Overview

You will build a Java-based microservice (using any Java framework - for example Spring Boot) that tracks "live" sports events and, for each live event, periodically (every 10 seconds) calls an external REST endpoint, transforms the response into a message, and publishes it to a message broker (for example Kafka). You are free to use any libraries, tools or frameworks - and you're encouraged to leverage AI assistants (e.g., ChatGPT, GitHub Copilot) - but you must review, validate, and document any AI-generated output.

It is expected for you to spend around 90 minutes to complete the exercise.

Requirements

- Expose a REST endpoint to receive event status updates (live ↔ not live).
- For each event marked live, schedule a task to call an external REST API every 10 seconds.
    - For the API you can assume that the API returns a json object with the following structure:

```
{
   "eventId": "1234",
   "currentScore": "0:0"
}
```

- Transform the API response into a message and publish it to a topic (for example using Kafka).
- Implement basic error handling and logging.
- Deliver a working prototype along with documentation of your design choices, any AI usage, guide for running.

Delivery

- **Executable Solution:** Your solution must be executable.
- **GitHub Link:** Provide a link to a public Git repository containing your code.
- **README.md:** Must include:
    - Setup & run instructions.

- How to run any included tests.
- A summary of your design decisions.
- Documentation of any AI-assisted parts (what was generated, how you verified/improved it).

Detailed Task Description

A. Event Status Endpoint

a. Implement POST /events/status (or equivalent) that accepts a JSON payload with:
   i. eventId (string or number)
   ii. status (boolean or enum: “live” / “not live”)

b. Validate input and update in-memory state.

B. Periodic REST Calls

c. For each event in “live” state, schedule a job that fires every 10 seconds.

d. Each job should call an external REST API (hardcoded or configurable endpoint).
   i. API can be mocked or implemented as a simple separate service

e. Use the returned data to build a message payload.

C. Message Publishing

a. Publish the payload to a RocketMQ or Kafka topic.

b. Include retry logic for transient failures.

c. Log successes and failures appropriately.

D. Error Handling & Logging

a. Handle errors in external calls and message publishing.

b. Log key events, errors, and state changes for observability.

E. Testing

a. Provide unit and/or integration tests covering:
   i. Status updates
   ii. Scheduled calls
   iii. Message publication under normal and error conditions