1 UI.js

```
1
2   class UserInterface{
3           // AuctionHouse events
4           newSellerSubscribed(sellerAddress){
5                   throw "This function needs to be implemented in the subclasses";
6           }
7
8           newBidderSubscribed(bidderAddress){
9                   throw "This function needs to be implemented in the subclasses";
10          }
11
12          newAuctionSubmitted(sellerAddress, objectDescription){
13                  throw "This function needs to be implemented in the subclasses";
14          }
15
16          auctionDeployedSuccessfully(auctionAddress, auctionName, objectDesciption){
17                  throw "This function needs to be implemented in the subclasses";
18          }
19
20          // both Dutch and Vickrey events
21          notifyWinner(winnerAddress, bid){
22                  throw "This function needs to be implemented in the subclasses";
23          }
24
25          newBlock(blockNumber){
26                  throw "This function needs to be implemented in the subclasses";
27          }
28
29          escrowAccepted(address){
30                  throw "This function needs to be implemented in the subclasses";
31          }
32
33          escrowRefused(address){
34                  throw "This function needs to be implemented in the subclasses";
35          }
36
37          escrowClosed(){
38                  throw "This function needs to be implemented in the subclasses";
39          }
40
41          // Dutch event
42          notifyNotEnoughMoney(bidderAddress, bidSent, actualPrice){
43                  throw "This function needs to be implemented in the subclasses";
44          }
45
46          // Vickrey events
47
48          notifyCommittedEnvelop(bidderAddress){
49                  throw "This function needs to be implemented in the subclasses";
50          }
51
52          notifyWithdraw(bidderAddress){
53                  throw "This function needs to be implemented in the subclasses";
54          }
55
56          notifyOpen(bidderAddress, value){
57                  throw "This function needs to be implemented in the subclasses";
58          }
59
60          notifyFirstBid(bidderAddress,value){
61                  throw "This function needs to be implemented in the subclasses";
62          }
63
64          notifySecondBid(bidderAddress,value){
65                  throw "This function needs to be implemented in the subclasses";
66          }
67  }
```

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          }
```

```
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
```

```
        // bidding a value to the Auction
        async bid(bidValue) {
                let overrides = {
                gasLimit: 6000000,
                value: ethers.utils.parseEther(bidValue)
                };

                await this.contract.bid(overrides);

        }

        getInitialPrice(){
                return this.contract.getInitialPrice();
        }

        async getCurrentPrice(){
                return await this.contract.getCurrentPrice();
        }

        async getOpenedFor(){
                return await this.contract.getOpenedFor();
        }
}

class VickreyAuction extends Auction{
        constructor(){
                super("VickreyAuction");
        }

        async deploy(signer, _reservePrice, _commitmentPhaseLength, _withdrawalPhaseLength,
            _openingPhaseLenght, _depositRequired, _seller, miningRate){
                // deploying the DutchAuction
                let factory = await this.getContractFactory(signer);

                _reservePrice = ethers.utils.parseEther(_reservePrice);
                _depositRequired = ethers.utils.parseEther(_depositRequired);
                this.contract = await factory.deploy(_reservePrice, _commitmentPhaseLength,
                    _withdrawalPhaseLength, _openingPhaseLenght, _depositRequired, _seller, miningRate);

                console.log(this.contract.address);
                this.contractAddress = this.contract.address;

                // The contract is NOT deployed yet; we must wait until it is mined
                await this.contract.deployed()
        }

        registerToEvents(ui){
                super.registerToEvents(ui);

                this.contract.on("CommittedEnvelop", (bidderAddress) => {
                        ui.notifyCommittedEnvelop(bidderAddress);
                });

                this.contract.on("Withdraw", (bidderAddress) => {
                        ui.notifyWithdraw(bidderAddress);
                });
                this.contract.on("Open", (bidderAddress, value) => {
                        ui.notifyOpen(bidderAddress,value);
                });

                this.contract.on("FirstBid", (bidderAddress, value) => {
                        ui.notifyFirstBid(bidderAddress,value);
                });

                this.contract.on("SecondBid", (bidderAddress, value) => {
                        ui.notifySecondBid(bidderAddress,value);
                });
        }

        async commitBid(bid, nonce, depositRequired){
                let overrides = {
```

```
                gasLimit: 50000,
                value: ethers.utils.parseEther(depositRequired)
                };
                // we use the utility provided by the contract so that we are sure that the parameters
                    are passes correctly to the function
                let envelop = await this.contract.doKeccak( ethers.utils.parseEther(bid),nonce);
                await this.contract.commitBid(envelop,overrides);
        }

        async withdraw(){
                await this.contract.withdraw();
        }

        async open(nonce, bid){
                let overrides = {
                gasLimit: 60000,
                value: ethers.utils.parseEther(bid)
                };
                await this.contract.open(nonce,overrides);
        }

        async finalize(){
                let overrides = {
                gasLimit: 6000000
                };
                await this.contract.finalize(overrides);
        }

        async getDepositRequired(){
                return await this.contract.getDepositRequired();
        }

        async getCommitmentPhaseLength(){
                return await this.contract.getCommitmentPhaseLength();
        }

        async getWithdrawalPhaseLength(){
                return await this.contract.getWithdrawalPhaseLength();
        }

        async getOpeningPhaseLength(){
                return await this.contract.getOpeningPhaseLength();
        }
}

class AuctionHouse{
        constructor(){
                this.contract = null;
        }

        async deploy(signer){
                let auctionHouseJSON = await $.getJSON("AuctionHouse.json");
                let auctionHouseFactory = new ethers.ContractFactory(auctionHouseJSON.abi,
                    auctionHouseJSON.bytecode, signer);

                this.contract = await auctionHouseFactory.deploy();
                await this.contract.deployed();

                console.log(this.contract.address);

                return this.contract.address;
        }

        async connect(signer, auctionHouseAddress){
                let c = await $.getJSON("AuctionHouse.json");

                this.contract = new ethers.Contract(auctionHouseAddress, c.abi, signer);
                console.log("connected to auction house");
        }

        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         }
```

```
});

// alling the getGracePeiod on the deployed contract
// this function should be used to see if the auction is opened
$("#getGracePeriod").click(async function () {
        let user = getUser();

        if (user != null && user.auctionContract != null) {
                let gracep = await user.auctionContract.getGracePeriod();
                if (gracep <= 0) {
                        $("#getGracePeriodSuccess").show();
                        $("#getGracePeriodResult").hide();
                } else {
                        $("#getGracePeriodResult").text(gracep.toString());
                }

        }

});

// calling getOpenedFor on the deployed contract
// alert is showed if the transaction is reverted
$("#getOpenedFor").click(async function () {
        let user = getUser();

        if (user != null && user.auctionContract != null) {
                try {
                        let openedfor = await user.auctionContract.getOpenedFor();
                        $("#getOpenedForDanger").hide();
                        $("#getOpenedForResult").text(openedfor.toString());
                } catch (err) {
                        // console.log(err);
                        $("#getOpenedForDanger").show();
                        $("#getOpenedForResult").hide();
                }
        }
});


// calling getDepositRequired on the deployed contract
// alert is showed if the transaction is reverted
$("#getDepositRequired").click(async function () {
        let user = getUser();

        if (user != null && user.auctionContract != null) {

                let deposit = await user.auctionContract.getDepositRequired();
                $("#getDepositRequiredResult").text(deposit.toString());
        }
});


// calling getCommitmentPhaseLength on the deployed contract
// alert is showed if the transaction is reverted
$("#getCommitmentPhaseLength").click(async function () {
        let user = getUser();

        if (user != null && user.auctionContract != null) {
                try {
                        let phasel = await user.auctionContract.getCommitmentPhaseLength();
                        $("#getCommitmentPhaseLengthResult").text(phasel.toString());
                        $("#getCommitmentPhaseLengthResult").show();
                        $("#getCommitmentPhaseLengthDanger").hide();
                } catch (err) {
                        // notifyTransactionError("transaction reverted");
                        $("#getCommitmentPhaseLengthResult").hide();
                        $("#getCommitmentPhaseLengthDanger").show();
                }

        }
});
```

```
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();
```

```
});
```

5 sellerView.js

```
$("#metamaskAccountUsedBtn").click(function () {
        // showing the card to subscribe to the AuctionHouse
        $("#subscribeToAuctionHouseCard").show();

        showSpinnerNextTo("#subscribeToAuctionHouse");

        setTimeout(checkForAuctionHouseAddress, 1000);
});

$("#subscribeToAuctionHouse").click(function () {
        // subscribing to the AuctionHouse
        $("#subscribeToAuctionHouse").hide();
        showSpinnerNextTo("#subscribeToAuctionHouse");

        let address = $("#auctionHouseAddress").text();
        seller.subscribeToAuctionHouse(address);
});


$("#submitAuction").click(function () {
        // submitting a new auction
        $("#submitAuction").hide();
        showSpinnerNextTo("#submitAuction");

        let objectDescription = $("#objectDescription").val();
        seller.submitAuction(objectDescription);
});
```

6 SellerUI.js

```
var sellerUI = null

class SellerUI extends UserInterface {
        // AuctionHouse events
        newSellerSubscribed(sellerAddress) {
                if (sellerAddress.toLowerCase() == seller.pubKey) {
                        // notify a successfull subscription to the AuctionHouse
                        hideSpinnerNextTo("#subscribeToAuctionHouse");
                        $("#subscribeToAuctionHouseSuccess").show();
                        $("#submitCard").show();
                }
        }

        newBidderSubscribed(bidderAddress) { return; } // does nothing

        newAuctionSubmitted(sellerAddress, objectDescription) {
                if (sellerAddress.toLowerCase() == seller.pubKey) {
                        // notify that the auction was successfully submitted
                        hideSpinnerNextTo("#submitAuction");
                        $("#submitAuctionSuccess").show();
                        }
        }

        auctionDeployedSuccessfully(auctionAddress, auctionType, objectDesciption) {
                seller.notifyNewAuction(auctionAddress, auctionType, objectDesciption);

                changeViewBasedOn(auctionType);
                // showing the list of functions to interact with the contract
                $("#contractFunctionsCard").show();
        }

        // both Dutch and Vickrey events
        notifyWinner(winnerAddress, bid) {
```

```
                console.log(winnerAddress + " won bidding " + bid);

                addAlertElement("<strong>" + winnerAddress + "</strong>  won!","success");
        }

        // displaying the just added block number
        newBlock(blockNumber) {
                console.log("Block added " + blockNumber);
                $("#addBlockResult").text(blockNumber);
        }

        // notify that the escrow has been accepted
        escrowAccepted(address) {
                if(address.toLowerCase() == seller.pubKey){
                $("#acceptEscrowResultSuccess").show();
                $("#refuseEscrowListElem").hide();
                }
        }

        // notify that the escrow has been refused
        escrowRefused(address) {
                if(address.toLowerCase() == seller.pubKey){
                $("#refuseEscrowResultSuccess").show();
                $("#acceptEscrowListElem").hide();
                }
        }

        // notify that the escrow has been concluded
        escrowClosed() {
                $("#concludeEscrowResultSuccess").show();

                addAlertElement("Escrow Closed successfully","success");
        }

        // Dutch event
        notifyNotEnoughMoney(bidderAddress, bidSent, actualPrice) { return; } // does nothing

        // Vickrey events

        notifyCommittedEnvelop(bidderAddress) {
                addAlertElement("Envelop committed by <strong>" + bidderAddress + "</strong>","secondary
                    ");
                console.log("Envelop commited " + bidderAddress);
        }

        notifyWithdraw(bidderAddress) {
                console.log("Withdrawal " + bidderAddress);
                addAlertElement("Withdrawal by by <strong>" + bidderAddress + "</strong>","secondary");
        }

        notifyOpen(bidderAddress, value) {
                console.log("Open " + bidderAddress + " bid " + value);
                addAlertElement("<strong>" + bidderAddress + "</strong> opened the envelop","secondary")
                    ;
        }

        notifyFirstBid(bidderAddress, value) {
                console.log("First bid " + bidderAddress + " bid " + value);
                addAlertElement("First bid <strong>" + bidderAddress + "</strong> bid <strong>" + value
                    + "</strong>" ,"secondary");
        }

        notifySecondBid(bidderAddress, value) {
                console.log("second bid " + bidderAddress + " bid " + value);
                addAlertElement("Second bid <strong>" + bidderAddress + "</strong> bid <strong>" + value
                     + "</strong>" ,"secondary");
        }
}
```

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

7 Seller.js

```

// variable storing the sller (user) informations
let seller = null;

class Seller extends User{
        constructor(){
                super();
        }

        // connecting to the AuctionHouse
        async subscribeToAuctionHouse(auctionHouseAddress){
                this.auctionHouseContract = new AuctionHouse();
                await this.auctionHouseContract.connect(App.provider.getSigner(),auctionHouseAddress );

                this.auctionHouseContract.registerToEvents(sellerUI);

                try{
                        await this.auctionHouseContract.subscribeAsSeller();
                }catch(err){
                        // already subscribed
                        notifyTransactionError("Probably that address is already subscribed");
                        sellerUI.newSellerSubscribed(this.pubKey);
                }
        }


        notifyNewAuction(auctionAddress, auctionType, objectDesciption){
                if(auctionType == "VickreyAuction"){
                this.auctionContract = new VickreyAuction();
                } else {
                this.auctionContract = new DutchAuction();
                }

                this.auctionContract.objectDescription = objectDesciption;
                this.auctionContract.contractAddress = auctionAddress;
                this.connectToContract();
        }

        // submitting a new auction giving a description of the selled object
        submitAuction(objectDescription){
                this.auctionHouseContract.submitAuction(objectDescription);
        }

        // connecting to the deployed contract
        async connectToContract(){

                await this.auctionContract.connect( App.provider.getSigner(), this.auctionContract.
                    contractAddress);
                this.auctionContract.registerToEvents(sellerUI);
        }

}

// called when the window loads
$(window).on('load', function () {
        seller = new Seller();
});
```

8 bidderView.js

```

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

```
        // showing the card to subscribe to the AuctionHouse
        $("#subscribeToAuctionHouseCard").show();
        showSpinnerNextTo("#subscribeToAuctionHouse");

        setTimeout(checkForAuctionHouseAddress, 1000);
});

$("#subscribeToAuctionHouse").click(function () {
        // subscribing to the AuctionHouse
        $("#subscribeToAuctionHouse").hide();
        showSpinnerNextTo("#subscribeToAuctionHouse");

        let address = $("#auctionHouseAddress").text();
        bidder.subscribeToAuctionHouse(address);

});


$("#bidButton").click(async function () {
        // bidding

        $("#bidButton").hide();
        let bidValue = $("#bidValue").val();
        if (bidder.auctionContract.type == "DutchAuction") {
                try {
                        await bidder.bid(bidValue);
                } catch (err) {
                        console.log(err);
                        // "reverting the UI" if something went wrong, alerting the user
                        notifyTransactionError("transaction reverted");
                        $("#bidButton").show();
                }
        } else{
                let nonce = $("#nonceChoosen").val();
                let deposit = $("#depositRequired").val();
                try {
                        await bidder.commitBid(bidValue,nonce,deposit);
                } catch (err) {
                        // "reverting the UI" if something went wrong, alerting the user
                        notifyTransactionError("transaction reverted");
                        $("#bidButton").show();

                }
        }

});


$("#withdrawButton").click(async function () {
        // withdrawing

        $("#withdrawButton").hide();

        try {
                await bidder.withdraw();
        } catch (err) {
                // "reverting the UI" if something went wrong, alerting the user
                notifyTransactionError("transaction reverted");
                $("#withdrawButton").show();

        }
});


$("#openButton").click(async function () {
        // withdrawing


        $("#openButton").hide();
        let bidValue = $("#bidValue").val();
        let nonce = $("#nonceChoosen").val();
```

```
        try {
                await bidder.open(nonce, bidValue);
        } catch (err) {
                console.log(err);
                // "reverting the UI" if something went wrong, alerting the user
                notifyTransactionError("transaction reverted");
                $("#openButton").show();

        }
});


$("#joinAuctionModal").click(function () {
        // joining the created Auction
        bidder.connectToContract();
        $("#auctionCreatedModal").modal("hide");
        $("#currentAuctionCard").show();

        $("#withdrawButton").hide();
        $("#openButton").hide();

        $("#joinedNewAuctionSuccess").show();

        $("#contractFunctionsCard").show();
})
```

9 BidderUI.js

```

var bidderUI = null;

class BidderUI extends UserInterface {
        // AuctionHouse events
        newSellerSubscribed(sellerAddress) { return; } // does nothing

        newBidderSubscribed(bidderAddress) {
                if (bidderAddress.toLowerCase() == bidder.pubKey) {
                        // notify the user of a successful subscription to the AuctionHouse contract
                        hideSpinnerNextTo("#subscribeToAuctionHouse");
                        $("#subscribeToAuctionHouseSuccess").show();
                }
        }

        newAuctionSubmitted(sellerAddress, objectDescription) {
                console.log("A new auction has been submitted by a seller. object: " + objectDescription
                    );
        }

        auctionDeployedSuccessfully(auctionAddress, auctionType, objectDesciption) {

                bidder.notifyNewAuction(auctionAddress, auctionType, objectDesciption);

                changeViewBasedOn(auctionType);
                // showing the modal to join the new auction
                $("#contractAddressModal").text(auctionAddress);
                $("#auctionTypeModal").text(auctionType);
                $("#objectDescriptionModal").text(objectDesciption);
                $("#auctionCreatedModal").modal("toggle");
        }

        // both Dutch and Vickrey events
        notifyWinner(winnerAddress, bid) {
                // showing the winner
                hideSpinnerNextTo("#bidButton");
                console.log(winnerAddress + " won bidding " + bid);

                addAlertElement("<strong>" + winnerAddress + "</strong> won!","success");
        }

        // 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         }
```

```
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){
```

```
                await this.auctionContract.commitBid(bidValue,nonce,deposit);
        }

        async withdraw(){
                await this.auctionContract.withdraw();
        }

        async open(nonce, bid){
                await this.auctionContract.open(nonce,bid);
        }

}

// called when the window loads
$(window).on('load', function () {
        bidder = new Bidder();
});
```

11 auctioneerView.js

```

$("#metamaskAccountUsedBtn").click(function () {
        // showing the card to deploy the auction house and that one with the sellers and bidders lists
        $("#auctionHouseCard").show();
        $("#sellersAndBiddersListCard").show();

        showSpinnerNextTo("#auctionHouseContractAddressDeployBtn");
        // checking if there is already an auction deployed
        $.ajax({
                type: "GET",
                url: "auctionhouse/address",
                dataType: 'json',
                success: function (data) {

                        if (data.contractAddress != "") {
                                // an AuctionHouse already exists
                                auctioneerUI.setAuctionHouseAddress(data.contractAddress);
                                auctioneer.connectToAuctionHouse(data.contractAddress);

                                // getting the subscribed sellers
                                $.ajax({
                                        type: "GET",
                                        url: "auctionhouse/subscribedsellers",
                                        dataType: 'json',
                                        success: function (data) {
                                                data.sellers.forEach(function (elem) {
                                                $("#subscribedSellersList").append("<li class='list-
                                                    group-item'>" + elem + "</li>");
                                                })
                                        }
                                })
                                // getting the subscribed bidders
                                $.ajax({
                                        type: "GET",
                                        url: "auctionhouse/subscribedbidders",
                                        dataType: 'json',
                                        success: function (data) {
                                                data.bidders.forEach(function (elem) {
                                                $("#subscribedBiddersList").append("<li class='list-
                                                    group-item'>" + elem + "</li>");
                                                })
                                        }
                                })
                        } else {
                                // the AuctionHouse needs to be deployed
                                hideSpinnerNextTo("#auctionHouseContractAddressDeployBtn");
                                $("#auctionHouseContractAddressDeployBtn").show();
                        }

                }
```

```
        });
});

$("#auctionHouseContractAddressDeployBtn").click(async function () {
        // deploying the AuctionHouse contract
        $("#auctionHouseContractAddressDeployBtn").hide();
        showSpinnerNextTo("#auctionHouseContractAddressDeployBtn");
        $("#auctionHouseContractAddress").text("");

        auctioneer.init();
})

$("#auctionType").change(function () {
        let type = $("#auctionType option:selected").text();

        changeViewBasedOn(type);
})


$("#deployContract").click(async function () {
        // deploying the Auction contract
        showSpinnerNextTo("#deployContract");
        $("#deployContract").hide();
        $("#newAuctionSubmitted").hide();

        let auctionType = $("#auctionType option:selected").text();
        let objectDescription = $("#currentAuctionObjectDescription").text();

        if (auctionType == "DutchAuction") {
                // deploying DutchAuction
                let strategy = $("#decreasingStrategy option:selected").text();
                let _reservePrice = $("#_reservePrice").val();
                let _initialPrice = $("#_initialPrice").val();
                let _openedForLength = $("#_openedForLength").val();
                let _seller = $("#_sellerAddress").val();
                let miningRate = $("#miningRate").val();

                try {
                        await auctioneer.initDutchAuction(strategy, _reservePrice, _initialPrice,
                            _openedForLength, _seller, miningRate, objectDescription);
                } catch (err) {
                        console.log(err);
                        // "reverting the UI" if something went wrong, and notify the user
                        notifyTransactionError("transaction reverted");
                        $("#deployContract").show();
                        hideSpinnerNextTo("#deployContract");
                }

        } else if (auctionType == "VickreyAuction") { // deploying VickreyAuction
                let _reservePrice = $("#_reservePrice").val();
                let _depositRequired = $("#_depositRequired").val();
                let _commitmentPhaseLength = $("#_commitmentPhaseLength").val();
                let _withdrawalPhaseLength = $("#_withdrawalPhaseLength").val();
                let _openingPhaseLength = $("#_openingPhaseLength").val();
                let _seller = $("#_sellerAddress").val();
                let miningRate = $("#miningRate").val();

                try {
                        await auctioneer.initVickreyAuction(_reservePrice, _commitmentPhaseLength,
                            _withdrawalPhaseLength, _openingPhaseLength, _depositRequired, _seller,
                            miningRate, objectDescription);
                } catch (err) {
                        console.log(err);
                        // "reverting the UI" if something went wrong, and notify the user
                        notifyTransactionError("transaction reverted");
                        $("#deployContract").show();
                        hideSpinnerNextTo("#deployContract");
                }
        } else {
                throw "auction type " + auctionType;
        }
```

```

});

$("#finalize").click(async function () {
        try {
                await auctioneer.finalize();
        } catch (err) {
                notifyTransactionError("transaction reverted");
        }
})

$("#destroyContract").click(async function () {
        await auctioneer.destroyContracts();
})
```

12 AuctioneerUI.js

```

var auctioneerUI = null;

class AuctioneerUI extends UserInterface {
        // AuctionHouse events

        // adding an address to the sellers' list
        newSellerSubscribed(sellerAddress) {
                $("#subscribedSellersList").append("<li class='list-group-item'>" + sellerAddress + "</
                    li>");
                $.ajax({
                        type: "POST",
                        url: "auctionhouse/" + auctioneer.auctionHouse.contract.address  + "/seller",
                        dataType: 'json',
                        contentType: 'application/json',
                        data: JSON.stringify({ "sellerAddress": sellerAddress })
                })
        }

        // adding an address to the bidder's list
        newBidderSubscribed(bidderAddress) {
                $("#subscribedBiddersList").append("<li class='list-group-item'>" + bidderAddress + "</
                    li>");
                $.ajax({
                        type: "POST",
                        url: "auctionhouse/" + auctioneer.auctionHouse.contract.address  + "/bidder",
                        dataType: 'json',
                        contentType: 'application/json',
                        data: JSON.stringify({ "bidderAddress": bidderAddress })
                })
        }

        // displaying the card to deploy a new auction
        newAuctionSubmitted(sellerAddress, objectDescription) {
                $("#auctionCard").show();
                $("#newAuctionSubmitted").text("New");
                $("#newAuctionSubmitted").show();

                $("#currentAuctionHeader").text("A new Auction has been submitted!");
                $("#currentAuctionObjectDescription").text(objectDescription);
                $("#_sellerAddress").val(sellerAddress);
        }

        auctionDeployedSuccessfully(auctionAddress, auctionType, objectDesciption) {
                console.log("new auction create " + auctionType + " description " + objectDesciption);

                changeViewBasedOn(auctionType);

                hideSpinnerNextTo("#deployContract");
                $("#newAuctionSubmitted").text("Success");
                $("#newAuctionSubmitted").show();

```

```
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  });
```