Coding Assignment for Sr. Software Engineer

Vibe Coding Assignment: Multi-Language Microservices Web Application

---

Objective:

TO: Design and implement a production-grade, full-stack solution **using GenAI prompt coding** (Vibe coding / Context Engineering) techniques with a focus on scalability, modularity, and cross-technology integration

SO: We can demonstrate our comfort and readiness to join iH's product development team.

Notes before starting:

- **Recommendation:** We are specifically seeking clear architecture and components integration and well-designed entity relationships optimized for graph database queries, rather than a very polished UI.
- **Estimated Time:** ~4 hours

Key Results:

1. **Choose a model**:
    1. Movie recommendation system
    2. E-commerce product catalog and order tracking
    3. Ride-sharing dispatch and matching platform
    4. Fitness and wellness activity tracker
    5. Event management and attendee networking system

2. **Architect & Build:**
    1. Design Requirements:
        1. Frontend: web-based using modern framework (eg: React, Angular, Vue, Svelte, etc.) that interacts with backend services through RESTful APIs.
        2. Backend Microservices: at least two (2) written in different languages (eg: Node.js + Go, Python + Rust, Java + Kotlin).
        3. Relational Database - for transactional data storage (eg: PostgreSQL, MySQL, SQLite).
        4. Graph Database - for analytics and relationship queries (e.g., Neo4j, ArangoDB, Dgraph).

            5. REST API - one (1) exposed by A microservice and consumed by B microservice.
            6. Event-drive Microservice - one (1) message queue or topic-based communication system that publishes between at least two services (eg: Kafka, RabbitMQ, etc).
            7. Data - Published data persists in both the relational and graph databases simultaneously.
            8. Containers - Docker + Docker-Compose can orchestrate full system.

    2. Considerations and Constraints:

        1. All services must be independently buildable and runnable.
        2. Frontend must communicate exclusively through defined APIs.
        3. System capable of error handling and basic test coverage.
        4. Data consistency between OLTP and OLAP layers for published messages
        5. System runs successfully with a single docker-compose up command.
        6. Frontend must implement optimistic updates and clearly indicate to the user when content processing is pending.
        7. Both databases can be queried independently for valid, consistent data.
        8. All services include container definitions that successfully interoperate in Docker.
        9. Graceful handling of failures (e.g., queue retries, transaction rollback, or error response management).

3. **Document** (in README.md) - clearly explaining:
    1. Architecture overview.
    2. How to build, run, and test system locally.

Deliverable:

- **LINK - to fully functioning solution meeting above requirements via** docker-compose **for review**