Lead Backend Engineer: Coding Challenge

Overview

This challenge evaluates your ability to translate business requirements into code by implementing an event-driven microservice.

Task Description

Implement a service satisfying the challenge requirements. We ask that you complete the challenge using Python 3.10+. The solution must use the provided `topanga_queries` CRUD helpers to interact with the provided SQLite database engine that contains Rental, Asset, and User records. You are free to change any code in this dependency as you see fit, but it is not necessary for completing this challenge.

A **Rental** is a record that associates an **Asset** (reusable container) with a **User** in order to keep them accountable for returns. A rental is created with status `IN_PROGRESS` when a user checks out an asset and its status is updated to `COMPLETED` when an eligible asset is returned before the rental expires.

This service handles Rental return events. When a user returns an asset, their unique QR code (encoded with their `user_id`) and the asset's QR code sticker (encoded with the `asset_id`) are scanned at a return bin - emitting the event containing the QR data, timestamp, and `location_id` of the return bin. The QR data are base64 encoded UTF-8 strings of their corresponding IDs.

```javascript
// Example event:
{
      "timestamp": "2025-02-03T12:00:00",
      "location_id": "topanga-location-01",
      "user_qr_data": "dHBnX3UxMjM0NTY3ODkw",
      "asset_qr_data": "dHBnX2EwMDAwMDAwMDAx"
}
```

A user can have multiple rentals `IN_PROGRESS` at once, and when they return one we want to mark the oldest non-expired rental for that user as `COMPLETED`. The Asset's `asset_type` must match one of the candidate Rental's `eligible_asset_types` to be eligible to complete it.

Topanga.io

The service must be executable by another Topanga engineer on their local machine with a single command or payload. Include clear instructions in the README on how to initialize any dependencies for it and invoke it with an event payload.

A rental return also triggers downstream effects like sending the Rental user a notification, updating their user stats, accumulating rewards etc. Document in the README how you might handle these side-effects in the service and/or architecture.

The provided `topanga_queries` dependency contains sample data in the SQLite database binary to test with and methods to refresh the data back to its initial state. The `example_events` directory also contains some test events (as JSON files) you can use to test your service. All of the test events are valid payloads - any errors raised by invoking your service with those should be handled in your service.

Deliverables

Your deliverable will be a .zip file containing:

1. The source code of the application
2. Dependencies & how to install them (e.g. requirements.txt with pip). If unit tests are included (optional), provide steps to run them.
3. A README.md text file providing
    1. A high-level overview of engineering choices made
    2. Execution instructions for running the application / how to invoke it with the sample events
    3. A list of TODOs to make the service production-ready
    4. Notes on how downstream side-effects for Rental returns may be handled
4. A cloud architecture diagram that maps out what the infrastructure may look like based on the prompt. Your system diagram should tell us what cloud provider you are assuming, what services from that provider you would utilize, and where this service sits in that stack when deployed.

Evaluation Criteria

The main criteria for evaluating your submission will be:

1. Quality of Deliverables
2. Engineering Choices
3. Working Solution