# Peer to Peer Systems and Blockchains
# Academic Year 2018/2019
# Final Project
# Development of a Dapp for Smart Auctions

## Report

Leonardo Frioli
580611

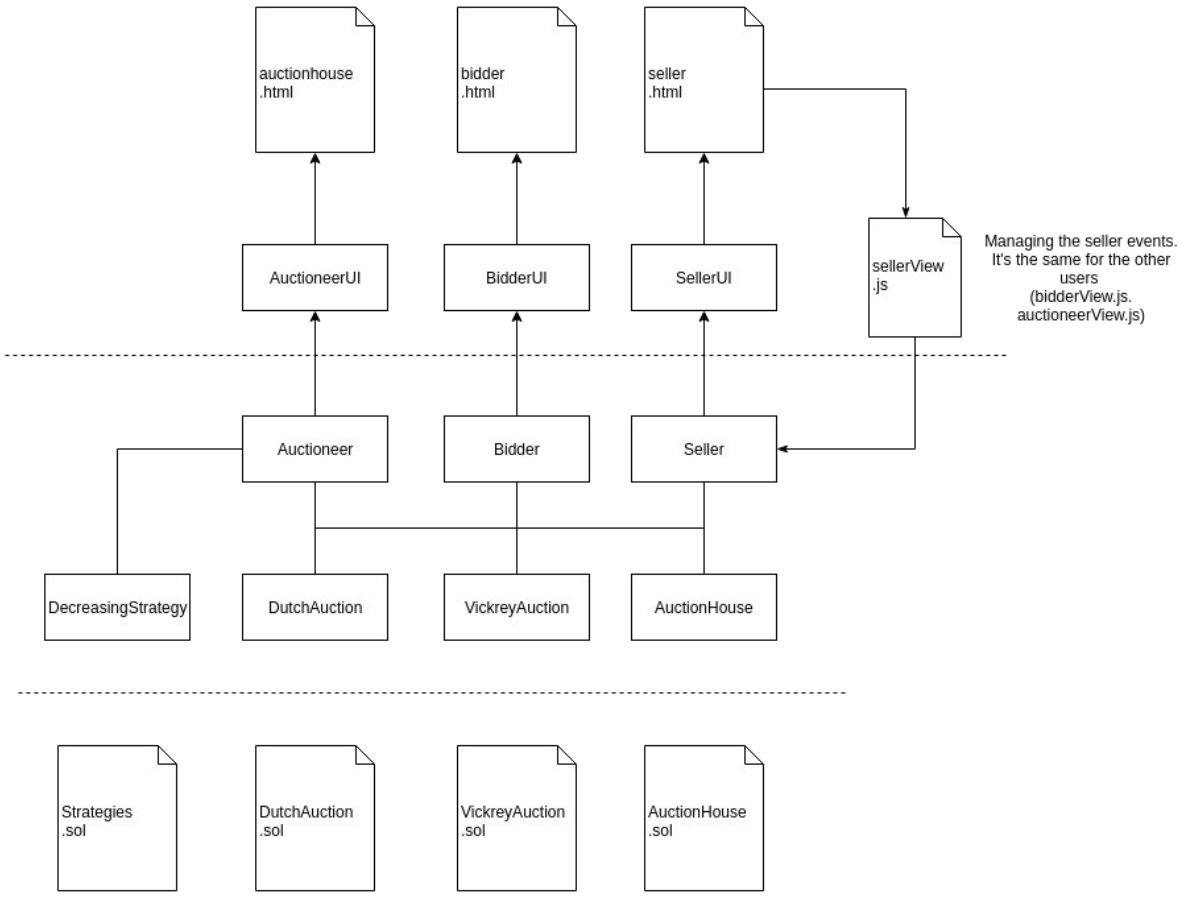
# Index

# Introduction

The aim of this report is to show the architecture of the final term and to explain the most relevant implementation choices done during its development.
We used javascript as main language, but instead of using web3.js we used ethers.js because it's better documented. We have also written a small server script using express.js instead of relying on lite-server because it is too basic. Anyway we will talk about that later in this report.

# Project Structure and Software architecture

The structure of the project and its architecture are strictly related, because each file is representative of a class or of a group of classes sharing the same functionalities. The structure will be presented in the picture below. We started from the Solidity files and created one javascript class class for each contract. This classes implement the interaction with the contracts on the blockchain. Then we have one class for each one of the users (auctioneer, seller, bidder) and other classes related to the users but managing the UI. This classes (BidderUI, SellerUI, AuctioneerUI) are used to implement a set functions that are called to inform the user via the web interface when the events on the blockchain occur.



# Notable Implementation details

## Final-term fixes

During the implemetnation of the Dapp, some fixes on the contracts were performed. The most relevant is the introduction of a contract interface **ISmartAuction.sol** that is implemented by both the Auctions and it's used to group the shared methods and fields.

## AuctionHouse.sol

Since in the requirements was requested to inform the Bidder of the creation of a new Auction, we decided to introduce a contract acting as the AuctionHouse. This contract allows bidders and sellers to subscribe to the AuctionHouse, more or less like in the real life. The contract can be used by the seller to submit an auction and by the bidders to receive an event when an auction is created. This contract is foundamental because, if we want to listen to the SmartAuction events, we first need to connect to it, and therefore we need its address. The address of the deployed Auction is obtained thanks to an event published by the AuctionHouse. The address could be passed using another path, but we preferred to keep the logic as much as possible on the blockchain.

## simpleserver.js

Once the Auctioneer deployed the AuctionHouse, the bidders and the sellers need to know the address of the contract in order to connect to it and to perform the subscription. Before we had a similar problem related to the SmartAuction and we solved it deploying the AuctionHouse. In this case, instead, it's pointless to deploy another contract, so we choose to send the AuctionHouse address to a simple server implemented with node and express.js, and let the bidders/sellers get the address with an ajax request.

## Auction start event

The requirements say that an event should notify the user when the auction starts. It's not very clear if it refers to the moment when the auction is successfully deployed or when the grace period is expired. In the first case there is already an event coming from the AuctionHouse informing the bidders that a new auction is available; in the second case it's not possible to fire an event without calling a function checking if the period is over. Since a kind of polling is needed in this last case, we implemented a function that simply returns how many "grace period" blocks are left and informs the user via UI if the auction is started. If the bidders want to know about the grace period they have to call the function and they will get an immediate response.

# How to test the project locally

1) Run “npm install”
2) Run “truffle compile”
3) Start Ganache
4) Import from Ganache to Metamask three accounts and rename them according to their role
5) Run “npm start”
4) Navigate to http://localhost:3002/
5) Open the pages realted to the three users
6) Go to the AuctionHouse view, select the account in metamask representing the Auctioneer and click on “Use this account”. This allows the frontend to infrom you if you are trying to perform an action with a wrong account. For istance, if you are on the AuctionHouse View, but the metamask selected account is not the Auctioneer, you will be informed by an alert. This process should be done also by the bidder and the seller.
7) Remaining to the AuctionHouse View, click on “Deploy” to deploy the AuctionHouse.
8) Change address to the bidder one, change view and subscribe the bidder to the AuctionHouse.
9) Switch to the seller, subscribe him to the AuctionHouse and click on submit, to inform the AuctionHouse that you want a new Auction to be deployed.
10) Go to the AuctionHouse View (changing the account) and deploy the Auction you prefer.
11) Go to the Bidder View (changing the account) and join the Auction.
12) Now all the three users can interact easily with the deployed Auction, if one of your transaction is reverted you will be informed by the UI.

The interaction is guided by the UI and the parameters used to deploy the auction can be left untouched because they reflect those ones used in the final-term. The default value for the mining rate is 300, that means that the grace period will be of only one block. The grace period “expires immediatly” because the Auctioneer, after the deployment of the Auction, calls a function on the AuctionHouse emitting an Event to inform the other users about the new auction. This call is mined in the “grace period” block. If you want to try better the grace period you could set the mining rate to 150.