**Installation process:**

The Auction server is a separate Maven project with its own pom.xml file. It uses springboot framework. To run the server:

* Extract the zip file. Go to eclipse -> Import -> Import Existing Maven project.
* Root Directory: **[your-installation-directory**]\CartCrafters\Auction\AuctionModule
* Do a Maven Update
* Go to the package com.example.auctionserver.controller -> CraftCraftersApplication.java. Right Click –> Run as -> Java Application. The spring application should start in port 8080
* The web browser UI now has the basic implementation. Right now, both the forward auction and the dutch auction webpage only show one pre-selected item being auctioned. To test the forward auction implementation using web-browser:  
  - Forward Auction: <http://localhost:8080/forward-auction>

-Dutch Auction: http://localhost:8080/dutch-auction

**SQL Database**

The path to the database is specified in the application.properties file in:

**[your-installation-directory**]\CartCrafters\Auction\AuctionModule\src\main\resources

We have already put the auction.db database in the root directory of the Auction server (in the same path that Auction servers pom.xml is located)

Troubleshooting:

If for some reason, the auction.db cannot be located, then we have provided the SQL scripts to create a auction table in the auction.db and populate with about 20 rows. Once you have created the db, you can put it in the same directory as the pom.xml of the auction server and double check with the specified path in the application.properties file mentioned above.

**Testing with Postman:**

The following postman test cases were run. The error messages and exceptions are described below. The exceptions will be further handled during the front-end development phase during the next deliverable.

**Dutch Bidding**

* GET <http://localhost:8080/auction/dutch/get-all>

Gives a list of all Dutch auctions, open and closed

* GET <http://localhost:8080/auction/dutch/get-all-open>

Gives a list of open Dutch auctions

* GET <http://localhost:8080/auction/dutch/details?auctionId=1>

Replace the auctionId with any auction ID and it will return a JSON with all the details of the auction that matches the auction ID. Sample JSON for this one:  
  
{

    "auctionId": 1,

    "itemId": 101,

    "auctionType": "dutch",

    "initialPrice": 100.0,

    "currentPrice": 60.0,

    "startTimeOfAuction": "2023-11-08T17:00:00.000+00:00",

    "endTimeOfAuction": "2023-11-10T17:00:00.000+00:00",

    "auctionEnded": **true**,

    "soldToUserId": 0

}

* GET <http://localhost:8080/auction/dutch/user/winner?auctionId=1&userId=456>

Returns true or false depending on whether the auction with auctionId is won by userId passed in

* GET <http://localhost:8080/auction/dutch/user/winner?auctionId=1&userId=456>

Response: {

    "success": **false**,

    "message": "Auction has not ended yet."

}

* GET <http://localhost:8080/auction/dutch/user/winner?auctionId=198&userId=456>

Response: {

    "success": **false**,

    "message": "Auction not found with ID: 198"

}

* GET <http://localhost:8080/auction/dutch/user/winner?auctionId=3&userId=123>

Response:

{

    "success": **true**,

    "message": "User is the winner!"

}

* GET <http://localhost:8080/auction/dutch/user/winner?auctionId=3&userId=12>

Response:

{

    "success": **false**,

    "message": "User is not the winner."

}

* POST <http://localhost:8080/auction/dutch/decrement?auctionId=3>

This endpoint decrements the current price with the fixed decrement value for the Dutch auction, as required by the project specification. Everytime, the seller manally hits the endpoint, the price is decremented. However, we also have a DutchAuctionWatch that schedules a monitoring task that lowers the price after a specific time value. The conditionsand exceptions of decrement are handled in dutchAuctionUpdate class’s decrementPrice(auctionId) method.

* POST <http://localhost:8080/auction/dutch/buy>

{

"auctionId": 3,

"userId": 123

}

A POST request with the body that contains the auctionId and userId closes the Dutch bid and makes the userID the winner. The return JSON object of the request looks similar to this:

{

    "auctionId": 3,

    "itemId": 103,

    "auctionType": "dutch",

    "initialPrice": 200.0,

    "currentPrice": 200.0,

    "startTimeOfAuction": "2023-11-10T15:00:00.000+00:00",

    "endTimeOfAuction": "2023-11-13T12:46:22.779+00:00",

    "auctionEnded": **true**,

    "soldToUserId": 123,

    "minimumPrice": 0.0,

    "decrement": 0

}

* POST <http://localhost:8080/auction/dutch/buy>

{

"auctionId": 39,

"userId": 123

}

Response: Auction not found with ID: 39

* POST <http://localhost:8080/auction/dutch/buy>

{

"auctionId": 3,

"userId": 123

}

Response: Auction has already ended.

**Forward Bidding**

* **GET** <http://localhost:8080/auction/forward/get-all>
* **GET** <http://localhost:8080/auction/forward/get-all-open>
* **GET** <http://localhost:8080/auction/forward/details?auctionId=9>
* **GET** <http://localhost:8080/auction/forward/details?auctionId=99>
* POST <http://localhost:8080/auction/forward/bid>
* POST http://localhost:8080/auction/forward/close?auctionId=1