create a responsive admin UI to list, create, edit, and delete products.
• Validate forms and show error messages.
• Implement simple token-based auth.
• Use Docker to run frontend and backend together.

UI Feature Description

The admin dashboard should have the following views:
- Product List Page: Fetch all products and display them in a table.
- Product Form Page: Create/edit form with validation for all fields.
- Delete Confirmation: Modal or inline action to delete a product.
- Optional: Add filter/search by name or tag.

Frontend should consume REST API from Task 1 and provide basic responsive layout suitable for mobile/tablet/desktop.

Task 3: Real Local Model Integration — LLM-based Tag Suggestion

Goal: Integrate a real local AI model to automatically suggest product tags based on name + description.

Suggestions

Suggestion A: Ollama + LLM
- Install Ollama and run model (e.g., mistral)
- Call Ollama API with prompt to get tags

Suggestion B: Sentence-transformers + Semantic Matching
- Use Python with sentence-transformers to embed and match tags
- Return top 3 suggestions based on similarity

Requirements

• Implement /suggest-tags endpoint in backend.
• Input: { name: string, description: string }
• Output: { suggestedTags: string[] }
• Must use real local model (no cloud)
• Optional: Integrate this in admin dashboard with an 'Auto-Suggest Tags' button

Inference Service Description

You will simulate a production use case where product data is enriched via local AI. This can be done via:
- A local HTTP service that wraps an LLM (e.g., Mistral running on Ollama)
- Or a local embedding engine that matches semantic similarity

Make the module modular and replaceable. Document all steps clearly. You are expected to demonstrate the model working locally.