AllRide GmbH
Hackescher Markt 2,
10178 Berlin
E-Mail: info@allride.io
https://allride.io

Coding Challenge: Bulk User Import System

Hello Candidate,

Congratulations on moving to the next stage! This coding challenge is designed to assess your full-stack development skills, particularly with Kotlin, React, and event-driven architectures, which are crucial for our upcoming Subscription Engine re-architecture.

Challenge Overview:

Your task is to build a small application that allows for the bulk import of user data from a CSV file. The system should consist of a simple frontend to upload the file and a backend service to process and store the user data. A publish/subscribe mechanism should be used to decouple the initial file upload from the actual data processing.

Time Allotment: 4-6 hours. Please prioritize functionality and clean code.

Task Requirements:

1. **Frontend (React):**
    - Create a simple web interface where a user can select and upload a CSV file.
    - The CSV file will contain user data (e.g., `id`, `firstName`, `lastName`, `email`). You can define a simple structure.
    - Upon successful upload, the frontend should provide feedback to the user (e.g., "File uploaded successfully, processing started").
    - *Bonus (if time permits):* Basic validation of the file type (e.g., ensure it's a .csv).
2. **Backend (Kotlin):**
    - **API Endpoint:** Create an API endpoint that accepts the CSV file from the frontend.
    - **File Storage:** Temporarily store the uploaded CSV file on the server.
    - **Publisher:** After receiving the file, the backend should publish an event (e.g., `FileUploadedEvent`) to a message queue. This event should contain necessary information like the path to the uploaded file.
    - **Subscriber/Worker:** Implement a separate component or service that subscribes to `FileUploadedEvent`.
        - This subscriber should read the CSV file.
        - Parse the data from the CSV.
        - "Store" the user data. For this challenge, you can simulate storage by logging the parsed user data to the console or storing it in an in-memory collection (e.g., a List or Map). *A full database setup is not required for this exercise.*
        - Implement basic error handling for CSV parsing (e.g., log an error for invalid rows but continue processing others).
3. **Publish/Subscribe System:**
    - Utilize a publish/subscribe mechanism. While we use **GCP Pub/Sub** in production, for this challenge, you can:
        - Use a local in-memory message queue (e.g., using Kotlin Coroutines Channels or a simple event bus library).
        - *Alternatively, if you are comfortable and it fits within the time, feel free to use a Dockerized version of a message broker like RabbitMQ or Kafka, or even a mock GCP Pub/Sub if you have a preferred way of doing so locally.* Please document your choice.
4. **API Design:**
    - Design your API endpoint(s) following a **domain-driven design** approach where appropriate for this small scope. Think about the resources and actions involved.

AllRide GmbH
Hackescher Markt 2,
10178 Berlin
E-Mail: info@allride.io
https://allride.io

Tech Stack:

- **Backend:** Kotlin (You can use a framework like Spring Boot, Ktor, or Micronaut – your choice, but please state it).
- **Frontend:** React.
- **Messaging:** An event-driven approach as described above.
- **Build Tools:** Your choice (e.g., Gradle/Maven for Kotlin, npm/yarn for React).

Deliverables:

1. **Source Code:**
    - Push your complete solution (both frontend and backend) to a new public repository on your personal GitHub account.
    - Provide clear instructions in a `README.md` file on how to build and run your application. This should include:
        - Prerequisites (e.g., Java version, Node.js version).
        - Steps to build the backend and frontend.
        - Steps to run both services.
        - A brief explanation of your design choices, especially regarding the pub/sub implementation and any trade-offs you made due to the time constraint.
        - An example CSV file structure you expect.
2. **Link to GitHub Repository:** Share the link to the GitHub repository with us.

Evaluation Criteria:

- **Functionality:** Does the application meet the core requirements?
- **Code Quality:** Is the code clean, well-structured, and readable?
- **Backend Design:** Soundness of the backend logic, API design, and event-driven implementation.
- **Frontend Implementation:** Basic functionality and interaction.
- **Error Handling:** How gracefully does the application handle potential issues (e.g., file parsing errors)?
- **Understanding of Concepts:** Demonstration of understanding of Kotlin, React, APIs, and event-driven architecture.
- **Documentation:** Clarity of the `README.md` file.

We understand that 4-6 hours is a limited time, so focus on delivering a working solution that demonstrates your core skills. It's better to have a smaller, well-executed solution than a larger, incomplete one.

Good luck! We're excited to see your solution.

Best regards,

The AllRide Hiring Team