Assignment

Event Processing API

Problem

Build a backend service that implements an endpoint for transforming and storing event data, and another endpoint for reading the transformed output as JSON.

Endpoint 1: Transform and Store Event Data

Implement an HTTP endpoint that allows for negotiation of some mechanism for the client to stream data to the server (e.g. HTTP long polling or a WebSocket). The endpoint should accept the **session ID** as a parameter. The format of the ID is up to you. Once the stream is open, the server should accept messages from the client in the following JSON format:

```
[
  {
    "timestamp": 1569972082,
    "type": "SESSION_START",
    "session_id": "e6fa79ca-d142-4046-a134-5134f16a0b5e",
  },
  {
    "timestamp": 1569972084,
    "type": "EVENT",
    "name": "cart_load"
  },
  ...
]
```

- Each message is a **JSON array** containing 1 or more JSON objects.
- Each **JSON object** contained within the array represents a single event.
- The client can send an unlimited number of messages containing 1 or more event entries.
- All objects contain a `type` key. The permissible entry types are `SESSION_START`, `SESSION_END` and `EVENT`. `SESSION_{START,END}` mark the start and end of the session. An `EVENT` is a single event occurrence that is recorded as a point with a timestamp.

- All entries contain a `timestamp` key, which is UNIX epoch time in seconds. Realistically, we would want higher precision (at least milliseconds), but we will use seconds here to simplify the problem.
- *Some* entries contain a `session_id` key. This is an identifier that is used to match related entries. For example, a `SESSION_START` and a `SESSION_END` entry pair will have the same session ID. In the above example, we used a uuid4 format for the session ID, but this can be anything you choose.
- Some entry types contain additional keys that are specific to that event type. The documentation below on the different event types goes into more detail on this.
- **There is no ordering guarantee for entries**, except that `SESSION_START` will always be the first event, and `SESSION_END` will always be the last event. You could have events with timestamps that are not in chronological order.
- Once the `SESSION_END` event is received, the stream can be closed and the session is considered complete.

Trace Events

SESSION_START

```
{
     "timestamp": 1569972082,
     "type": "SESSION_START",
     "session_id": "e6fa79ca-d142-4046-a134-5134f16a0b5e",
]
```

- There are no additional entry-specific keys.
- `session_id` for the matching `SESSION_END` entry will be the same.
- Guaranteed to be the **first** entry in the session.

SESSION_END

```
{
     "timestamp": 1569972090,
     "type": "SESSION_END",
     "session_id": "e6fa79ca-d142-4046-a134-5134f16a0b5e",
}
```

- There are no additional entry-specific keys.
- `session_id` for the matching `SESSION_START` entry will be the same.
- Guaranteed to be the **last** entry in the trace.

EVENT

```
{
     "timestamp": 1569972083,
     "type": "EVENT",
     "name": "cart_loaded",
}
```

- There is one additional key, `name`, which is the name of the event.
- There is no `session_id`, since the event is just a single point.

Data Transformation

The service should transform the raw trace data into a form that is more readily consumable by a front-end. **You do not have to implement a front-end**. This transformed data representation has the following key criteria:

- Events are ordered in **chronological** order.
- Each "pair" of related events (`SESSION_{START,END}`) is grouped and represented using a single object with `start` and `end` timestamps, and optionally `children` representing events that belong to a parent -- the parent is the session, which is the root object.

Example

Let's take the following example of a complete input (which would have been delivered over the course of multiple streamed messages):

```
[
    {
        "timestamp": 1569972082,
        "type": "SESSION_START",
        "session_id": "e6fa79ca-d142-4046-a134-5134f16a0b5e",
    },
    {
        "timestamp": 1569972083,
        "type": "EVENT",
        "name": "cart_loaded"
```

```
    },
    {
        "timestamp": 1569972090,
        "type": "SESSION_END",
        "session_id": "e6fa79ca-d142-4046-a134-5134f16a0b5e",
    }
]
```

- The entries start with `SESSION_START` and end with `SESSION_END`, as defined by the spec.
- There is a single `EVENT`.

The transformed output should look like this:

```
{
    "type": "SESSION",
    "start": 1569972082,
    "end": 1569972090,
    "children": [
        {
           "type": "EVENT",
           "timestamp": 1569972083,
           "name": "cart_loaded"
        }
    ]
}
```

The events are ordered, grouped, and represented in a hierarchical structure.

Edge Cases

- There's an edge case where you might have multiple events with identical timestamps. In this scenario the sorting doesn't matter.
- There may be other edge cases not discussed here. If you find any, make sure to note them and document how they would be handled.

Data Storage

The transformed output is persisted and should be retrievable by the **session ID** that was sent when the client first started the stream of events. There is no requirement on *how* or *where* the data needs to be stored -- that's up to you.

Endpoint 2: Retrieve Transformed Data

Implement a second HTTP endpoint where you pass the **session ID** as a parameter, and the returned response is the transformed JSON representation of the session described above. Error cases should be handled appropriately (e.g. if there is no session matching the specified ID).

Other Requirements

- Use git for version control, the git history should be included with your project submission (i.e. the `.git` directory should be intact). Avoid putting the entire project in a single commit, ideally your git history should show the evolution of the project.
- Include any instructions necessary to set up/bootstrap the service.

There are *no constraints* on the resources you are allowed to use (feel free to search for anything you need) to solve this problem or how much time you can take.

Submission

Upon completion, please email your zipped project directory.