Ad Request Auction System

1. Development

-> Each module is split into packages under the **/pkg** directory.
-> Each module uses a repository and a use case pkg.
-> Models are placed at the parent level to allow generic to all modules.

/repository

-> Persistance layer.
-> Access data storage.

/usecase

-> Application Layer
-> Business logic is applied in this layer.
-> Repository is injected as a dependency from main.go

/models

-> Models used by all the modules are specified here.

/delivery

-> Transportation Layer.
-> Exposes usecase through different communication protocols.
-> uses http pkg to expose rest APIS.

2. Persistance Layer

-> All data are stored in memory

3. Invalidating bidders based on response delay.

-> The value in the BID_DELAY environment value is used as the maximum time allowed for a bidder to respond.
-> The bid delay value is set in the http request's timeout field. So the http request is terminated if the bidder takes more than the bid delay value to respond.

4. Config Variables

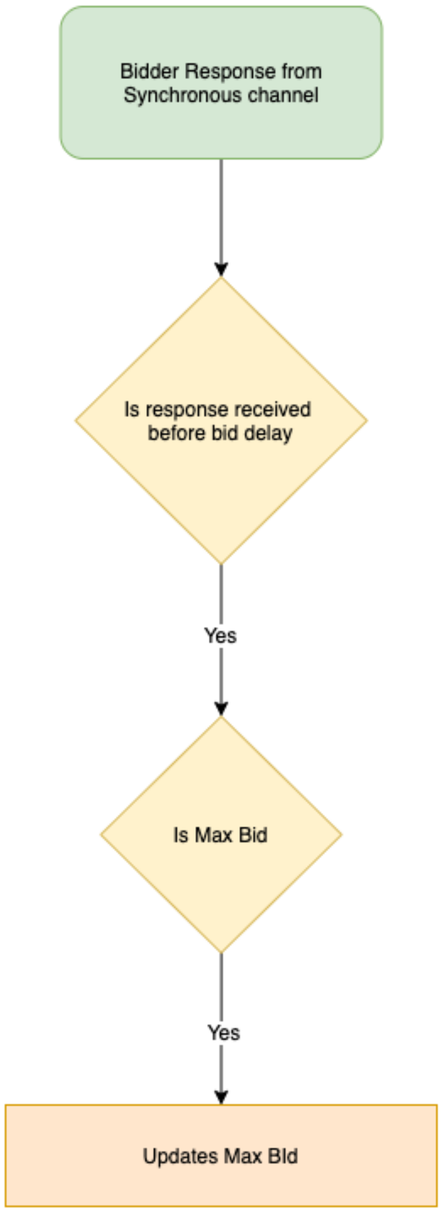
Auction service

-> **BID_DELAY -** Time to wait for the bidder to respond in milliseconds

Bidder Sevice

-> **BID_DELAY -** Time to wait before placing the bid
-> **AUCTION_SERVICE_URL -** The endpoint of the auction service used to register the bidder.
-> **PORT -** Port for the bidder service to bind the http server.

5. Max Bid Calculation


Bidder Response from Synchronous channel
Is response received before bid delay
Yes
Is Max Bid
Yes
Updates Max BId

Max Bid Calculation Flow Diagram

-> If two bids are of the same value and are the maximum bid, the one which was received first is chosen.

6. Simulating multiple bidders

-> Multiple bidders are simulated by running multiple instances of bidder services in different docker containers.

-> The number of services to run is specified in the command below

```
docker-compose up --scale bidderservice=3
```