Senior Full-Stack Developer Coding Task

Complete Task Description

Introduction

As a Senior Full-Stack Engineer, you will develop a small, complete order management system with a REST API backend and React.js frontend in this task.

The goal is to assess your skills in API design, backend architecture, frontend structuring, and your approach to technical topics. Clean software design and comprehensible decisions are the key focus.

Project Objective

Build a system that enables simple order management. Users should be able to create and manage products and create orders from them. The frontend communicates via a clearly defined REST API with the backend.

Requirements

1. Product Management

- Create, update, delete, and list products
- Product attributes:
  - `id` (UUID or autoincrement)
  - `name` (String)
  - `price` (Number)
  - `stock` (Number)

2. Order Management

- An order contains one or more products (each with quantity)
- Total price is calculated server-side
- List orders and display details

3. Users (Simplified)

- No full user management required
- Use a fixed user ID or several static demo users possible

Technical Conditions

- **Backend:** Java (e.g. Quarkus, Spring Boot) or TypeScript (e.g. Node.js, NestJS or Express)
- Layered architecture preferred: Onion or Hexagonal
- REST API: Correct resource paths, status codes, well documented (e.g. OpenAPI/Swagger)
- Data storage: In-memory, file or simple DB (e.g. SQLite/H2)
- **Frontend:** React.js with functional components
- Clear interface for product and order management
- Clear component structure (e.g. Pages/Features/UI)
- API connection (e.g. via Fetch/Axios)

Evaluation Criteria

- Architecture and code quality (clean separation of layers, modularization, naming)
- API design (consistency, resources, status codes)
- Use of design patterns (layering, dependency injection, DTOs, DDD approaches)
- Frontend implementation (structure, reusability, API connection)
- Problem-solving approach (handling unclear points, documenting assumptions)
- Communication (README, design explanations)

Quality and architecture are more important than maximum functionality.

Use of AI Coding Agents

As part of this task, you may and should deliberately use AI Coding Agents (e.g. GitHub Copilot, Claude, Cursor, OpenAI GPT, Codeium, Tabnine) to support coding, architecture decisions, and technical research. INFORM GmbH values the meaningful and reflective use of modern AI technologies.

Recommended AI Use Cases

- Code and boilerplate auto-completion
- Generation of test cases or sample data
- Support for API design and documentation
- Implementation of specific features or architecture patterns (e.g. layering, DTOs)
- Research of technical concepts and frameworks
- Code review and refactoring (e.g. identifying weaknesses)

Note on AI Usage

- **Document:** Explain in the README where AI agents were used and how they supported your solution process.
- **Reflect:** Assess the quality and reliability of AI tools were tasks solved or did weaknesses remain?
- **Transparency:** State where you made manual adjustments or corrections.
- **Independence:** The final solution must remain comprehensible, maintainable, and understandable for others.

Additional Evaluation Criteria

- You show that you can use AI-powered tools sensibly, critically, and responsibly.
- You are able to recognize the limits and risks of AI coding agents.
- The solution path remains understandable for humans, even if parts are generated by AI.

Deliverables & Timeline

Working period: Recommended: 6 - 8 hours (can be distributed over multiple days).

Submission: Please submit your solution as a public Git repository (GitHub, GitLab, etc.)

1. **Complete source code** (backend + frontend) with the following structure

   ```
   /frontend  React UI
   /backend  API service
   /README.md  Architecture, setup, design decisions
   ```

2. **README content:**
   - Architecture overview (short: diagram or text)
   - Setup instructions (how to start frontend and backend)

- Summary of key design decisions
- (Optional) Suggestions for further development

3. **Postman Collection** or OpenAPI Spec for API testing

Bonus (optional)

- Pagination or search function in the product list
- Basic unit tests for domain or API components
- CI/CD configuration (e.g. GitHub Actions)
- TypeScript type sharing between frontend and backend