sovrn

Take Home Ad Server Exercise

Construct a basic ad server to meet the requirements of the endpoints described below. Consider the code produced to be written for production purposes. The application may use any frameworks as long as the requirements are met. Assume database tables have been created for you. Endpoints should be on port 80. No additional software may be used at runtime(a different database for example), but feel free to share those ideas in the comments.txt described below. The exercise should be submitted by creating a public repo on your personal github account and sharing that link back to sovrn. Submit your commit history as you would on a normal production project.

A readme file is expected and must contain instructions for how to run the application from scratch, including how to configure the connection to the MySQL server. Further information can be added to the readme as you see fit. Additional communications you wish to make about the problem should be included in a comments.txt file along with your solution.

Overview

The ad server is called from a page visitor's browser. There is javascript on the website that will coordinate calls to the ad server and display the ads on the page. The javascript first calls the /ad endpoint to ask for an ad that is relevant to the current website and visitor. The ad server asks all partners for an ad for this page view. The partners send an ad with a bid price on their response if they want to participate in the auction. The ad server selects the highest bid as the winner and returns it to the page. The javascript inserts the ad html to the page that was returned. If the user clicks the ad, the javascript will send a notification call to the /click endpoint to record that click for that specific ad transaction. This is the end of the 'live' activity that happens in the browser when a user visits the page. The /history endpoint is used to retrieve the history of ads served and which were valid clicks for dashboarding, reporting, user UI, and similar purposes.

Endpoints

/ad

Description: When a user loads a webpage, sovrn's JavaScript will construct this endpoint using information about the page being viewed and the user viewing the page. A request will be made using this endpoint, and your ad server will respond to the request.

Requirements

- Query params required on GET requests:
  - width – width of the ad spot on the page
  - height – height of the ad spot on the page
  - userid – id of the user that owns the website and is requesting an ad
  - url – the complete url of the page that the ad will be placed on
- JSON Response Body

```
{
        tid: “UUID”,
        html: “….some html code block from the winning auction….”
}
```

- The endpoint will serve an ad provided by one of the partner providers
- Partners are selected by looking up the user, finding the appropriate size mapping and looking up the matching providers from the database (see MySQL table creation statements below)
- A partner must have a matching size for this user AND have a direct association to the user through the user_provider_assoc table to qualify for serving an ad on the current transaction
- Requests to the providers for ads are POSTed to the url in the provider MySQL table
- Response times from this endpoint are expected to be less than 200ms. Providers are allowed 150ms of response time after we request an ad from them.
- Example request and response messages for partners are below.

Request:	**Response:**

{
    width: 300,
    height: 250,
    domain: “pageThatTheUserVisited.com”
    userip: “44.33.22.11”
    useragent: “…Browser user agent string…”
}
	{
	    bidprice: 0.00045,
	    adhtml: “…Some code html block…“
	}

/click

Description: This endpoint records when a user clicks on an ad. This could be used to track how successful an ad is. The click information is recorded with the original request information.

Requirements

- Query params required on GET requests:
  - tid – transaction id associated with the original request
  - userid – userid associated with the original ad request
- Clicks should update ad request history (see endpoint below) to represent click activity

/history

Description: This endpoint is somewhat contrived for the purpose of this test. It will list out the previous 100 ad transactions. This could be used by some UI to show users a history of their site traffic for example.

Requirements

- The history only has to maintain the last 100 transactions
- The transactions can be kept in memory if you choose
- Example output of single record

```
{
        transactionId: “UUID”,
        userid: 45,
        bids: [{
                providerId: 12,
                bidPrice: 0.00034,
              },{
                providerId: 14,
                bidPrice: 0.00064
              }],
        winningPrice: 0.00035,
        winningProvider: 12,
        clickResult: “CLICK”
}
```

- clickResult field has three possible values
  - REQUEST – ad was returned to the page but no click call was received
  - CLICK – ad was served to the page and this ad was clicked which called the /click endpoint
  - STALE – ad was clicked but it was clicked more than X seconds after the original ad was served to the page. The number of seconds should be configurable through the command line set system property named “CLICKTIMER”

Questions(to be answered in comments.txt):

- How could a user commit fraud with the current design?
- What might you do to mitigate it?

MySQL Creation Statements

```
CREATE TABLE `user` (
	`user_id` INT NOT NULL,
	`username` varchar(100) NOT NULL UNIQUE,
	PRIMARY KEY (`user_id`)
);

CREATE TABLE `user_size_assoc` (
	`user_id` INT NOT NULL,
	`user_size_assoc_id` INT NOT NULL,
	`ad_size_id` INT NOT NULL,
	PRIMARY KEY (`user_size_assoc_id`)
);

CREATE TABLE `ad_size` (
	`ad_size_id` INT NOT NULL,
	`width` INT NOT NULL,
	`height` INT NOT NULL,
	PRIMARY KEY (`ad_size_id`)
);

CREATE TABLE `provider_size_assoc` (
	`provider_size_assoc_id` INT NOT NULL,
	`ad_size_id` INT NOT NULL,
	`provider_id` INT NOT NULL,
	PRIMARY KEY (`provider_size_assoc_id`)
);

CREATE TABLE `provider` (
	`provider_id` INT NOT NULL,
	`provider_name` varchar(64) NOT NULL UNIQUE,
	`url` varchar(512) NOT NULL AUTO_INCREMENT UNIQUE,
	PRIMARY KEY (`provider_id`)
);

CREATE TABLE `user_provider_assoc` (
	`user_provider_assoc_id` INT NOT NULL,
	`user_id` INT NOT NULL,
	`provider_id` INT NOT NULL,
	PRIMARY KEY (`user_provider_assoc_id`)
);

ALTER TABLE `user_size_assoc` ADD CONSTRAINT `user_size_assoc_fk0` FOREIGN KEY (`user_id`)
REFERENCES `user`(`user_id`);

ALTER TABLE `user_size_assoc` ADD CONSTRAINT `user_size_assoc_fk1` FOREIGN KEY
(`ad_size_id`) REFERENCES `ad_size`(`ad_size_id`);

ALTER TABLE `provider_size_assoc` ADD CONSTRAINT `provider_size_assoc_fk0` FOREIGN KEY
(`ad_size_id`) REFERENCES `ad_size`(`ad_size_id`);

ALTER TABLE `provider_size_assoc` ADD CONSTRAINT `provider_size_assoc_fk1` FOREIGN KEY
(`provider_id`) REFERENCES `provider`(`provider_id`);

ALTER TABLE `user_provider_assoc` ADD CONSTRAINT `user_provider_assoc_fk0` FOREIGN KEY
(`user_id`) REFERENCES `user`(`user_id`);
```