You’ve been selected for the next round! Thanks for taking the time to do this test. We do want to hire the best people and we want them to be happy joining us. For that, it should be a fit from both sides. And that’s why we want you to show us your skills.

This Test.

We do value your time, and this is a simple test. It shouldn’t take more than 3 hours. If it does, then you are doing something wrong and you should limit your self to the max 3 hours. You have 4 working days from the time you were sent the test to complete it.

The Problem: Events Processor

The Problem:

- We have a frontend mobile app for fitness training, sth like Nike Training App. We have a corresponding backend that manages users, training programs, user progress, etc.

- What we care about here is a micro-service that manages and process events the app pushes to our backend. This micro-service gets called via a `POST` operation on one of its endpoints **`/v1/event`**. The mobile app will call this api endpoint. This endpoint accepts an event object and process it.

Example: Events Processor

Our system will receive a continuous series of events like the following snapshot: Two users opening the app, playing and 1 cancelling a training program.

```
{
  "user_id": "foo2",
  "device_id": "bar2",
  "type": "app_launch",
  "time_stamp": "2021-02-28 16:22:52",
},

{
  "user_id": "foo1",
  "device_id": “bar1",
  "type": "training_program_started",
  "training_program_id": "2352",
  "training_program_title": "7 Minutes of HIIT Training",
  "time_stamp": "2021-02-28 16:23:25",
},

{
  "user_id": "foo2",
  "device_id": "bar2",
  "type": "training_program_started",
  "training_program_id": "2352",
  "time_stamp": "2021-02-28 16:52:00",
},

{
  "user_id": "foo2",
  "device_id": "bar2",
  "type": "training_program_cancelled",
  "training_program_id": "2352",
  "time_stamp": "2021-02-28 16:54:21",
},

{
  "user_id": "foo1",
  "device_id": "bar1",
  "type": "training_program_finished",
  "training_program_id": "2352",
  "time_stamp": "2021-02-28 16:59:43",
},

//etc
```

Technical Spec

- We want to improve the events object structure, naming convention, style, if any. Feel free to propose changes to the event object structure as you see fit and embed them into your solution design.
- Do setup the domain logic, resources and data model that are needed for this problem.
- We consider the events are chronologically ordered, meaning, the events come in order of their time_stamp.
- If a user finishes a training program that is more than 30 minutes long we want to call an api endpoint `/v1/notify` with a proper message to congratulate the user on doing his daily training dose mentioning the exact minutes he trained. Notifications are handled in another micro-service that we don't care about in this test. We only have to call `/v1/notify`
- If a user opens the app and don't start any training program within 10 minutes, we want to push a notification to `/v1/notify` encouraging him to start training now.
- This system should have the ability to handle unknown (unsupported) events gracefully. It should also handle new events in the future with minimum code changes. So, the design of this system, should keep this in mind.

Requirements and Assessment Scale

- You can use any framework or library of your choice. Preferably Python flask.
- Handling of notifications at `/notify` is not part of this test. You only have to call an endpoint.
- Setup a local server that can handle the RESTful API.
- There's no need to setup DB tables or use of DB tables at all. Use any temp data structure you like. Or mock a DB if you want.
- You should provide sufficient evidence that your solution is complete by indicating that it works correctly against the supplied test data.
- Use of TDD is extremely encouraged.
- While this problem a small task to solve, **we expect you to submit what you believe is a quality code. Code that you'd be able to run, maintain, and evolve;**
- We use a very clear assessment scale. We assess your solution according to the design you choose, your files structure, design patterns, anti-design patterns, and, equally important, its cleanness. Long functions and deep-nested ifs and whiles are signs of bad coding style. Good coding is very clear and clean visually, architecturally, and 90% of time, it doesn't need any comments.
- This is a cmd/console application. No UI is needed;