1 UI.js

```
class UserInterface{
        // AuctionHouse events
        newSellerSubscribed(sellerAddress){
                throw "This function needs to be implemented in the subclasses";
        }

        newBidderSubscribed(bidderAddress){
                throw "This function needs to be implemented in the subclasses";
        }

        newAuctionSubmitted(sellerAddress, objectDescription){
                throw "This function needs to be implemented in the subclasses";
        }

        auctionDeployedSuccessfully(auctionAddress, auctionName, objectDesciption){
                throw "This function needs to be implemented in the subclasses";
        }

        // both Dutch and Vickrey events
        notifyWinner(winnerAddress, bid){
                throw "This function needs to be implemented in the subclasses";
        }

        newBlock(blockNumber){
                throw "This function needs to be implemented in the subclasses";
        }

        escrowAccepted(address){
                throw "This function needs to be implemented in the subclasses";
        }

        escrowRefused(address){
                throw "This function needs to be implemented in the subclasses";
        }

        escrowClosed(){
                throw "This function needs to be implemented in the subclasses";
        }

        // Dutch event
        notifyNotEnoughMoney(bidderAddress, bidSent, actualPrice){
                throw "This function needs to be implemented in the subclasses";
        }

        // Vickrey events

        notifyCommittedEnvelop(bidderAddress){
                throw "This function needs to be implemented in the subclasses";
        }

        notifyWithdraw(bidderAddress){
                throw "This function needs to be implemented in the subclasses";
        }

        notifyOpen(bidderAddress, value){
                throw "This function needs to be implemented in the subclasses";
        }

        notifyFirstBid(bidderAddress,value){
                throw "This function needs to be implemented in the subclasses";
        }

        notifySecondBid(bidderAddress,value){
                throw "This function needs to be implemented in the subclasses";
        }
}
```

2 Auction.js

```
1
2   class Auction {
3        constructor(type){
4                this.type = type // DutchAuction or VickreyAuction
5                this.contract = null;
6                this.contractAddress = null;
7                this.objectDescription = null;
8        }
9
10       async getContractFactory(signer){
11               // getting the json
12               let auctionJSON = await $.getJSON( this.type + ".json");
13
14               // Create an instance of a Contract Factory
15               return  new ethers.ContractFactory( auctionJSON.abi, auctionJSON.bytecode, signer);
16
17       }
18
19       async connect(signer, address){
20               let c = await $.getJSON( this.type + ".json")
21               this.contract = new ethers.Contract(address, c.abi, signer);
22               console.log("connected");
23       }
24
25       deploy(){
26               throw "This method needs to be redefined in the subclasses";
27       }
28
29       async destroy(){
30               await this.contract.destroyContract();
31       }
32
33       registerToEvents(ui){
34               this.contract.on("Winner", (winnerAddress, bid) => {
35                       ui.notifyWinner(winnerAddress, bid);
36               });
37
38               this.contract.on("NewBlock", (blockNumber) => {
39                       ui.newBlock(blockNumber);
40               });
41
42               this.contract.on("EscrowAccepted", (address) => {
43                       ui.escrowAccepted(address);
44               });
45
46               this.contract.on("EscrowRefused", (address) => {
47                       ui.escrowRefused(address);
48               });
49
50               this.contract.on("EscrowClosed", () => {
51                       ui.escrowClosed();
52               });
53       }
54
55       async acceptEscrow(){
56               await this.contract.acceptEscrow();
57       }
58
59       async refuseEscrow(){
60               await this.contract.refuseEscrow();
61       }
62
63       async concludeEscrow(){
64               await this.contract.concludeEscrow();
65       }
66
67       getSeller(){
68               return this.contract.getSeller();
69       }
```

```javascript
70
71          getReservePrice(){
72                  return this.contract.getReservePrice();
73          }
74
75          async getGracePeriod(){
76                  return await this.contract.getGracePeriod();
77          }
78
79          addBlock(){
80                  this.contract.addBlock();
81          }
82  }
83
84  class DecreasingStrategy{
85          constructor(type){
86                  this.type = type; // Linear, Logarithmic, InverseLogarithmic
87                  this.strategy = null;
88          }
89
90          async deploy(signer){
91                  // deploying the choosen strategy
92                  let decreasingStrategyJSON = await $.getJSON( this.type + "DecreasingStrategy.json");
93
94                  let decreasingStrategyFactory = new ethers.ContractFactory( decreasingStrategyJSON.abi,
                        decreasingStrategyJSON.bytecode, signer );
95
96                  this.strategy = await decreasingStrategyFactory.deploy();
97
98                  await this.strategy.deployed();
99          }
100
101         async destroy(){
102                 await this.strategy.destroyContract();
103         }
104
105 }
106
107 class DutchAuction extends Auction{
108         constructor(){
109                 super("DutchAuction");
110         }
111
112         async deploy(signer,_reservePrice, _initialPrice, _openedForLength, _seller,
                    decreasingStrategyAddress, miningRate){
113                 // deploying the DutchAuction
114                 let factory = await this.getContractFactory(signer);
115
116                 _reservePrice = ethers.utils.parseEther(_reservePrice);
117                 _initialPrice = ethers.utils.parseEther(_initialPrice);
118                 this.contract = await factory.deploy(_reservePrice, _initialPrice, _openedForLength,
                        _seller,  decreasingStrategyAddress, miningRate);
119
120                 console.log(this.contract.address);
121                 this.contractAddress = this.contract.address;
122
123                 // The contract is NOT deployed yet; we must wait until it is mined
124                 await this.contract.deployed()
125
126         }
127
128         registerToEvents(ui){
129                 super.registerToEvents(ui);
130
131                 // this is specific to DutchAuction
132                 this.contract.on("NotEnoughMoney", (bidderAddress, bidSent, actualPrice) => {
133                         ui.notifyNotEnoughMoney(bidderAddress, bidSent, actualPrice);
134                 });
135         }
136
137
```

```
138             // bidding a value to the Auction
139             async bid(bidValue) {
140                     let overrides = {
141                     gasLimit: 6000000,
142                     value: ethers.utils.parseEther(bidValue)
143                     };
144
145                     await this.contract.bid(overrides);
146
147             }
148
149             getInitialPrice(){
150                     return this.contract.getInitialPrice();
151             }
152
153             async getCurrentPrice(){
154                     return await this.contract.getCurrentPrice();
155             }
156
157             async getOpenedFor(){
158                     return await this.contract.getOpenedFor();
159             }
160     }
161
162     class VickreyAuction extends Auction{
163             constructor(){
164                     super("VickreyAuction");
165             }
166
167             async deploy(signer, _reservePrice, _commitmentPhaseLength, _withdrawalPhaseLength,
                   _openingPhaseLenght, _depositRequired, _seller, miningRate){
168                     // deploying the DutchAuction
169                     let factory = await this.getContractFactory(signer);
170
171                     _reservePrice = ethers.utils.parseEther(_reservePrice);
172                     _depositRequired = ethers.utils.parseEther(_depositRequired);
173                     this.contract = await factory.deploy(_reservePrice, _commitmentPhaseLength,
                        _withdrawalPhaseLength, _openingPhaseLenght, _depositRequired, _seller, miningRate);
174
175                     console.log(this.contract.address);
176                     this.contractAddress = this.contract.address;
177
178                     // The contract is NOT deployed yet; we must wait until it is mined
179                     await this.contract.deployed()
180             }
181
182             registerToEvents(ui){
183                     super.registerToEvents(ui);
184
185                     this.contract.on("CommittedEnvelop", (bidderAddress) => {
186                             ui.notifyCommittedEnvelop(bidderAddress);
187                     });
188
189                     this.contract.on("Withdraw", (bidderAddress) => {
190                             ui.notifyWithdraw(bidderAddress);
191                     });
192                     this.contract.on("Open", (bidderAddress, value) => {
193                             ui.notifyOpen(bidderAddress,value);
194                     });
195
196                     this.contract.on("FirstBid", (bidderAddress, value) => {
197                             ui.notifyFirstBid(bidderAddress,value);
198                     });
199
200                     this.contract.on("SecondBid", (bidderAddress, value) => {
201                             ui.notifySecondBid(bidderAddress,value);
202                     });
203             }
204
205             async commitBid(bid, nonce, depositRequired){
206                     let overrides = {
```

4

```
207            gasLimit: 50000,
208            value: ethers.utils.parseEther(depositRequired)
209            };
210            // we use the utility provided by the contract so that we are sure that the parameters
                   are passes correctly to the function
211            let envelop = await this.contract.doKeccak( ethers.utils.parseEther(bid),nonce);
212            await this.contract.commitBid(envelop,overrides);
213        }
214
215    async withdraw(){
216            await this.contract.withdraw();
217        }
218
219    async open(nonce, bid){
220            let overrides = {
221            gasLimit: 60000,
222            value: ethers.utils.parseEther(bid)
223            };
224            await this.contract.open(nonce,overrides);
225        }
226
227    async finalize(){
228            let overrides = {
229            gasLimit: 6000000
230            };
231            await this.contract.finalize(overrides);
232        }
233
234    async getDepositRequired(){
235            return await this.contract.getDepositRequired();
236        }
237
238    async getCommitmentPhaseLength(){
239            return await this.contract.getCommitmentPhaseLength();
240        }
241
242    async getWithdrawalPhaseLength(){
243            return await this.contract.getWithdrawalPhaseLength();
244        }
245
246    async getOpeningPhaseLength(){
247            return await this.contract.getOpeningPhaseLength();
248        }
249 }
250
251 class AuctionHouse{
252    constructor(){
253            this.contract = null;
254        }
255
256    async deploy(signer){
257            let auctionHouseJSON = await $.getJSON("AuctionHouse.json");
258            let auctionHouseFactory = new ethers.ContractFactory(auctionHouseJSON.abi,
                   auctionHouseJSON.bytecode, signer);
259
260            this.contract = await auctionHouseFactory.deploy();
261            await this.contract.deployed();
262
263            console.log(this.contract.address);
264
265            return this.contract.address;
266        }
267
268    async connect(signer, auctionHouseAddress){
269            let c = await $.getJSON("AuctionHouse.json");
270
271            this.contract = new ethers.Contract(auctionHouseAddress, c.abi, signer);
272            console.log("connected to auction house");
273        }
274
275    async destroy(){
```

```
276                    await this.contract.destroyContract();
277            }
278
279        registerToEvents(ui){
280                    this.contract.on("NewSellerSubscribed", (sellerAddress) => {
281                    ui.newSellerSubscribed(sellerAddress);
282                    });
283
284                    this.contract.on("NewBidderSubscribed", (bidderAddress) => {
285                    ui.newBidderSubscribed(bidderAddress);
286                    });
287
288                    this.contract.on("AuctionSubmitted", (sellerAddress, objectDescription) => {
289                    ui.newAuctionSubmitted(sellerAddress, objectDescription);
290                    });
291
292                    this.contract.on("NewAuction", (auctionAddress, auctionName, objectDesciption) => {
293                    ui.auctionDeployedSuccessfully(auctionAddress, auctionName, objectDesciption);
294                    });
295            }
296
297        async subscribeAsBidder(){
298                    await this.contract.subscribeAsBidder();
299            }
300
301        async subscribeAsSeller(){
302                    await this.contract.subscribeAsSeller();
303            }
304
305        submitAuction(objectDescription){
306                    this.contract.submitAuction(objectDescription);
307            }
308
309        notifyNewAuction(auctionAddress, auctionType, objectdescr){
310                    this.contract.notifyNewAuction(auctionAddress, auctionType, objectdescr);
311            }
312
313    }
```

3 appUI.js

```
1
2    // @return the defined user, it can be the auctionhouse or the bidder or the seller
3    function getUser() {
4            if (typeof auctioneer !== 'undefined') return auctioneer;
5            else if (typeof bidder !== 'undefined') return bidder;
6            else if (typeof seller !== 'undefined') return seller;
7            else {
8                    throw "No user is defined";
9            }
10   }
11
12
13   // showing/hiding a spinner next to the passed element
14   function showSpinnerNextTo(element) {
15           $(element).next().show();
16   }
17   function hideSpinnerNextTo(element) {
18           $(element).next().hide();
19   }
20
21
22   // seller and bidder will periodically check if the AuctionHouse has been deployed
23   let addressDisplayed = false;
24   function checkForAuctionHouseAddress() {
25           $.ajax({
26           type: "GET",
27           url: "auctionhouse/address",
28           dataType: 'json',
29           success: function (data) {
```

```
30
31          if (data.contractAddress != "") {
32                  console.log(data.contractAddress);
33                  $("#auctionHouseAddress").text(data.contractAddress);
34                  hideSpinnerNextTo("#subscribeToAuctionHouse");
35                  $("#subscribeToAuctionHouse").show();
36                  addressDisplayed = true;
37          }
38
39          },
40          complete: function () {
41                  if (addressDisplayed == false)
42                  setTimeout(checkForAuctionHouseAddress, 1000);
43          }
44          });
45  }
46
47  function changeViewBasedOn(auctionType){
48
49          if (auctionType == "VickreyAuction"){
50                  $(".vickrey").show(); // or attr diplay block
51                  $(".dutch").hide();
52          } else {
53                  $(".vickrey").hide();
54                  $(".dutch").show();
55          }
56  }
57
58  const alertHtml = function(text,type){
59  return '<div class="alert alert-' + type + ' alert-dismissible fade show" role="alert"> <span> ' + text
        +
60                  '</span> <button type="button" class="close" data-dismiss="alert" aria-label="Close"> <
                    span aria-hidden="true">&times;</span> ' +
61                  '</button> </div>';
62  }
63
64  function addAlertElement(text, type){
65          $("#alertListDiv").append(alertHtml(text,type));
66  }
67
68  function toggleNotificationModal(text){
69          $("#notificationModalInfo").text(text);
70          $("#notificationModal").modal("toggle");
71  }
72
73
74  // showing an alert if transaction fails
75  function notifyTransactionError(err) {
76          toggleNotificationModal("Something went wrong! " + err);
77  }
78
79
80
81  // accepting the escrow, alert is showed if the transaction is reverted
82  var escrowAccepted = false;
83  $("#acceptEscrow").click(async function () {
84          let user = getUser();
85
86          if (user != null && user.auctionContract != null) {
87                  try {
88                          await user.auctionContract.acceptEscrow();
89                          escrowAccepted = true;
90                  } catch (err) {
91                          notifyTransactionError("transaction reverted");
92                  }
93          }
94  });
95
96  // refusing the escrow, alert is showed if the transaction is reverted
97  $("#refuseEscrow").click(async function () {
98          let user = getUser();
```

```
 99
100         if (user != null && user.auctionContract != null) {
101                 try {
102                         await user.auctionContract.refuseEscrow();
103                 } catch (err) {
104                         notifyTransactionError("transaction reverted");
105                 }
106         }
107  });
108
109  // concluding the escrow, alert is showed if the transaction is reverted
110  $("#concludeEscrow").click(async function () {
111         let user = getUser();
112
113         if (user != null && user.auctionContract != null) {
114                 try {
115                         await user.auctionContract.concludeEscrow();
116                 } catch (err) {
117                         notifyTransactionError("transaction reverted");
118                 }
119         }
120  });
121
122  // calling getSeller on the deployed contract
123  $("#getSeller").click(function () {
124         let user = getUser();
125
126         if (user != null && user.auctionContract != null) {
127                 user.auctionContract.getSeller().then((seller) => {
128                         $("#getSellerResult").text(seller);
129                 });
130
131         }
132  });
133
134  // calling getReservePrice on the deployed contract
135  $("#getReservePrice").click(function () {
136         let user = getUser();
137
138         if (user != null && user.auctionContract != null) {
139                 user.auctionContract.getReservePrice().then((price) => {
140                         $("#getReservePriceResult").text(price.toString());
141                 });
142         }
143  });
144
145  // calling getInitialPrice on the deployed contract
146  $("#getInitialPrice").click(function () {
147         let user = getUser();
148
149         if (user != null && user.auctionContract != null) {
150                 user.auctionContract.getInitialPrice().then((price) => {
151                         $("#getInitialPriceResult").text(price.toString());
152                 });
153         }
154  });
155
156  // calling getCurrentPrice on the deployed contract
157  // alert is showed if the transaction is reverted
158  $("#getCurrentPrice").click(async function () {
159         let user = getUser();
160
161         if (user != null && user.auctionContract != null) {
162                 try {
163                         let price = await user.auctionContract.getCurrentPrice()
164                         $("#getCurrentPriceResult").text(price.toString());
165                 } catch (err) {
166                         // console.log(err);
167                         notifyTransactionError("transaction reverted");
168                 }
169         }
```

```
170    });
171
172    // alling the getGracePeiod on the deployed contract
173    // this function should be used to see if the auction is opened
174    $("#getGracePeriod").click(async function () {
175            let user = getUser();
176
177            if (user != null && user.auctionContract != null) {
178                    let gracep = await user.auctionContract.getGracePeriod();
179                    if (gracep <= 0) {
180                            $("#getGracePeriodSuccess").show();
181                            $("#getGracePeriodResult").hide();
182                    } else {
183                            $("#getGracePeriodResult").text(gracep.toString());
184                    }
185
186            }
187
188    });
189
190    // calling getOpenedFor on the deployed contract
191    // alert is showed if the transaction is reverted
192    $("#getOpenedFor").click(async function () {
193            let user = getUser();
194
195            if (user != null && user.auctionContract != null) {
196                    try {
197                            let openedfor = await user.auctionContract.getOpenedFor();
198                            $("#getOpenedForDanger").hide();
199                            $("#getOpenedForResult").text(openedfor.toString());
200                    } catch (err) {
201                            // console.log(err);
202                            $("#getOpenedForDanger").show();
203                            $("#getOpenedForResult").hide();
204                    }
205            }
206    });
207
208
209    // calling getDepositRequired on the deployed contract
210    // alert is showed if the transaction is reverted
211    $("#getDepositRequired").click(async function () {
212            let user = getUser();
213
214            if (user != null && user.auctionContract != null) {
215
216                    let deposit = await user.auctionContract.getDepositRequired();
217                    $("#getDepositRequiredResult").text(deposit.toString());
218            }
219    });
220
221
222    // calling getCommitmentPhaseLength on the deployed contract
223    // alert is showed if the transaction is reverted
224    $("#getCommitmentPhaseLength").click(async function () {
225            let user = getUser();
226
227            if (user != null && user.auctionContract != null) {
228                    try {
229                            let phasel = await user.auctionContract.getCommitmentPhaseLength();
230                            $("#getCommitmentPhaseLengthResult").text(phasel.toString());
231                            $("#getCommitmentPhaseLengthResult").show();
232                            $("#getCommitmentPhaseLengthDanger").hide();
233                    } catch (err) {
234                            // notifyTransactionError("transaction reverted");
235                            $("#getCommitmentPhaseLengthResult").hide();
236                            $("#getCommitmentPhaseLengthDanger").show();
237                    }
238
239            }
240    });
```

```
241
242     // calling getWithdrawalPhaseLength on the deployed contract
243     // alert is showed if the transaction is reverted
244     $("#getWithdrawalPhaseLength").click(async function () {
245             let user = getUser();
246
247             if (user != null && user.auctionContract != null) {
248                     try {
249                             let phasel = await user.auctionContract.getWithdrawalPhaseLength();
250                             console.log("withdrawal length" + phasel);
251                             $("#getWithdrawalPhaseLengthResult").text(phasel.toString());
252                             $("#getWithdrawalPhaseLengthResult").show();
253                             $("#getWithdrawalPhaseLengthDanger").hide();
254
255                     } catch (err) {
256                             // notifyTransactionError("transaction reverted");
257                             $("#getWithdrawalPhaseLengthResult").hide();
258                             $("#getWithdrawalPhaseLengthDanger").show();
259                     }
260             }
261     });
262
263     // calling getOpeningPhaseLength on the deployed contract
264     // alert is showed if the transaction is reverted
265     $("#getOpeningPhaseLength").click(async function () {
266             let user = getUser();
267
268             if (user != null && user.auctionContract != null) {
269                     try {
270                             let phasel = await user.auctionContract.getOpeningPhaseLength();
271                             $("#getOpeningPhaseLengthResult").text(phasel.toString());
272                             $("#getOpeningPhaseLengthResult").show();
273                             $("#getOpeningPhaseLengthDanger").hide();
274
275                     } catch (err) {
276                             // notifyTransactionError("transaction reverted");
277                             $("#getOpeningPhaseLengthResult").hide();
278                             $("#getOpeningPhaseLengthDanger").show();
279                     }
280
281             }
282     });
283
284
285     // calling add block
286     // the block added will be notified by an event on the blockchain
287     $("#addBlock").click(function () {
288             let user = getUser();
289
290             if (user !== null && user.auctionContract !== null) {
291                     user.auctionContract.addBlock();
292             } else {
293                     console.log("auction not created yet");
294             }
295     });
296
297     // select the metamask's account you will use in that web page
298     // if you are cliking a button with a different account, an alert message will notify you
299     $("#metamaskAccountUsedBtn").click(function () {
300             $("#metamaskAccountUsedBtn").hide();
301             // from now on the displayed address won't change
302             $("#currentMetamaskAccount").attr("id", ethereum.selectedAddress);
303
304             let user = getUser();
305             user.pubKey = ethereum.selectedAddress.toLowerCase();
306     });
307
308     // showing tha alert message to notify the user to be consistent using the accounts
309     $(".btn")
310             .not("#metamaskAccountUsedBtn")
311             .not("#auctionHouseContractAddressCopyBtn")
```

```
312        .not("#notificationModalDismissBtn")
313        .mouseover(function () {
314        let user = getUser();
315
316        if (user.pubKey != null && user.pubKey != ethereum.selectedAddress.toLowerCase()) {
317            toggleNotificationModal("Before you must change the address to that one you selected at
                   the beginning!");
318        }
319  });
320
321
322
323  //called when window loads
324  $(window).on('load', function () {
325        $('[data-toggle="tooltip"]').tooltip(); //enablig tooltips
326
327        $("#currentMetamaskAccount").text(ethereum.selectedAddress);
328
329        // updating the displayed account every time it changes
330        // if the account is selected, only that one willl be displayed
331        ethereum.on('accountsChanged', function (accounts) {
332            $("#currentMetamaskAccount").text(ethereum.selectedAddress);
333        })
334
335
336        if(ethereum.networkVersion  == 3) // ropsten
337            $("#addBlockListElem").hide();
338
339  });
```

4 app.js

```
1
2   App = {
3
4         provider : null,
5         url: 'http://localhost:8545',
6
7         initProvider : function() {
8
9             // connecting to the provider
10            if(typeof web3 != 'undefined') { // Check whether exists a provider, e.g Metamask
11                // connecting to Metamask
12                App.provider = new ethers.providers.Web3Provider(web3.currentProvider);
13                try {
14                // Permission popup
15                    ethereum.enable().then(async() => { console.log("DApp connected " );});
16                }
17                catch(error) { console.log(error); }
18            } else { // Otherwise, create a new local instance of Web3
19                let currentProvider = new web3.providers.HttpProvider(App.url);
20                App.provider = new ethers.providers.Web3Provider(currentProvider);
21            }
22        },
23   }
24
25
26   class User{
27         constructor(){
28             this.pubKey = null;
29             this.auctionContract = null;
30             this.auctionHouseContract = null;
31         }
32
33   }
34
35
36   // called when the window loads
37   $(window).on('load', function () {
38         App.initProvider();
```

11

```
39   });
```

5   sellerView.js

```
 1
 2   $("#metamaskAccountUsedBtn").click(function () {
 3          // showing the card to subscribe to the AuctionHouse
 4          $("#subscribeToAuctionHouseCard").show();
 5
 6          showSpinnerNextTo("#subscribeToAuctionHouse");
 7
 8          setTimeout(checkForAuctionHouseAddress, 1000);
 9   });
10
11   $("#subscribeToAuctionHouse").click(function () {
12          // subscribing to the AuctionHouse
13          $("#subscribeToAuctionHouse").hide();
14          showSpinnerNextTo("#subscribeToAuctionHouse");
15
16          let address = $("#auctionHouseAddress").text();
17          seller.subscribeToAuctionHouse(address);
18   });
19
20
21   $("#submitAuction").click(function () {
22          // submitting a new auction
23          $("#submitAuction").hide();
24          showSpinnerNextTo("#submitAuction");
25
26          let objectDescription = $("#objectDescription").val();
27          seller.submitAuction(objectDescription);
28   });
```

6   SellerUI.js

```
 1
 2   var sellerUI = null
 3
 4   class SellerUI extends UserInterface {
 5          // AuctionHouse events
 6          newSellerSubscribed(sellerAddress) {
 7                 if (sellerAddress.toLowerCase() == seller.pubKey) {
 8                        // notify a successfull subscription to the AuctionHouse
 9                        hideSpinnerNextTo("#subscribeToAuctionHouse");
10                        $("#subscribeToAuctionHouseSuccess").show();
11                        $("#submitCard").show();
12                 }
13          }
14
15          newBidderSubscribed(bidderAddress) { return; } // does nothing
16
17          newAuctionSubmitted(sellerAddress, objectDescription) {
18                 if (sellerAddress.toLowerCase() == seller.pubKey) {
19                        // notify that the auction was successfully submitted
20                        hideSpinnerNextTo("#submitAuction");
21                        $("#submitAuctionSuccess").show();
22                 }
23          }
24
25          auctionDeployedSuccessfully(auctionAddress, auctionType, objectDesciption) {
26                 seller.notifyNewAuction(auctionAddress, auctionType, objectDesciption);
27
28                 changeViewBasedOn(auctionType);
29                 // showing the list of functions to interact with the contract
30                 $("#contractFunctionsCard").show();
31          }
32
33          // both Dutch and Vickrey events
34          notifyWinner(winnerAddress, bid) {
```

```
35              console.log(winnerAddress + " won bidding " + bid);
36
37              addAlertElement("<strong>" + winnerAddress + "</strong>  won!","success");
38          }
39
40          // displaying the just added block number
41          newBlock(blockNumber) {
42              console.log("Block added " + blockNumber);
43              $("#addBlockResult").text(blockNumber);
44          }
45
46          // notify that the escrow has been accepted
47          escrowAccepted(address) {
48              if(address.toLowerCase() == seller.pubKey){
49              $("#acceptEscrowResultSuccess").show();
50              $("#refuseEscrowListElem").hide();
51              }
52          }
53
54          // notify that the escrow has been refused
55          escrowRefused(address) {
56              if(address.toLowerCase() == seller.pubKey){
57              $("#refuseEscrowResultSuccess").show();
58              $("#acceptEscrowListElem").hide();
59              }
60          }
61
62          // notify that the escrow has been concluded
63          escrowClosed() {
64              $("#concludeEscrowResultSuccess").show();
65
66              addAlertElement("Escrow Closed successfully","success");
67          }
68
69          // Dutch event
70          notifyNotEnoughMoney(bidderAddress, bidSent, actualPrice) { return; } // does nothing
71
72          // Vickrey events
73
74          notifyCommittedEnvelop(bidderAddress) {
75              addAlertElement("Envelop committed by <strong>" + bidderAddress + "</strong>","secondary
                      ");
76              console.log("Envelop commited " + bidderAddress);
77          }
78
79          notifyWithdraw(bidderAddress) {
80              console.log("Withdrawal " + bidderAddress);
81              addAlertElement("Withdrawal by by <strong>" + bidderAddress + "</strong>","secondary");
82          }
83
84          notifyOpen(bidderAddress, value) {
85              console.log("Open " + bidderAddress + " bid " + value);
86              addAlertElement("<strong>" + bidderAddress + "</strong> opened the envelop","secondary")
                      ;
87          }
88
89          notifyFirstBid(bidderAddress, value) {
90              console.log("First bid " + bidderAddress + " bid " + value);
91              addAlertElement("First bid <strong>" + bidderAddress + "</strong> bid <strong>" + value
                      + "</strong>" ,"secondary");
92          }
93
94          notifySecondBid(bidderAddress, value) {
95              console.log("second bid " + bidderAddress + " bid " + value);
96              addAlertElement("Second bid <strong>" + bidderAddress + "</strong> bid <strong>" + value
                      + "</strong>" ,"secondary");
97          }
98      }
99
100
101
```

```
102    // called when the window loads
103    $(window).on('load', function () {
104            sellerUI = new SellerUI();
105    });
```

7   Seller.js

```
1
2    // variable storing the sller (user) informations
3    let seller = null;
4
5    class Seller extends User{
6            constructor(){
7                    super();
8            }
9
10           // connecting to the AuctionHouse
11           async subscribeToAuctionHouse(auctionHouseAddress){
12                   this.auctionHouseContract = new AuctionHouse();
13                   await this.auctionHouseContract.connect(App.provider.getSigner(),auctionHouseAddress );
14
15                   this.auctionHouseContract.registerToEvents(sellerUI);
16
17                   try{
18                           await this.auctionHouseContract.subscribeAsSeller();
19                   }catch(err){
20                           // already subscribed
21                           notifyTransactionError("Probably that address is already subscribed");
22                           sellerUI.newSellerSubscribed(this.pubKey);
23                   }
24           }
25
26
27           notifyNewAuction(auctionAddress, auctionType, objectDesciption){
28                   if(auctionType == "VickreyAuction"){
29                   this.auctionContract = new VickreyAuction();
30                   } else {
31                   this.auctionContract = new DutchAuction();
32                   }
33
34                   this.auctionContract.objectDescription = objectDesciption;
35                   this.auctionContract.contractAddress = auctionAddress;
36                   this.connectToContract();
37           }
38
39           // submitting a new auction giving a description of the selled object
40           submitAuction(objectDescription){
41                   this.auctionHouseContract.submitAuction(objectDescription);
42           }
43
44           // connecting to the deployed contract
45           async connectToContract(){
46
47                   await this.auctionContract.connect( App.provider.getSigner(), this.auctionContract.
                          contractAddress);
48                   this.auctionContract.registerToEvents(sellerUI);
49           }
50
51    }
52
53    // called when the window loads
54    $(window).on('load', function () {
55            seller = new Seller();
56    });
```

8   bidderView.js

```
1
2    $("#metamaskAccountUsedBtn").click(function () {
```

```
3          // showing the card to subscribe to the AuctionHouse
4          $("#subscribeToAuctionHouseCard").show();
5          showSpinnerNextTo("#subscribeToAuctionHouse");
6
7          setTimeout(checkForAuctionHouseAddress, 1000);
8    });
9
10   $("#subscribeToAuctionHouse").click(function () {
11          // subscribing to the AuctionHouse
12          $("#subscribeToAuctionHouse").hide();
13          showSpinnerNextTo("#subscribeToAuctionHouse");
14
15          let address = $("#auctionHouseAddress").text();
16          bidder.subscribeToAuctionHouse(address);
17
18   });
19
20
21   $("#bidButton").click(async function () {
22          // bidding
23
24          $("#bidButton").hide();
25          let bidValue = $("#bidValue").val();
26          if (bidder.auctionContract.type == "DutchAuction") {
27                 try {
28                        await bidder.bid(bidValue);
29                 } catch (err) {
30                        console.log(err);
31                        // "reverting the UI" if something went wrong, alerting the user
32                        notifyTransactionError("transaction reverted");
33                        $("#bidButton").show();
34                 }
35          } else{
36                 let nonce = $("#nonceChoosen").val();
37                 let deposit = $("#depositRequired").val();
38                 try {
39                        await bidder.commitBid(bidValue,nonce,deposit);
40                 } catch (err) {
41                        // "reverting the UI" if something went wrong, alerting the user
42                        notifyTransactionError("transaction reverted");
43                        $("#bidButton").show();
44
45                 }
46          }
47
48   });
49
50
51   $("#withdrawButton").click(async function () {
52          // withdrawing
53
54          $("#withdrawButton").hide();
55
56          try {
57                 await bidder.withdraw();
58          } catch (err) {
59                 // "reverting the UI" if something went wrong, alerting the user
60                 notifyTransactionError("transaction reverted");
61                 $("#withdrawButton").show();
62
63          }
64   });
65
66
67   $("#openButton").click(async function () {
68          // withdrawing
69
70
71          $("#openButton").hide();
72          let bidValue = $("#bidValue").val();
73          let nonce = $("#nonceChoosen").val();
```

```
74        try {
75                await bidder.open(nonce, bidValue);
76        } catch (err) {
77                console.log(err);
78                // "reverting the UI" if something went wrong, alerting the user
79                notifyTransactionError("transaction reverted");
80                $("#openButton").show();
81
82        }
83 });
84
85
86 $("#joinAuctionModal").click(function () {
87        // joining the created Auction
88        bidder.connectToContract();
89        $("#auctionCreatedModal").modal("hide");
90        $("#currentAuctionCard").show();
91
92        $("#withdrawButton").hide();
93        $("#openButton").hide();
94
95        $("#joinedNewAuctionSuccess").show();
96
97        $("#contractFunctionsCard").show();
98 })
```

9   BidderUI.js

```
1
2 var bidderUI = null;
3
4 class BidderUI extends UserInterface {
5        // AuctionHouse events
6        newSellerSubscribed(sellerAddress) { return; } // does nothing
7
8        newBidderSubscribed(bidderAddress) {
9                if (bidderAddress.toLowerCase() == bidder.pubKey) {
10                        // notify the user of a successful subscription to the AuctionHouse contract
11                        hideSpinnerNextTo("#subscribeToAuctionHouse");
12                        $("#subscribeToAuctionHouseSuccess").show();
13                }
14        }
15
16        newAuctionSubmitted(sellerAddress, objectDescription) {
17                console.log("A new auction has been submitted by a seller. object: " + objectDescription
                        );
18        }
19
20        auctionDeployedSuccessfully(auctionAddress, auctionType, objectDesciption) {
21
22                bidder.notifyNewAuction(auctionAddress, auctionType, objectDesciption);
23
24                changeViewBasedOn(auctionType);
25                // showing the modal to join the new auction
26                $("#contractAddressModal").text(auctionAddress);
27                $("#auctionTypeModal").text(auctionType);
28                $("#objectDescriptionModal").text(objectDesciption);
29                $("#auctionCreatedModal").modal("toggle");
30        }
31
32        // both Dutch and Vickrey events
33        notifyWinner(winnerAddress, bid) {
34                // showing the winner
35                hideSpinnerNextTo("#bidButton");
36                console.log(winnerAddress + " won bidding " + bid);
37
38                addAlertElement("<strong>" + winnerAddress + "</strong> won!","success");
39        }
40
41        // displaying the just added block number
```

```
42          newBlock(blockNumber) {
43                  console.log("Block added " + blockNumber);
44                  $("#addBlockResult").text(blockNumber);
45          }
46
47          // notify that the escrow has been accepted
48          escrowAccepted(address) {
49                  if(address.toLowerCase() == bidder.pubKey){
50                          $("#acceptEscrowResultSuccess").show();
51                          $("#refuseEscrowListElem").hide();
52                  }
53          }
54
55          // notify that the escrow has been refused
56          escrowRefused(address) {
57                  if(address.toLowerCase() == bidder.pubKey){
58                          $("#refuseEscrowResultSuccess").show();
59                          $("#acceptEscrowListElem").hide();
60                  }
61          }
62
63          // notify that the escrow has been concluded
64          escrowClosed() {
65                  $("#concludeEscrowResultSuccess").show();
66
67                  addAlertElement("Escrow Closed successfully","success");
68          }
69
70          // Dutch event
71
72          notifyNotEnoughMoney(bidderAddress, bidSent, actualPrice) {
73
74                  if (bidderAddress.toLowerCase() == bidder.pubKey) {
75                          // telling the user that his bid wasn't enough
76                          hideSpinnerNextTo("#bidButton");
77                          $("#bidButton").show();
78                          console.log("You bidded " + bidSent + " but the actual price was " + actualPrice
                                );
79
80                          addAlertElement("You bidded <strong>" + bidSent + "</strong> but the actual
                                price was <strong>" + actualPrice + "</strong>","warning");
81                  }
82          }
83
84          // Vickrey events
85
86          notifyCommittedEnvelop(bidderAddress) {
87                  console.log("Envelop commited " + bidderAddress);
88                  if(bidderAddress.toLowerCase() == bidder.pubKey){
89                          addAlertElement("Envelop committed","success");
90
91                          $("#withdrawButton").show();
92                          $("#openButton").show();
93
94                  }
95          }
96
97          notifyWithdraw(bidderAddress) {
98                  console.log("Withdrawal " + bidderAddress);
99                  if(bidderAddress.toLowerCase() == bidder.pubKey){
100                         addAlertElement("Withdrawn","success");
101                 }
102         }
103
104         notifyOpen(bidderAddress, value) {
105                 console.log("Open " + bidderAddress + " bid " + value);
106                 if(bidderAddress.toLowerCase() == bidder.pubKey){
107                         addAlertElement("Opened","success");
108
109                 }
110         }
```

17

```
111
112          notifyFirstBid(bidderAddress, value) {
113                  console.log("First bid " + bidderAddress + " bid " + value);
114                  addAlertElement("First bid <strong>" + bidderAddress + "</strong> bid <strong>" + value
                         + "</strong>" ,"secondary");
115          }
116
117          notifySecondBid(bidderAddress, value) {
118                  console.log("second bid " + bidderAddress + " bid " + value);
119                  addAlertElement("Second bid <strong>" + bidderAddress + "</strong> bid <strong>" + value
                         + "</strong>" ,"secondary");
120          }
121  }
122
123
124  // called when the window loads
125  $(window).on('load', function () {
126          bidderUI = new BidderUI();
127  });
```

10 Bidder.js

```
1
2    // variable storing the bidder (user) informations
3    var bidder = null;
4
5    class Bidder extends User {
6            constructor() {
7                    super();
8
9            }
10
11           // connecting to the AuctionHouse
12           async subscribeToAuctionHouse(auctionHouseAddress) {
13                   this.auctionHouseContract = new AuctionHouse();
14                   await this.auctionHouseContract.connect(App.provider.getSigner(),auctionHouseAddress );
15
16                   this.auctionHouseContract.registerToEvents(bidderUI);
17
18                   try{
19                           await this.auctionHouseContract.subscribeAsBidder();
20                   }catch(err){
21                           // already subscribed
22                           notifyTransactionError("Probably that address is already subscribed");
23                           bidderUI.newBidderSubscribed(this.pubKey);
24                   }
25           }
26
27           notifyNewAuction(auctionAddress, auctionType, objectDesciption){
28                   if(auctionType == "VickreyAuction"){
29                           this.auctionContract = new VickreyAuction();
30                   } else {
31                           this.auctionContract = new DutchAuction();
32                   }
33                   this.auctionContract.objectDescription = objectDesciption;
34                   this.auctionContract.contractAddress = auctionAddress;
35           }
36
37           // connecting to the deployed contract
38           async connectToContract() {
39                   await this.auctionContract.connect( App.provider.getSigner(), this.auctionContract.
                          contractAddress);
40                   this.auctionContract.registerToEvents(bidderUI);
41           }
42
43           // bidding a value to the Auction
44           async bid(bidValue) {
45                   await this.auctionContract.bid(bidValue);
46           }
47           async commitBid(bidValue, nonce, deposit){
```

```
48                        await this.auctionContract.commitBid(bidValue,nonce,deposit);
49            }
50
51            async withdraw(){
52                        await this.auctionContract.withdraw();
53            }
54
55            async open(nonce, bid){
56                        await this.auctionContract.open(nonce,bid);
57            }
58
59    }
60
61    // called when the window loads
62    $(window).on('load', function () {
63            bidder = new Bidder();
64    });
```

11   auctioneerView.js

```
1
2    $("#metamaskAccountUsedBtn").click(function () {
3            // showing the card to deploy the auction house and that one with the sellers and bidders lists
4            $("#auctionHouseCard").show();
5            $("#sellersAndBiddersListCard").show();
6
7            showSpinnerNextTo("#auctionHouseContractAddressDeployBtn");
8            // checking if there is already an auction deployed
9            $.ajax({
10                   type: "GET",
11                   url: "auctionhouse/address",
12                   dataType: 'json',
13                   success: function (data) {
14
15                           if (data.contractAddress != "") {
16                                   // an AuctionHouse already exists
17                                   auctioneerUI.setAuctionHouseAddress(data.contractAddress);
18                                   auctioneer.connectToAuctionHouse(data.contractAddress);
19
20                                   // getting the subscribed sellers
21                                   $.ajax({
22                                           type: "GET",
23                                           url: "auctionhouse/subscribedsellers",
24                                           dataType: 'json',
25                                           success: function (data) {
26                                                   data.sellers.forEach(function (elem) {
27                                                           $("#subscribedSellersList").append("<li class='list-
                                                                   group-item'>" + elem + "</li>");
28                                                   })
29                                           }
30                                   })
31                                   // getting the subscribed bidders
32                                   $.ajax({
33                                           type: "GET",
34                                           url: "auctionhouse/subscribedbidders",
35                                           dataType: 'json',
36                                           success: function (data) {
37                                                   data.bidders.forEach(function (elem) {
38                                                           $("#subscribedBiddersList").append("<li class='list-
                                                                   group-item'>" + elem + "</li>");
39                                                   })
40                                           }
41                                   })
42                           } else {
43                                   // the AuctionHouse needs to be deployed
44                                   hideSpinnerNextTo("#auctionHouseContractAddressDeployBtn");
45                                   $("#auctionHouseContractAddressDeployBtn").show();
46                           }
47
48                   }
```

```
49            });
50      });
51
52      $("#auctionHouseContractAddressDeployBtn").click(async function () {
53            // deploying the AuctionHouse contract
54            $("#auctionHouseContractAddressDeployBtn").hide();
55            showSpinnerNextTo("#auctionHouseContractAddressDeployBtn");
56            $("#auctionHouseContractAddress").text("");
57
58            auctioneer.init();
59      })
60
61      $("#auctionType").change(function () {
62            let type = $("#auctionType option:selected").text();
63
64            changeViewBasedOn(type);
65      })
66
67
68      $("#deployContract").click(async function () {
69            // deploying the Auction contract
70            showSpinnerNextTo("#deployContract");
71            $("#deployContract").hide();
72            $("#newAuctionSubmitted").hide();
73
74            let auctionType = $("#auctionType option:selected").text();
75            let objectDescription = $("#currentAuctionObjectDescription").text();
76
77            if (auctionType == "DutchAuction") {
78                    // deploying DutchAuction
79                    let strategy = $("#decreasingStrategy option:selected").text();
80                    let _reservePrice = $("#_reservePrice").val();
81                    let _initialPrice = $("#_initialPrice").val();
82                    let _openedForLength = $("#_openedForLength").val();
83                    let _seller = $("#_sellerAddress").val();
84                    let miningRate = $("#miningRate").val();
85
86                    try {
87                            await auctioneer.initDutchAuction(strategy, _reservePrice, _initialPrice,
88                                    _openedForLength, _seller, miningRate, objectDescription);
                    } catch (err) {
89                            console.log(err);
90                            // "reverting the UI" if something went wrong, and notify the user
91                            notifyTransactionError("transaction reverted");
92                            $("#deployContract").show();
93                            hideSpinnerNextTo("#deployContract");
94                    }
95
96            } else if (auctionType == "VickreyAuction") { // deploying VickreyAuction
97                    let _reservePrice = $("#_reservePrice").val();
98                    let _depositRequired = $("#_depositRequired").val();
99                    let _commitmentPhaseLength = $("#_commitmentPhaseLength").val();
100                   let _withdrawalPhaseLength = $("#_withdrawalPhaseLength").val();
101                   let _openingPhaseLength = $("#_openingPhaseLength").val();
102                   let _seller = $("#_sellerAddress").val();
103                   let miningRate = $("#miningRate").val();
104
105                   try {
106                           await auctioneer.initVickreyAuction(_reservePrice, _commitmentPhaseLength,
                                   _withdrawalPhaseLength, _openingPhaseLength, _depositRequired, _seller,
                                   miningRate, objectDescription);
107                   } catch (err) {
108                           console.log(err);
109                           // "reverting the UI" if something went wrong, and notify the user
110                           notifyTransactionError("transaction reverted");
111                           $("#deployContract").show();
112                           hideSpinnerNextTo("#deployContract");
113                   }
114           } else {
115                   throw "auction type " + auctionType;
116           }
```

20

```
117
118
119     });
120
121     $("#finalize").click(async function () {
122             try {
123                     await auctioneer.finalize();
124             } catch (err) {
125                     notifyTransactionError("transaction reverted");
126             }
127     })
128
129     $("#destroyContract").click(async function () {
130             await auctioneer.destroyContracts();
131     })
```

12 AuctioneerUI.js

```
1
2     var auctioneerUI = null;
3
4     class AuctioneerUI extends UserInterface {
5             // AuctionHouse events
6
7             // adding an address to the sellers' list
8             newSellerSubscribed(sellerAddress) {
9                     $("#subscribedSellersList").append("<li class='list-group-item'>" + sellerAddress + "</
                        li>");
10                    $.ajax({
11                            type: "POST",
12                            url: "auctionhouse/" + auctioneer.auctionHouse.contract.address  + "/seller",
13                            dataType: 'json',
14                            contentType: 'application/json',
15                            data: JSON.stringify({ "sellerAddress": sellerAddress })
16                    })
17            }
18
19            // adding an address to the bidder's list
20            newBidderSubscribed(bidderAddress) {
21                    $("#subscribedBiddersList").append("<li class='list-group-item'>" + bidderAddress + "</
                        li>");
22                    $.ajax({
23                            type: "POST",
24                            url: "auctionhouse/" + auctioneer.auctionHouse.contract.address  + "/bidder",
25                            dataType: 'json',
26                            contentType: 'application/json',
27                            data: JSON.stringify({ "bidderAddress": bidderAddress })
28                    })
29            }
30
31            // displaying the card to deploy a new auction
32            newAuctionSubmitted(sellerAddress, objectDescription) {
33                    $("#auctionCard").show();
34                    $("#newAuctionSubmitted").text("New");
35                    $("#newAuctionSubmitted").show();
36
37                    $("#currentAuctionHeader").text("A new Auction has been submitted!");
38                    $("#currentAuctionObjectDescription").text(objectDescription);
39                    $("#_sellerAddress").val(sellerAddress);
40            }
41
42            auctionDeployedSuccessfully(auctionAddress, auctionType, objectDesciption) {
43                    console.log("new auction create " + auctionType + " description " + objectDesciption);
44
45                    changeViewBasedOn(auctionType);
46
47                    hideSpinnerNextTo("#deployContract");
48                    $("#newAuctionSubmitted").text("Success");
49                    $("#newAuctionSubmitted").show();
50
```

```
51                  $("#contractFunctionsCard").show();
52          }
53
54          // both Dutch and Vickrey events
55          notifyWinner(winnerAddress, bid) {
56                  addAlertElement("<strong>" + winnerAddress + "</strong> won!","success");
57
58                  if(auctioneer.auctionContract.type == "VickreyAuction"){
59                          $("#finalizeSuccess").show();
60                  }
61          }
62
63          // displaying the just added block number
64          newBlock(blockNumber) {
65                  console.log("Block added " + blockNumber);
66                  $("#addBlockResult").text(blockNumber);
67          }
68
69          // notify that the escrow has been accepted
70          escrowAccepted(address) {
71                  if(address.toLowerCase() == auctioneer.pubKey){
72                          $("#acceptEscrowResultSuccess").show();
73                          $("#refuseEscrowListElem").hide();
74                  }
75          }
76
77          // notify that the escrow has been refused
78          escrowRefused(address) {
79                  if(address.toLowerCase() == auctioneer.pubKey){
80                          $("#refuseEscrowResultSuccess").show();
81                          $("#acceptEscrowListElem").hide();
82                  }
83          }
84
85          // notify that the escrow has been concluded
86          escrowClosed() {
87                  $("#concludeEscrowResultSuccess").show();
88
89                  addAlertElement("Escrow Closed successfully","success");
90
91                  // only visible to auctionhouse
92                  $("#destroyContractListElement").show();
93          }
94
95          // Dutch event
96          notifyNotEnoughMoney(bidderAddress, bidSent, actualPrice) { return; } // does nothing
97
98          // Vickrey events
99
100         notifyCommittedEnvelop(bidderAddress) {
101                 addAlertElement("Envelop committed by <strong>" + bidderAddress + "</strong>","secondary
                            ");
102                 console.log("Envelop commited " + bidderAddress);
103         }
104
105         notifyWithdraw(bidderAddress) {
106                 console.log("Withdrawal " + bidderAddress);
107                 addAlertElement("Withdrawal by by <strong>" + bidderAddress + "</strong>","secondary");
108         }
109
110         notifyOpen(bidderAddress, value) {
111                 console.log("Open " + bidderAddress + " bid " + value);
112                 addAlertElement("<strong>" + bidderAddress + "</strong> opened the envelop","secondary")
                            ;
113         }
114
115         notifyFirstBid(bidderAddress, value) {
116                 console.log("First bid " + bidderAddress + " bid " + value);
117                 addAlertElement("First bid <strong>" + bidderAddress + "</strong> bid <strong>" + value
                            + "</strong>" ,"secondary");
118         }
```

```
119
120            notifySecondBid(bidderAddress, value) {
121                    console.log("second bid " + bidderAddress + " bid " + value);
122                    addAlertElement("Second bid <strong>" + bidderAddress + "</strong> bid <strong>" + value
                            + "</strong>" ,"secondary");
123            }
124
125
126            // new implemented methods
127
128            // displaying the AuctionHouse contract's address
129            setAuctionHouseAddress(address) {
130                    hideSpinnerNextTo("#auctionHouseContractAddressDeployBtn");
131
132                    $("#auctionHouseContractAddress").text(address);
133            }
134    }
135
136
137    // called when the window loads
138    $(window).on('load', function () {
139            auctioneerUI = new AuctioneerUI();
140    });
```

13 Auctioneer.js

```
1
2    // variable storing the auctionhouse (user) informations
3    var auctioneer = null;
4
5    class Auctioneer extends User { // auctioneer
6            constructor() {
7                    super();
8                    this.decreasingStrategy = null;
9                    this.auctionHouse = null;
10            }
11
12            // deploying the Auction House contract
13            async init() {
14                    this.auctionHouse = new AuctionHouse();
15                    let address = await this.auctionHouse.deploy(App.provider.getSigner());
16
17                    // diplaying the contract address
18                    auctioneerUI.setAuctionHouseAddress(address);
19
20                    // sending the auction house address to the server so that it can be taken by the bidder
                            and the seller
21                    $.ajax({
22                            type: "POST",
23                            url: "auctionhouse/address",
24                            dataType: 'json',
25                            contentType: 'application/json',
26                            data: JSON.stringify({ "contractAddress": address })
27                    })
28
29                    this.auctionHouse.registerToEvents(auctioneerUI);
30            }
31
32                    // connecting to the deployed contract
33            async connectToAuctionHouse(address) {
34                    this.auctionHouse = new AuctionHouse();
35                    await this.auctionHouse.connect(App.provider.getSigner(),address );
36
37                    this.auctionHouse.registerToEvents(auctioneerUI);
38            }
39
40
41            // deploying the Dutch Auction
42            async initDutchAuction(strategy, _reservePrice, _initialPrice, _openedForLength, _seller,
                    miningRate, _objectDesciption) {
```

```
43
44              this.decreasingStrategy = new DecreasingStrategy(strategy);
45              await this.decreasingStrategy.deploy(App.provider.getSigner());
46
47              this.auctionContract = new DutchAuction();
48              this.auctionContract.objectDesciption = _objectDesciption;
49
50              await this.auctionContract.deploy(App.provider.getSigner(), _reservePrice, _initialPrice
                    , _openedForLength, _seller, this.decreasingStrategy.strategy.address, miningRate);
51
52              this.auctionContract.registerToEvents(auctioneerUI);
53
54              this.auctionHouse.notifyNewAuction(this.auctionContract.contractAddress, this.
                    auctionContract.type, this.auctionContract.objectDesciption);
55          }
56
57      async initVickreyAuction(_reservePrice, _commitmentPhaseLength,_withdrawalPhaseLength,
            _openingPhaseLength, _depositReuired, _seller, miningRate, _objectDesciption){
58
59              this.auctionContract = new VickreyAuction();
60              this.auctionContract.objectDesciption = _objectDesciption;
61
62              await this.auctionContract.deploy(App.provider.getSigner(), _reservePrice,
                    _commitmentPhaseLength,_withdrawalPhaseLength,_openingPhaseLength, _depositReuired,
                    _seller, miningRate);
63
64              this.auctionContract.registerToEvents(auctioneerUI);
65
66              this.auctionHouse.notifyNewAuction(this.auctionContract.contractAddress, this.
                    auctionContract.type, this.auctionContract.objectDesciption);
67          }
68
69      async finalize(){
70      await this.auctionContract.finalize();
71          }
72
73      async destroyContracts() {
74              await this.auctionContract.destroy();
75              if (this.decreasingStrategy != null)
76                      await this.decreasingStrategy.destroy();
77              await this.auctionHouse.destroy();
78
79              $.ajax({
80                      type: "POST",
81                      url: "auctionhouse/address",
82                      dataType: 'json',
83                      contentType: 'application/json',
84                      data: JSON.stringify({ "contractAddress": "" })
85              })
86          }
87  }
88
89  // called when the window loads
90  $(window).on('load', function () {
91          auctioneer = new Auctioneer();
92  });
```

14   simpleserver.js

```
1           var express = require('express');
2           var app = express();
3           const path = require('path');
4           const router = express.Router();
5
6           // variables keeping the state of the server
7           var auctionHouseContractAddress = "";
8           var subscribedSellers = [];
9           var subscribedBidders = [];
10
11          app.use(express.json());
```

```
12
13          app.get("/",function(req,res){
14                  res.sendFile( path.join(__dirname+'/src/index.html'));
15          })
16
17          app.get("/auctionhouse",function(req,res){
18                  res.sendFile( path.join(__dirname+'/src/auctionhouse.html'));
19          })
20
21          app.get("/bidder",function(req,res){
22                  res.sendFile( path.join(__dirname+'/src/bidder.html'));
23          })
24
25          app.get("/seller",function(req,res){
26                  res.sendFile( path.join(__dirname+'/src/seller.html'));
27          })
28
29          app.post("/auctionhouse/address",function(req,res){
30                  auctionHouseContractAddress = req.body.contractAddress;
31                  console.log(auctionHouseContractAddress);
32
33                  if(auctionHouseContractAddress == ""){
34                          // resetting the lists
35                          subscribedBidders = [];
36                          subscribedSellers = [];
37                  }
38
39                  res.sendStatus(200);
40          })
41
42          app.get("/auctionhouse/address",function(req,res){
43                  res.json({ contractAddress : auctionHouseContractAddress });
44          })
45
46          // managing subscribed sellers and bidders if AuctionHouse already deployed
47          app.post("/auctionhouse/:address/seller",function(req,res){
48                  if(req.params.address == auctionHouseContractAddress){
49                          sellerAddress = req.body.sellerAddress;
50
51                          if(subscribedSellers.includes(sellerAddress) == false){
52                                  console.log("new seller " + sellerAddress);
53                                  subscribedSellers.push(sellerAddress);
54                          }
55
56                          res.sendStatus(200);
57                  }
58          })
59
60          app.post("/auctionhouse/:address/bidder",function(req,res){
61                  if(req.params.address == auctionHouseContractAddress){
62                          bidderAddress = req.body.bidderAddress;
63
64                          if(subscribedBidders.includes(bidderAddress) == false){
65                                  console.log("new bidder " + bidderAddress);
66                                  subscribedBidders.push(bidderAddress);
67                          }
68                          res.sendStatus(200);
69                  }
70          })
71
72          app.get("/auctionhouse/subscribedsellers",function(req,res){
73                  res.json({ sellers : subscribedSellers });
74          })
75
76          app.get("/auctionhouse/subscribedbidders",function(req,res){
77                  res.json({ bidders : subscribedBidders });
78          })
79
80
81
82
```

```
83
84          app.use(express.static(__dirname + '/src'));
85          app.use(express.static(__dirname + '/build/contracts'));
86
87
88          app.use('/', router);
89          app.listen(process.env.port || 3002);
90
91          console.log('Running at Port 3002');
92          console.log('http://localhost:3002/');
```