



# UNIVERSITÀ DI PISA

## Computer Engineering

### Distributed Systems and Middleware Technologies

# DISTRIBUTED AUCTION SYSTEM

Online auction application using an application server  
and the Erlang language

---

*STUDENTS:*

Lorenzo Cima  
Nicola Ferrante  
Simone Pampaloni  
Niccolò Scatena

Revision: (None) – (None)

Academic Year 2020/2021

## **Abstract**

This project has been developed as a workgroup during the classes of Distributed Systems and Middleware Technologies (Computer Engineering, University of Pisa). The aim of this workgroup is to give a demonstration of the students' capabilities to handle the development of a simple web application that exploits JEE and the peculiarities of the Erlang programming language.

# Contents

<b>1</b>	<b>Specifications</b>	<b>2</b>
<b>2</b>	<b>Actors and requirements</b>	<b>3</b>
2.1	Functional requirements . . . . .	3
2.2	Non-functional requirements . . . . .	3
<b>3</b>	<b>Use cases</b>	<b>5</b>
<b>4</b>	<b>Data Model</b>	<b>7</b>
<b>5</b>	<b>Architecture</b>	<b>9</b>
5.1	Web client . . . . .	9
5.2	Application server . . . . .	9
5.2.1	JEE beans . . . . .	9
5.2.2	JMS and message driven beans . . . . .	10
5.2.3	Websocket . . . . .	11
5.3	Erlang subsystem . . . . .	11
5.3.1	Dispatchers . . . . .	11
5.3.2	Executor Clusters . . . . .	12
5.3.3	Data persistance with Mnesia . . . . .	13
5.3.4	Fault tolerance techinques . . . . .	14
<b>A</b>	<b>User manual</b>	<b>16</b>
A.1	Introduction . . . . .	16
A.2	Login and Registration . . . . .	16
A.2.1	Registration . . . . .	16
A.3	Application Interface . . . . .	17
A.3.1	User Section . . . . .	17
A.3.2	Auctions Section . . . . .	17
A.3.3	Sell Section . . . . .	18

# 1 | Specifications

The application developed for this workgroup task is a tool for **online auctions**.

The main purpose of the application is to allow people to put items up for sale through auctions and allow people to try to buy items by bidding in available auctions.

An **auction** sells a number of items with a fixed minimum price for each item. Each **bid** defines the number of items you want to buy, and the price offered for each item. The auction system, for each bid made or deleted, will have to calculate the list of winning bids that maximize profit, considering as parameters the number of items for sale and the timestamp of the bids. The auction system of this application is set to “all or nothing” semantics, this means that winning bids will get the required quantity of items. If there are not enough items, the bid will not be placed among the winning ones.

Before login, a user must register to the application specifying its username and password. When logged in, the user can see the list of the available auctions. A user can select an auction from the list and view all the information of that auction and, if he wishes, can place a bid. Once a bid has been made, the user can follow the progress of the auction in real time in this page (if his bid is among the winning ones) and the auction will be added to the list of followed auctions which you can view later in a special section. Also, if you are no longer interested, the user can delete the bid.

A user can see the list of auctions created in a special section and can select a specific auction to see its progress in real time or its result.

## 2 | Actors and requirements

Main actors:

**Agent** The one who want to sell an object and open an auction for it.

**Bidder** the one who makes bids for an auction hoping to win it and buy the item.

Users can interface with the application in both ways dynamically even within the same session.

### 2.1 Functional requirements

**Registration process** New users must register in the system by declaring username and password.

**Login process** The system shall handle login process with a username and password, a user identifies himself/herself within the system, so the system is able to manage all the data concerning him/her.

**User Views** The system shall provide appropriate viewers for the users to see list of available auctions, list of auctions they have created and list of auctions in which they have a bid.

**Add/Delete Bid process** A user should be able to add or delete a bid on an auction in progress.

**Auctions handler** An agent should be able to add all the details of a new auction he/she wants to create and should be able to delete it.

**Real-time Auction progressing** the system must be able to calculate the list of winning bids for each auction as bids are added and deleted and must show users the progress of the auction.

### 2.2 Non-functional requirements

**Usability** The application must be easy to use, with a simple and intuitive graphical user interface.

**Portability** The application must be portable, so that any user can use it, independently from the system that he/she uses.

**Data Persistence** The application must achieve data persistence.

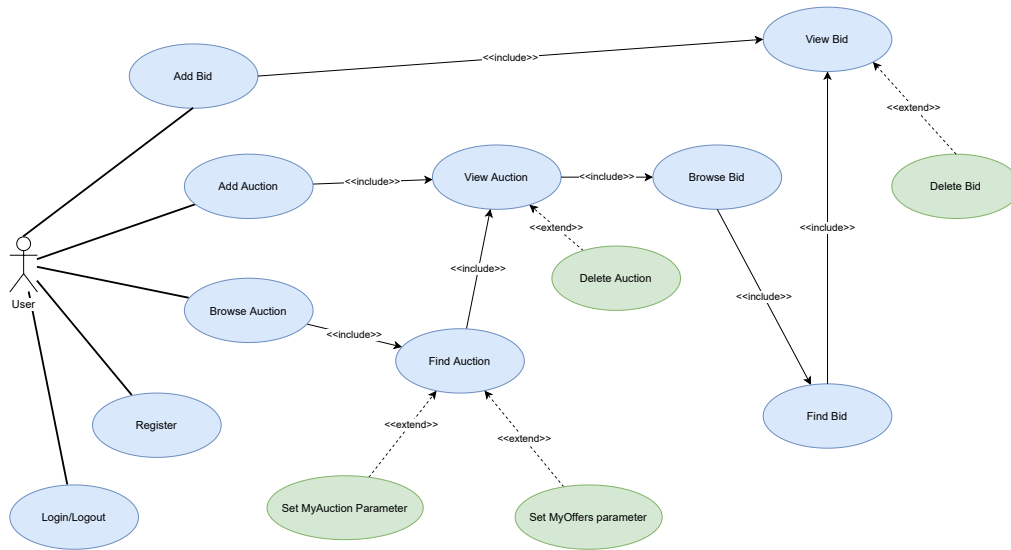
**Fault Tolerance** the application must implement techniques to manage the occurrence of failures.

**Availability** the application must be available in any time.

**Concurrency** The application must handle multiple users at the same time.

### 3 | Use cases

Figure 3.1 shows the use cases UML diagram.



**Figure 3.1:** Use cases UML diagram

The following use cases are defined (all use cases, except Login and Register, requires the user to be logged in):

**Login/Logout** A user can login in the application using its credentials (username, password). When logged in, he can logout from the application at any time.

**Register** The first time a user uses the application he/she will be asked to register himself/herself, providing an username and a password.

**Browse Auction** See the list of all available auctions saved in the application.

**Find Auction** Search auctions selecting the auctions you have created (**MyAuctions** parameter) or selecting the auctions in which you have made a bid (**MyOffers** parameter).

**View Auction** See all the information about an auction.

**Add Auction** Add a new auction, providing name, image of the item to sell, description, the date of the end of the auction, the starting price for the auction, the minimum raise and the quantity of items that the user wants to sell.

**Delete Auction** A user can delete an auction if it owns it.

**Browse Bid** after an auction selection, a user can see all the bids he/she made for this auction.

**Find Bid** Search a bid from the list.

**View Bid** See all the information about a bid.

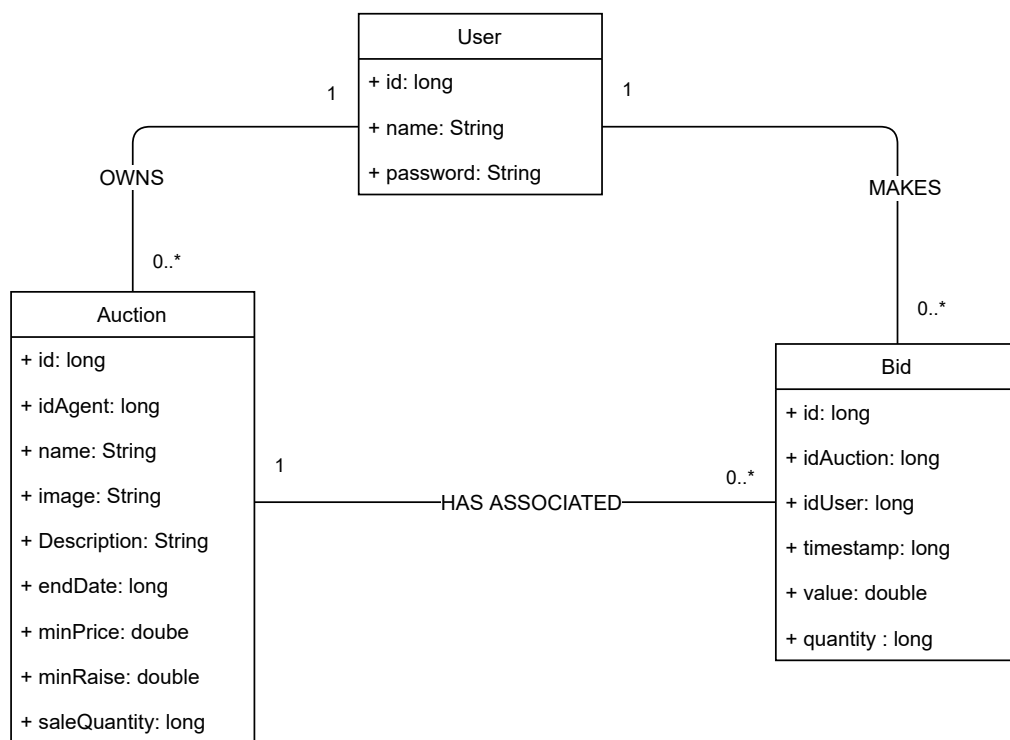
**Add Bid** Add a new bid for an auction, providing the value of the bid and the quantity of items for which the bid is made.

**Delete Bid** an user can remove a bid he/she made before.



## 4 | Data Model

Figure 4.1 shows the analysis classes of the data model used by the application.



**Figure 4.1:** Data model UML diagram (analysis classes)

**User** A registered user:

**id** Unique id of a user.

**name** The username of a user.

**password** The password of a user.

**Auction** An auction of the system:

**id** unique id of an auction.

**idAgent** id of the user that make the auction.

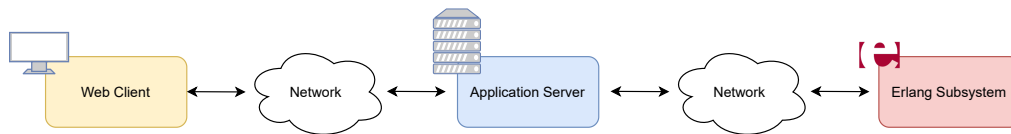
**name** name of the item of the auction.  
**image** image of the items of the auction.  
**description** description of the items of the auction.  
**endDate** the date and time when the auctions ends.  
**minPrice** starting price for each item of the bid.  
**minRaise** minimum raise for each new bid.  
**saleQuantity** number of items to sell.

**Bid** A bid for an auction made by a user:

**id** unique id of a bid.  
**idAuction** id of the auction.  
**idUser** id of the user that made the bid.  
**timestamp** timestamp the bid was made.  
**bidValue** value of the bid for single item.  
**quantity** number of items you want to buy.

## 5 | Architecture

Figure 5.1 shows the application's architecture.



**Figure 5.1:** Application architecture

### 5.1 Web client

The web client allows users to interact with the application. It just reads the input from the user from which builds http requests for the application server and send them out. When the server responds, the web client shows the results to the user and waits for new input. The web interface provides an intuitive and easy to use graphical interface where the user can scroll through lists, select items and fill forms.

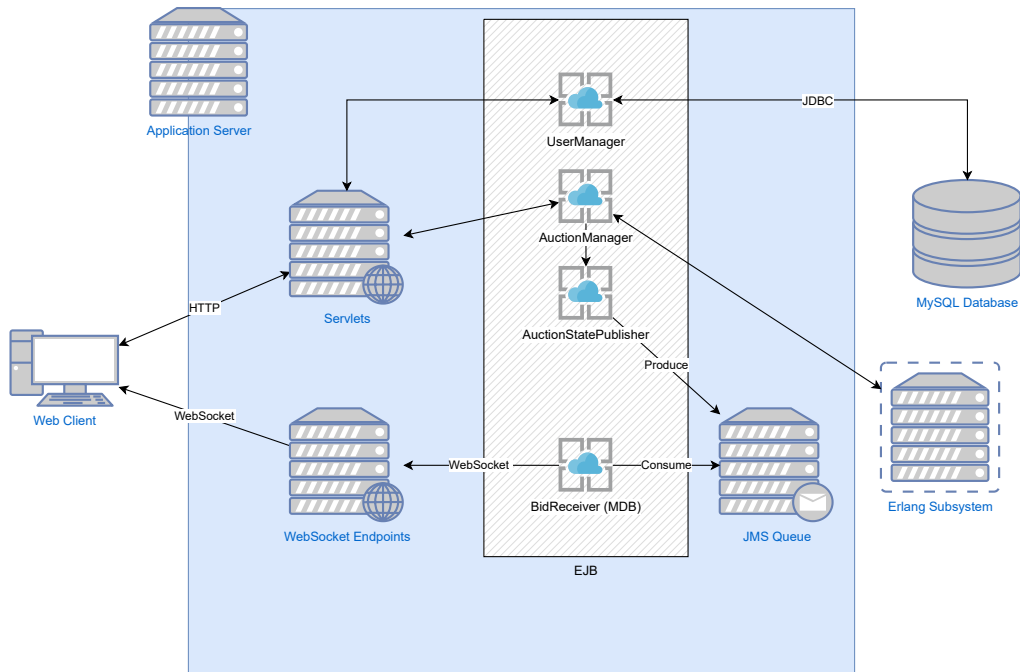
### 5.2 Application server

The application server provides the management of the user interface for the web client thanks to the use of servlets that exploit the JSP technology, and it manages the business logic using the JakartaEE framework that allows it to communicate with the Erlang subsystem. Furthermore, given the purpose of the application, the application server implements an asynchronous communication system with the web clients, which allows users to receive real time updates on the status of the auctions, using the Websocket communication protocol.

Figure 5.2 shows the application server's architecture.

#### 5.2.1 JEE beans

JEE beans provide business logic interfaces implementation. They implement the interface between MySQL database (for user data) and Erlang subsystem towards servlets. It is composed of the following components:



**Figure 5.2:** Application Server architecture.

**AuctionManager** A singleton session bean that manages the communication with the Erlang subsystem, exposing the interfaces between the former and servlets. This bean is also in charge of communicating changes in the auctions' state from the Erlang subsystem to the **AuctionStatePublisher**.

**UserManager** A stateless session bean that manages the communication with the MySQL database (that manages user informations), exposing the interfaces between the former and servlets.

**AuctionStatePublisher** A singleton session bean that act as a proxy for the actual state of all the bids in the Erlang subsystem, whenever the state changes (because a bid has been made or deleted, or an auction is closed) this bean is informed from the **AuctionManager**. Furthermore, it acts as a producer for a JMS queue, to promptly notify clients on changes in the auction

### 5.2.2 JMS and message driven beans

JMS and message driven beans provide the messaging framework for bids dispatching from Erlang subsystem for notifying changes in auctions' state to clients. It is composed of the following components:

**JMS queue** a queue is used in order to asynchronously notify clients of changes in the auction state.

**BidReceiver** Is a message-driven bean that receives the new state of the auctions from the queue and using Websocket ClientEndpoint API initiates the communication towards a Websocket ServerEndpoint. Then the ServerEndpoint communicate to the clients the updates.

### 5.2.3 Websocket

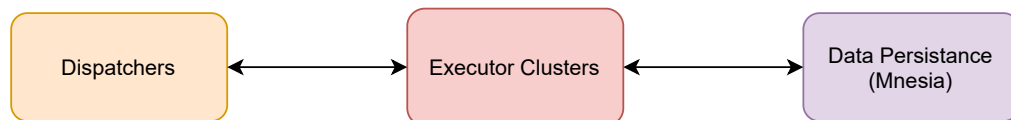
It be used as communication protocol among JEE beans and web clients. The web server acts as a relay, providing two endpoints: one from the application server towards the web server, and the other from the web server towards the clients.

This protocol allows us to implement a real time notification broadcasting service for all the clients interested in an auction in a clean way (no polling is needed), giving us the possibility to initiate the communication from the backend.

## 5.3 Erlang subsystem

The Erlang subsystem manages the persistence of auctions and bids data and is responsible for the calculation is responsible for calculating the auction status. The Erlang subsystem is designed and implemented with techniques that allow availability and fault tolerance. The Erlang subsystem takes advantage of Erlang language and Erlang's runtime system and its concurrent computing capabilities.

Figure 5.3 shows the Erlang subsystem's logical architecture.



**Figure 5.3:** Erlang subsystem logical architecture

### 5.3.1 Dispatchers

Dispatchers are a set of Erlang processes which have the task of interfacing with the application server. The task of a dispatcher is to assign the management of an auction to a cluster executor when an auction creation request arrives from the application server and subsequently forward the requests for that auction to the same executor cluster. The assignment of the auction to the cluster is done through a calculation between the id of the auction and the id of the cluster, for this reason every dispatcher knows where to forward the requests without the need for any kind of communication between them. Whenever the application server has a request for the Erlang subsystem, it sends it to a dispatcher which will

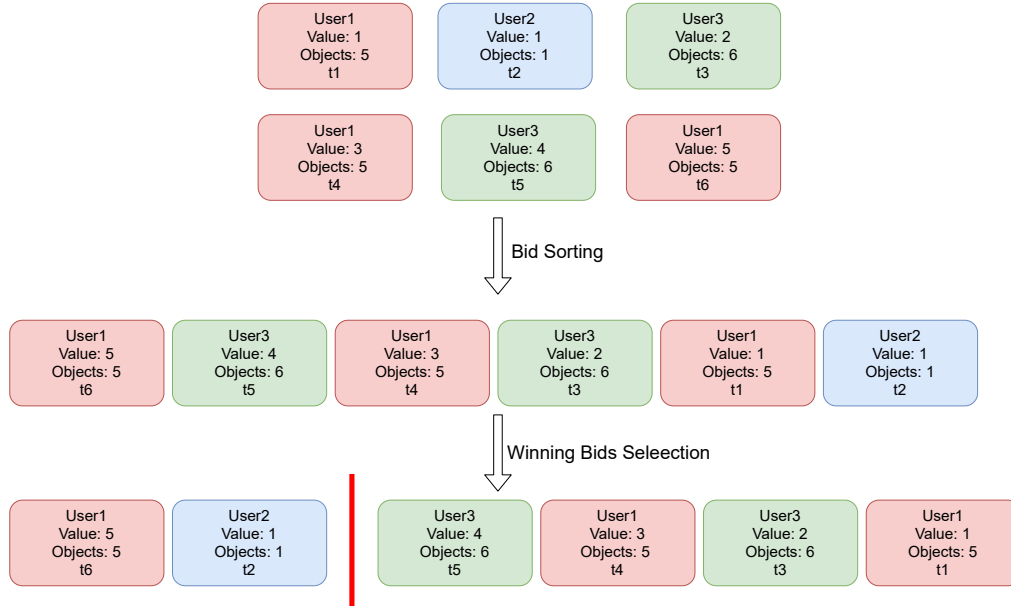
forward the request to the cluster executor managing that auction. Once the response from the cluster executor is received, the dispatcher will forward the response to the application server. Dispatchers are implemented through the `gen_server` behavior.

### 5.3.2 Executor Clusters

The executor clusters are a set of cluster and each cluster is composed of a set of Erlang processes. The choice for which it was decided to implement a cluster structure will be described in Section 5.3.4 “Fault tolerance techniques”. Each executor cluster manages a subset of auctions of the system (the assignment of an auction to a cluster is decided, as previously mentioned, by the dispatchers). Each cluster communicates with Mnesia (we will see later) for the data persistence and manages the progress of the auctions for which it is responsible. In particular, for each auction, the cluster must calculate the so-called **AuctionState**, i.e. produce the list of current winning bids each time a bid is added or deleted.

Following the all-or-nothing semantics described in the specifications, the **AuctionState** is calculated, trying to maximize the profit for the Agent, by taking the bids with the highest value (for the same value, the oldest bid is considered as the best) and that the sum of the quantities of items requested by these is less than or equal to the quantity of items for sale. Furthermore, it is ensured that only one bid per user is among the winning ones.

Figure 5.4 shows an **AuctionState** computation example.



**Figure 5.4:** AuctionState Computation

The **AuctionState** is kept in memory in order to be sent upon request by the application server and is recalculated (and subsequently sent) every

time an offer is added or deleted. Executor processes are implemented through the *gen\_server* behavior.

### 5.3.3 Data persistence with Mnesia

The persistence of data (auctions and bids) is managed by Mnesia, a distributed telecommunications DBMS that allows persistence, data replication, atomic transactions, and location transparency.

For the persistence of auctions and bids, two tables of type *set* have been created on Mnesia. To define the attributes of the tables, the following records have been defined.

```
1      -record(auction,{
2          id_auction,
3          id_agent,
4          name,
5          image,
6          description,
7          end_date,
8          min_price,
9          min_raise,
10         sale_quantity
11     }).
12
13     -record(bid, {
14         id_bid,
15         id_auction,
16         id_user,
17         timestamp,
18         bid_value,
19         quantity
20     }).
```

**Listing 5.1:** Mnesia Record

Through a specific Erlang module the following methods have been defined to interact with Mnesia. All the operations inside the methods have been implemented through `mnesia:transaction` which allows to perform transactional operations on Mnesia.

**insert\_auction** insert the auction passed as value.

**insert\_bid** insert the bid passed as value.

**delete\_auction** delete the auction with the id passed as value.

**delete\_bid** delete the bid with the id passed as value.

**get\_auction** returns a specific auction.

**get\_bid\_list** returns the list of bids for an auction (and optionally, of a given user).

`get_auction_list` returns the list of auctions available.

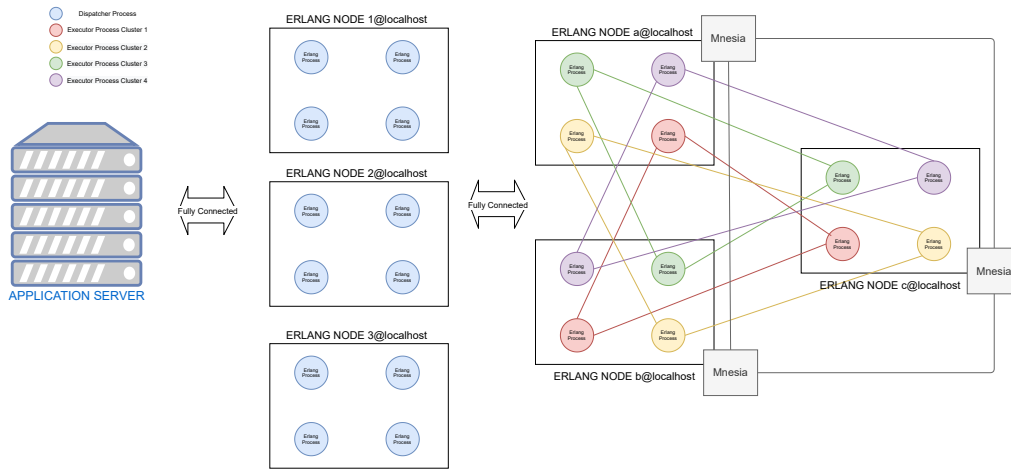
`get_bidder_auctions` returns the list of auctions in which the user has made at least 1 bid.

`get_agent_auctions` returns the list of auctions created by an agent.

### 5.3.4 Fault tolerance techniques

For each module of the Erlang subsystem some techniques have been implemented to ensure a good degree of availability and tolerance to tastes.

Figure 5.5 shows the Erlang subsystem's architecture.



**Figure 5.5:** Erlang subsystem architecture

#### Dispatchers

As can be seen from the architecture, multiple dispatcher instances are executed on different dispatcher Erlang nodes to allow the application server to always find an available dispatcher in case a dispatcher process/node fails. Furthermore, the presence of multiple dispatcher instances allows the application server to implement load-balancing techniques to balance the load of requests between the dispatchers available;

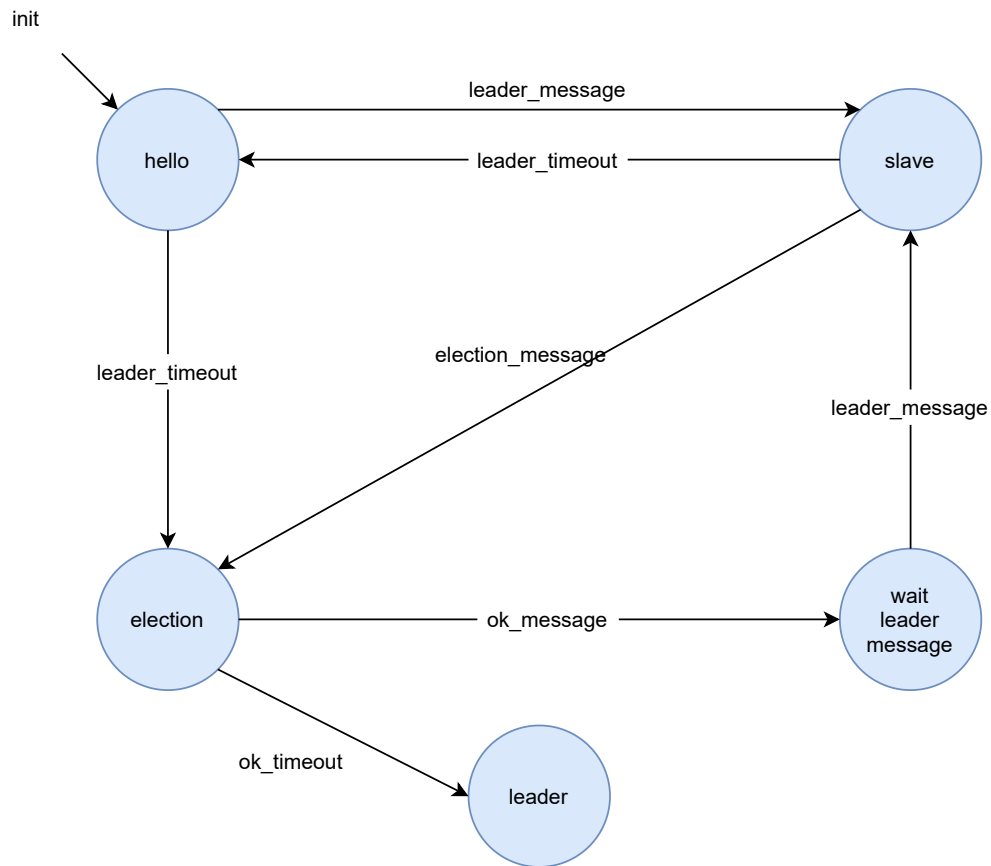
#### Executor clusters

Each executor cluster is organized according to a **leader-slaves structure**. The leader manages the requests coming from the dispatchers, communicate (if needed) to the Mnesia for the persistence of data, and calculate the current **AuctionState** of the auctions and then sends it to the slaves. The slaves have the function of replicas.

If the leader fails, the slaves will start a leader election process using the **Bully algorithm**, to allow the cluster to remain operational.



Figure 5.6 shows the Bully algorithm state machine implemented in this application.



**Figure 5.6:** Bully Algorithm State Machine

Furthermore, as is visible from the architecture, the processes of each executor cluster are executed in different executor Erlang nodes (even if it is possible that the same node executes more processes of the same cluster) so that the clusters continue their operation even in case of failure of an executor Erlang node.

# Appendix A

## User manual

### A.1 Introduction

DistributedAuctionSystem (DAS) is an application used to create online auctions and participate in them.

Every user can put items up for sale through auctions and can try to buy items by bidding in available auctions.

Any user who wants to buy items can act as a bidder and can scroll through the list of available auctions, select an auction he is interested in to see information, make bids and follow the progress of auctions in real time.

Any user who wants to sell items can act as an agent and has the ability to create new auctions and follow the progress of his current auctions.

All these features are made available to the user through a web interface that provides a simple and intuitive graphic interface.

### A.2 Login and Registration

When the application starts it opens the main page (Figure A.1), on which an old user can insert username and password to login, clicking on the button “Login”.

Otherwise, if the user is a new one, he must do the registration procedure.

#### A.2.1 Registration

If you are a new user, click on “Register” on the main page to open the registration form (Figure A.2).

Then, insert your username and your password. Insert another time your password to confirm. If you went in the registration page wrongly, click on “Login” to return to the login page.

When you filled out the form, click on “Register”. Now if you don’t receive error messages your account will be create. So, login to see all the application features.

## A.3 Application Interface

When you are logged in, You are redirected to the main page that contains the auctions currently available in the application. Click on one of it to see details and make a bid.

As you can see from the Figure A.3, at each top of the application page there is a drop-down menu to be able to access all sections of the app at any time.

### A.3.1 User Section

#### Modify Password

To modify your password you have to click on the User section on the top page menu and then to press “Modify Password”. In the page shown in Figure A.4 insert your old password and then insert twice the new password that you desired. Then, click on the button “Modify”. You will see a message that says if the operation is done correctly or not. If no error occurs, you can return to the home page clicking on the button “Home”.

#### Logout

To logout you have to click on the User section on the top page menu and then to press “Logout”. Now, if all is done correctly, you will see again the Login page.

### A.3.2 Auctions Section

#### MyAuctions

The page “MyAuctions” has the same structure of Figure A.3 but contains only objects that you have sold or that you want to sell.

#### MyOffers

Also the page “MyOffers” has the same structure of Figure A.3 and contains all the objects where you have done a bid, both with finished or in progress auctions.

#### Select Auction

In the pages “Auction List”, “MyAuctions” and “MyOffers” you can click on an auction to see its details. There are two possible detail pages:

**Agent detail page** If you are the owner of the selected object, you will see on the left your auction information, the momentary total gain and objects sold (they will become definitive at the end of auction

time). On the right you can see the list of winning offers. All is shown in Figure A.5. To delete the auction, you can press the button “Delete Auction”.

**Bidder detail page** If you are a customer you will see, as in Figure A.6, the general auction information and the minimum bid to momentary win the quantity of objects you prefer. On the right side you can bid, inserting the offer and the desired number of objects. Then, you have to press the button “Send Bid”. If all is done correctly, you will see if your offer is momentary a winning one or not, as shown in Figure A.7. The bid status is updated real time. If you see false in the column “is winning”, this means that to win you have to do another bid. You can delete your bid pressing the button “Delete”, on the row with the offer to cancel. At the end of the time, open the detail page, you will see if succeeded in buy the object or not, as shown in Figure A.8.

### A.3.3 Sell Section

To create a new auction you have to click on the Sell section on the top page menu. You are redirected to a page where you have to fill out a form, writing the name of the auction, the minimum bid and the minimum raise you accept for that auction, the number of items you want to sell for this auction, the date and time you want the auction to end and a description of the items. It is also necessary to upload an image of the items for sale. Then, click on the button “Sell”. If no errors are shown, you have created a new auction.

## Login



Username

Password

Login

Register

**Figure A.1:** Login interface

## Register

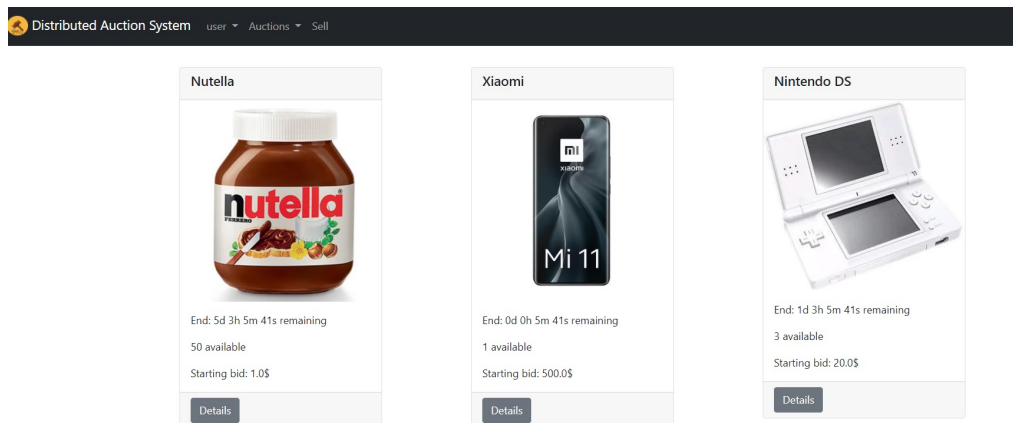


Username

Password

Confirm Password

**Figure A.2:** Registration of a user



**Figure A.3:** Application main page

## Modify Password



Old Password

Password


Confirm Password

[Modify](#)[Home](#)

**Figure A.4:** Modify Password page



### Nintendo DS



**Description:**  
Nintendo DS lite, grey  
Vendor ID: 1  
Starting Price: 20.0\$  
Minimum Raise: 1.0\$  
Available Objects: 3  
End time: 0d 21h 23m 34s remaining  
Total Gain: 75\$  
Sold: 3/ 3

### Control Panel


Delete Auction

#### Winning Bids

Quantity	Offer
3	25

**Figure A.5:** Details of an object from the point of view of its seller

### Nintendo DS



**Description:**  
 Nintendo DS lite, grey  
 Vendor ID: 1  
 Starting Price: 20.0\$  
 Minimum Raise: 1.0\$  
 Lowest Bids:

Quantity	Offer
less or equal to 1	20
between 1 and 3	21

Available Objects: 3  
 End time: 0d 21h 25m 37s remaining

### Control Panel


Offer

N.Objects

**Your Bids**  
 No offers

**Figure A.6:** Details of an object from the point of view of a customer, without offers done

### Nintendo DS



**Description:**  
 Nintendo DS lite, grey  
 Vendor ID: 1  
 Starting Price: 20.0\$  
 Minimum Raise: 1.0\$  
 Lowest Bids:

Quantity	Offer
less or equal to 3	26

Available Objects: 3  
 End time: 0d 21h 24m 17s remaining

### Control Panel

Offer

N.Objects

Send Bid

**Your Bids**


Quantity	Offer	Is Winning	
1	21.0	false	Delete

**Figure A.7:** Details of an object from the point of view of a customer, with offers done

Distributed Auction System
lorenzo ▾
Auctions ▾
Sell

You win!

### Nintendo DS




End:  
 1 available  
 Starting bid: 21.0\$

Quantity	Price
1	21.0

localhost:8080/web/auktion

**Figure A.8:** Page for the end of an auction on customer side

 Distributed Auction System

user ▾ Auctions ▾ Sell

### Sell something!

Name:  
Object

Minimum Bid:  
1

Minimum Raise:  
0.1

Number of Objects:  
10

End Day:  
05/06/2021

End Hour:  
21:00

Description:  
This is a beautiful object

Image:  
Scegli file Nessun file selezionato

Sell

**Figure A.9:** Creation of a new auction