Slang

Challenge:
Software Engineer

A coding challenge for the Software Engineer role at Slang

Overview

This document describes a coding challenge for engineering applicants looking to work as Software Engineers with Slang. We expect that you spend a maximum of two hours to develop a solution that answers the problem listed below.

Feel free to submit your work quickly if you feel that you've adequately demonstrated your abilities. Be sure to balance code quality, performance and productivity, and keep in mind that we take attention to detail very seriously!

The task is based on a two-step real-world task that we've had to solve, and while it's been boiled down into specifications appropriate for a short development assessment, it is still representative of some of the work you'd be exposed to in this role. We suggest that you read through everything here — it's a quick read — before starting. And, as always, let us know if you have any questions or comments! I will be available to answer any questions at ricardo@slangapp.com at any point throughout the day/night.

Specifications

The descriptions here are intentionally a bit free-form — we'd like you to make most of the decisions yourself.

First Step

We'd like you to write a simple program that fetches some data from one of our REST APIs (you'll be given access before you start your challenge and we'll share authentication header needed), parses the received JSON to produce a new data structure, and posts the results to another endpoint.

The format of the endpoint you need to consume is the following:

Request

```
GET https://api.slangapp.com/challenges/v1/activities
Content-Type: application/json
Authorization: Basic MTpC__SAMPLE__VjPQ==    ← replace this with your key
```

Response

```
200
```


```
{"activities":
  [
    {
      "id": 198891,
      "user_id": "emr5zqid",
      "answered_at": "2021-09-13T02:38:34.117-04:00",
      "first_seen_at": "2021-09-13T02:38:16.117-04:00"
    },
    {
      "id": 43990,
      "user_id": "emr5zqid",
      "answered_at": "2021-09-13T02:42:07.117-04:00",
      "first_seen_at": "2021-09-13T02:41:51.117-04:00"
```

```
    },
    {
      "id": 37191,
      "user_id": "emr5zqid",
      "answered_at": "2021-09-14T00:31:36.117-04:00",
      "first_seen_at": "2021-09-14T00:31:25.117-04:00"
    },
    ...
}
```

Or in case of error:

`401 Unauthorized`

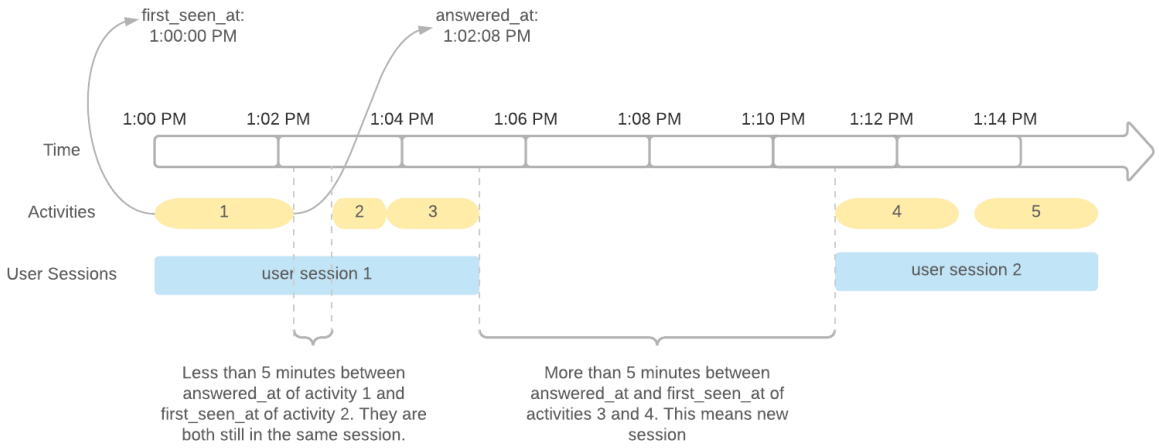
`400 Bad request`

`429 Too many requests` (en el caso de ser rate-limited)

You will get a JSON array containing a list of a few hundred “activities” in no particular order. Each activity object has its unique ID, the unique ID of the user that answered that activity, a first seen at timestamp and an answered at timestamp, both in UTC and ISO 8601 compliant.

Second Step

We’d like you to process this document so that we can get a list of user-sessions from these activities. A user-session is defined as the time between first_seen_at of the first user’s activity to the answered_at of the latest activity for a given user, as long as there isn’t more than 5 minutes between answered_at and first_seen_at. If more than 5 minutes have elapsed between these two dates, we consider this a new user session. For example:



These user session objects should:

- Be grouped by the user_id.
- Be ordered by their started_at.
- Have the list of activity ids that belong in the session
- Have a started_at (which corresponds to the first_seen_at of the first activity in the session)
- Have an ended_at (which corresponds to the answered_at of the last activity in the session)
- Have a duration in seconds (which should be the difference between the started_at and ended_at).
- Pay close attention to the format of the sample JSON object below!

Here's an example of an acceptable submission:

```
{
  "user_sessions": {
    "3pyg3scx": [
      {
        "ended_at": "2021-09-10T19:51:26.799-04:00",
        "started_at": "2021-09-10T19:22:23.799-04:00",
        "activity_ids": [
          251953,
```

```
      379044
    ],
    "duration_seconds": 173.0
  },
  {
    "ended_at": "2021-09-11T04:33:50.799-04:00",
    "started_at": "2021-09-11T04:05:20.799-04:00",
    "activity_ids": [
      296400,
      247727,
      461955
    ],
    "duration_seconds": 171.3
  }
],
"5lqea95d": [
  {
    "ended_at": "2021-09-08T13:45:49.803-04:00",
    "started_at": "2021-09-08T13:33:57.803-04:00",
    "activity_ids": [
      199554,
      262339,
      434783,
      461142
    ],
    "duration_seconds": 712.0
  },
],
...
```

```
}
```

Notice how this “user_sessions” object is a dictionary where the keys are the user_ids and the values are the array of user sessions for that user.

Third step

Once you have this new JSON data structure ready, you need to post it to another endpoint. The format of this endpoint is:

Request

```
POST https://api.slangapp.com/challenges/v1/activities/sessions
Content-Type: application/json
Authorization: Basic MTpC__SAMPLE__VjPQ==    ← replace this with your key
Body: {“user_sessions”: {...}}
```

Response

```
204 No content
```

Or in case of error:

```
401 Unauthorized
```

```
400 Bad request
```

```
429 Too many requests
```
(en el caso de ser rate-limited)

Bonus question

Can you provide an accurate runtime complexity analysis for your algorithm?

What we’re expecting

We’re expecting an algorithm that handles these three steps in a nice, cleanly organized and efficient way. It should handle edge cases. You can choose any language you feel most comfortable with to implement it. Make sure you follow conventions and idiomatic guidelines for your chosen language! We move fast but are fanatical about writing clean, idiomatic,

maintainable code. You should, as a comment within your functions, correctly explain the running time complexity of your algorithm. Extra credit if it's better than quadratic.

Submission

In addition to submitting your answer to the endpoint provided in the third step, you should submit your algorithms as a GitHub repository. You should share your repository with me (username **ricardovj**), and you should use the repository as you would for any other project — commit early and often! We're interested in seeing the history of your work, too.

All right — that's it. Thanks for reading through this, and let me know if you have any questions or comments. We're excited to see what you come up with!