A Take-Home Programming Exercise

Solve the following questions, you may use whatever other open-source libraries you see fit to get the job done. Good luck and have fun!

Instructions

Before getting started on the challenge, let's get some logistics out of the way:

1. You should create a **private** repository on Github and add **@nagalakshmi-ramasubramanian, @bsonrisa, @sundeep (Referred as reviewers later on)** as collaborators.
2. Create an empty branch with a name of your choosing and push it up. You should push your commits to this branch.
3. Create a pull request to master from this branch and copy/paste the "Requirements" section into the description.
4. Add the reviewers when you are ready for us to review.

Please complete this exercise **within 3 days** of when you receive it.

Your codebase should have tests, as well as a README with setup instructions. Please add any input and output files to your pull request.

An Event Processing Pipeline

For this exercise you will be building a simplified ***advertisement event processing pipeline,*** in two parts. First you will need to implement an Event Collector, and then an Event validator.

Events

For the purposes of this exercise, events are immutable and of two types:

1) Serving events: generated during advertisement serving, and each event represents an advertisement sent to a user.
2) User events: Events generated during user interactions with advertisements.

These two different sources of events flow through and are collected in files.
Event properties:

1) Both types of events have a similar schema: **{eventId, eventTimestamp, eventType, parentEventId, userId, advertiserId, deviceId, price}.** This should be the column order in the output CSV.
2) Any of the field could be empty, but valid events must have eventId and eventTimestamp not null

3) Serving events eventType is null/empty, while for user events eventType will be one of "impression" or "click".
4) Serving events should also have parentEventId set to null/empty. The parentEventId in a user event represents the eventId of the corresponding serving event. Validate the events while receiving and storing in appropriate csv files

The pipeline is responsible for receiving these events via HTTP requests, storing them durably, and validating them.

Event Collector

Requirements

Implement a REST API in the language of your choice for a simplified event collector. This API will expose a single endpoint /event, to which events will be posted over HTTP for collection.

1) The API should be able to service requests in a **single-threaded manner**: extract payload, validate schema, persist to local disk as **comma separated values**
2) Every event that hits the endpoint must either be persisted or invalidated
3) Store serving and user events in separate files locally

Note: The eventId field will be unique. There will not be 2 events with the same eventId.

Example Input:
These GET requests, in sequence
1) ?eventId=event1&eventTimestamp=10:00&parentEventId=&userId=user1&advertiserId=adv1&deviceId=&price=10
2) ?eventId=event2&eventTimestamp=10:01&parentEventId=&userId=user2&advertiserId=adv2&deviceId=&price=10
3) ?eventId=event3&eventTimestamp=10:11&eventType=impression&parentEventId=event1&userId=user1&advertiserId=adv1&deviceId=dev1&price=
4) ?eventId=event4&eventTimestamp=10:00&eventType=&parentEventId=event1&userId=user1&advertiserId=adv1&deviceId=&price=
5) ?eventId=event5&eventTimestamp=10:11&eventType=click&parentEventId=event2&userId=user2&advertiserId=adv2&deviceId=&price=

PS: Events 1 and 2 are server events. Events 3 and 5 are user events. Event #4 is unparsable, because it has a parentEventId but no eventType. .

Expected output

server-events.csv
event1,10:00,,,user1,adv1,deviceId1,10

event2,10:01,,,user2,adv2,deviceId2,12

user-events.csv
event3,10:11,impression,event1,user1,adv1,dev1,,
event5,10:11,click,event2,user2,adv2,,,

Event Validation:

Requirements

1. Write a program to validate and enrich user events by comparing with server events.
2. Validated user events have the following conditions met:
    a. parentEventId matches an eventId in the server events.
    b. userId in both event types are equal
    c. eventTimestamp in server events occur before event timestamp of user event
    d. Dedupe for parentEventId and the event type. That is, the output shouldn't contain multiple events with the same parentEventId and eventType.
    e. Pick the earliest eventTimestamp of the user events while deduping
3. Enrich the valid user event, augmenting with the advertiserId and price fields from the matching server event. The user and server events from the previous question will server as the input to this validation
4. All events have the structure

{eventId,eventTimestamp,eventType, parentEventId, userId, advertiserId, deviceId, price}
This is the column order in the CSV.

5. The eventId field will be unique. There will not be 2 events with the same eventId.

Example Input

```
server_events= [[event1, 10:00, , , user1, adv1, deviceId1, 10],
                 [event2, 10:01, , , user2, adv2, deviceId2, 12]]
user_events= [[event3, 10:05, impression, event1, user1, adv1, deviceId1, 10], //real event
               [event4, 10:08, impression,  event1, user1, adv1, deviceId1, 10], //real event, duplicate/late
               [event5, 9:00, click, event1, user1, adv1, deviceId1, 10], //fake event
               [event6, 10:30, click, event1, user1, adv1, deviceId1, 10], //real event
               [event7, 10:05, impression, event1, user2, adv1, deviceId1, 10] //fake user, user 2 doesn't
have event1
               ]]
```

Feel free to randomly generate events for your testcase (eg, use scalagen in case you choose to write in scala), the events should be read from two different files and the output is written into a different file. Events are immutable, they cannot be modified at the source.

Expected output

validated_events.csv

```
event3, 10:05, event1, user1, impression, adv1, deviceid1, 10
event6, 10:30, event1, user1, click, adv1, deviceid1, 10
```