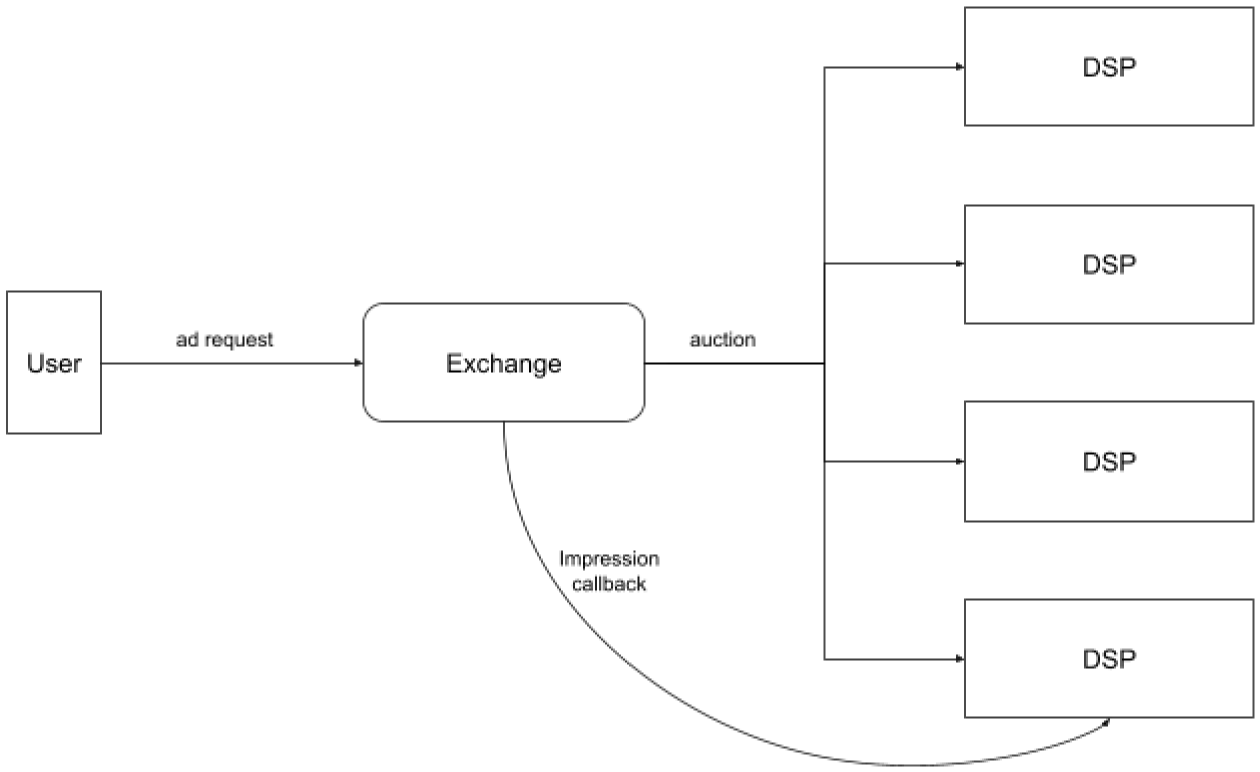
Background

Modern advertising on mobile applications is mostly done using Real Time Bidding (RTB). With this technique, when an application wants to show an ad to a user it sends an HTTP request to a service we call “Exchange” asking for an advertisement. This Exchange then starts an auction and gives multiple Demand Side Platforms, or DSP for short, the chance to buy the impression from the user. These DSPs need to respond, in less than 100ms, with a price that says how much they are willing to pay to get an impression from that user; based on these prices the Exchange will then decide who wins and return the ad of the winner to the application. Exchanges then notify the winner of the auction that they have successfully purchased that impression from the user. Below is a diagram showing the entire flow.



The main task of a DSP is to buy impressions from users. Good DSPs don’t buy every single impression they get since buying them means spending money and not all users are worth it. Imagine a user that has already seen 5 ads from a DSP and never once clicked on it, the probability that he or she will click on the sixth ad is extremely low. A smart DSP would simply stop buying this user after a few impressions, i.e short circuit them. This kind of short circuiting is called **Frequency Capping**, it's a feature that is highly desirable on all competing DSPs.

In addition to capping each user, DSPs have a mechanism that short circuits the spend of the entire system. We call this a **Budget**. This budget is the maximum amount of money that a DSP is willing to spend on a given period of time. DSPs use this budget to keep track of how much they are spending and if they go above it they stop buying impressions until the budget is reset. They also have mechanisms in place that prevent them from spending all the budget on a very short time period. This is to avoid both fraudulent traffic and spikes of requests. It is highly desirable for a DSP to be able to buy traffic the entire day, and not just at 6pm when there is a sudden increase of people playing games.

Problem

You will be implementing a simple DSP that does both the **Frequency Capping** and **Budget limiting** technique we mentioned previously. Your DSP will receive bid requests and respond with a **random bid price if the user has not reached the frequency capping and the budget is not exceeded** or a **204 if the user has reached the frequency capping or if it has reached the budget limit**. The thresholds for this frequency capping and budgets will be:

- At most 5 impressions **per minute,**
- At most 10 impressions **every 3 minutes,**
- A maximum of 10 USD to spend **per day.**

This means that if you get a bid request from a user to whom you've already shown 5 ads in one minute or 10 in three minutes you will respond with a 204, otherwise you respond with a random bid price only if that bid price does not exceed the budget, if it does exceed the budget you respond with a 204 as well.
Users are identified with a unique identifier that you receive in the request's payload.

You will be following a modification of the Open Real Time Bidding specification for your DSP. Your system needs to expose at least two HTTP endpoints, one for bid requests which is defined below and one for impression counts that you should define yourself. Once you've defined it you need to specify it in the `nurl` field of each bid response. This field is what exchange uses to callback when an impression occurs.

Method	**Path**	**Request**	**Response**
POST	/bid	JSON body: ● **id:** auction id, ● **imp** ○ **bidfloor**: minimum price, ● **device** ○ **ip:** user device ip, ● **user** ○ **id:** user unique identifier.	JSON body: ● **id:** auction id, ● **bidid**: unique ID for bid set by the DSP, ● **bid:** ○ **price:** price of bid, ○ **nurl:** impression callback url. Code: ● **200 OK:** if there is a bid

			• **`204 No Content:`** `if there is no bid`

In reality DSPs have very high RPS (requests per second) and very low latency. For example, a small DSP would typically receive ~500k RPS (requests per second) and would respond in less than 5ms. Keep this mind when you are thinking about the overall architecture, deciding which technologies to use and implementing the services.

Deliverables

At a minimum you should deliver an overall explanation of the architecture and the code you wrote to implement. However, the following items would be highly desirable:

- A guide that explains how to setup the system locally,
- An easy way to validate that the system meets the required criteria,
- Benchmarks.

You can use the one you are most comfortable with, however, we recommend you write the code in Go, it's a very easy language to learn and you can start here if you are not familiar with it.