On the map

a doma.dev test task.

v 1.1.1

Pola <pola@doma.dev>

Jonn <jonn@doma.dev>

October 2022

1 Introduction

Thank you for your interest in working at doma.dev and agreeing to complete this task! It is designed to take around an hour to complete.

You are given a project which has some functions which are yet not implemented. You have to fill in implementations of these functions following the specification given in this document.

Note that there may be bugs in the skeleton implementation. If you find an inconsistency, consider the specification correct and the implementation faulty.

2 The gist

Marketing department of the startup you work at wants to put your product *on the map*. They tasked you with making a bot which makes bids on advertisement space on a high-traffic website called `zerohr.io`. This website runs an ad auction, which implements *AdNonsense* protocol. Your goal is to place bids over the course of a month[1] and beat your competitors around the world in the amount of click-through events your bot has generated for your startup's website. Good luck!

[1] Of course, we won't run your submission for a month, but rather we'll simulate the passage of time.

3 Treasury, click-through, and sales

3.1 Treasury

The marketing budget your bot will use to make bids is `100,000` credits. AdNonsense provider won't let you spend more than that amount.

3.2 Click-through

Curious users visit your startup's website both naturally and thanks to the ads you place. Only the latter are considered to be click-through events.

3.3 Campaign sales

Just as organic traffic, click-through events may result in sales or in returning users, who will eventually make a purchase. Both sales due to click-through and sales of returning users, who discovered your startup by clicking on an ad you placed are considered "campaign sales".

4 AdNonsense protocol

AdNonsense works over HTTP and returns JSON. There is no API rate limiting in AdNonsense, however, since AdNonsense servers are operating under a constant load, availability may suffer from short outages[2], especially if the load spikes.

Each output sends along an `X-Nonsense-Auth-Chal` header. It is an ASCII string that you have to sign with the secret key you used to register. This signature has to be then inserted into `X-Nonsense-Auth-Sig` encoded as url-safe base64 string.

[2]These outages are also simulated. Hopefully. `;)`

4.1 POST /register

- Input:
  - `pk`: the public key that you'll use for authentication.
  - `url`: the URL of your startup's website[3].
- Output:
  - `status`:
    - HTTP 200: the registration was successful |
    - HTTP 403: the public key is already registered

4.2 GET /items

- Output: [
  - `id`: the id of the ad display slot.
  - `t0`: time when the ad will start showing.
  - `t1`: time when the ad will stop showing.
  - `topBid`: current top bid value.

  ]

4.3 GET /item/:id

- Output:
  - `id`: the id of the ad display slot.
  - `t0`: time when the ad will start showing.
  - `t1`: time when the ad will stop showing.
  - `bids`: [
    - `value`: bid value.
    - `bidder`: public key of the party that made the bid.

    ]

[3]Use `config/config.exs` to set this value, as it's going to be patched by the submission runner. See section 5.2.

4.4 POST /bid/:id

- Input:
  - `id`: the id of the ad display slot you're bidding on.
  - `value`: the value of the bid you're placing.
- Output:
  - `topBid`: current top bid value for this slot.
  - `bidder`: public key of the party that made the bid.

5 Submissions

As mentioned in "the gist" section, the run of your submission will be ran in a sped-up simulated time. Your submission will be competing against a bunch of naive strategies (implemented by us) and against strategies, submitted by other candidates. To qualify for a middle-level position, you're expected to beat all the naive strategies. To qualify for a junior position, you're expected to produce a working submission that scores comparable to an average naive strategy.

5.1 Running the bots

The bots are ran on the same machine over several rounds. During each round, each competitor is assigned one of several possible latencies. Every submission will have a fair chance to run at each relative latency. The latency is simulated using the same facilities as the time, but isn't sped up proportionally. Thus, the latency values are close to the real world ones.

5.2 Technology of the bot runner

The bots are ran in `docker compose`, attached to a single docker network. The API server is registered in this network under name `zerohr_io`. Thus, you can make requests to `http://zerohr_io` and docker will correctly resolve those. The names of the containers that you submit will be replaced with a random, deterministic and unique names in the following files:

- `*.yaml`
- `config/config.exs`