Общее описание

Тестовое задание описывает сервис, отрабатывающий реальный сценарий нашего продукта.

Продукт в общих чертах выглядит так:

- есть личный кабинет, где может зарегистрироваться владелец любого сайта, подтвердить владение некоторым сайтом, настроить рекламные плейсменты, которые должны на нём показываться, и установить необходимый код к себе на сайт
- при загрузке пользователем страницы такого сайта срабатывает наш JS-код, находит на странице размещённые плейсменты, и отправляет на наш бэкенд запросы за рекламой
- бэкенд в свою очередь отправляет запросы за рекламой к нескольким подключенным к нам рекламным партнёрам, каким-то образом выбирает, какую из полученной рекламы показать в плейсменте, и отвечает на исходный запрос этой рекламой

В рамках тестового задания мы попросим написать описанный выше бэкенд.

Описание задачи

Формат запросов, прилетающих от наших плейсментов, такой:

```
POST /placements/request
…
Content-Type: application/json

{
	"id": <string>,
	"tiles": [{
		"id": <uint>,
		"width": <uint>,
		"ratio": <float>
	}, … ],
	"context": {
		"ip": <ip4 string>,
		"user_agent": <string>
	}
}
```

Запрос считается ошибочным, если:

- отсутствует любое из описанных выше полей (WRONG_SCHEMA)
- пустой массив в tiles (EMPTY_TILES)
- пустое значение любого поля (EMPTY_FIELD)

На ошибочный запрос надо отвечать кодом 400 и писать сообщение об ошибке в лог, в котором должен присутствовать соответствующий код

==========

Сервис должен уметь понимать флаги (и требовать их наличие):

-p, порт на котором сервис должен слушать запросы
-d, список строк (через запятую), представляющих собой ip:port рекламных партнёров.
партнёров может быть от 1 до 10

Сервис должен писать логи в STDOUT

Сервис должен ожидать ответа каждого рекламного партнёра не более 200 мс
Суммарное время его ответа (от запроса плейсмента до ответа плейсменту) не должно превышать 250 мс

==========

Рекламные партнёры принимают запросы в следующем формате:
POST /bid_request
…
Content-Type: application/json

```
{
	"id": <string>,
	"imp": [{
		"id": <uint>,
		"minwidth": <uint>,
		"minheight": <uint>
	}, … ],
	"context": {
		"ip": <ip4 string>,
		"user_agent": <string>
	}
}
```

imp формируется из tiles
minwidth == width из запроса от плейсмента
minheight == math.Floor(width * ratio)
id - те же самые

В случае если им нечего предложить на такой запрос, они отвечают ответом с пустым телом и статусом 204 (No Content)
Если есть:
Content-Type: application/json

```
{
	"id": <string>,
	"imp": [{
		"id": <uint>,
		"width": <uint>,
		"height": <uint>,
```

```
                "title": <string>,
                "url": <string>,
                "price": <float>
        }, … ]
}
```

В ответе могут быть не все imp; они могут быть в другом порядке; но id будет тот же самый, что в соответствующем запрошенном imp
Внешний id ответа повторяет внешний id из запроса к рекламному партнёру

==========

Получив все ответы, ваш сервис должен для каждого элемента tiles из запроса плейсмента выбрать среди imp с таким же id тот, у которого максимальная цена. Одинаковых цен с одним id не будет
Если imp с каким-то id не получено, такого id не должно быть в ответе. Порядок imp в ответе должен соответствовать порядку tiles в запросе плейсмента. Формат ответа:
Content-Type: application/json

```
{
        "id": <string>,
        "imp": [{
                "id": <uint>,
                "width": <uint>,
                "height": <uint>,
                "title": <string>,
                "url": <string>
        }, … ]
}
```

Заключение

Для написанного сервиса будет предоставлено тестовое окружение. Количество запусков в этом окружении будет не ограничено. В ответ на запуск будет предоставлен отчёт, в котором будет указано, сколько тестов выполнено успешно, без ответа почему произошёл каждый из фейлов.
Решением задания является github-репозиторий с кодом.