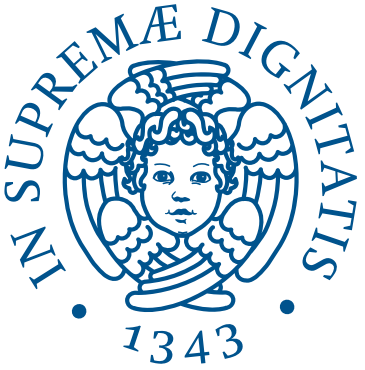

UNIVERSITÀ DI PISA

Computer Engineering

Distributed Systems and Middleware Technologies

Project Documentation

TEAM MEMBERS:
Tommaso Burlon
Francesco Iemma
Olgerti Xhanej

Academic Year: 2021/2022

Contents

1 — Project Specifications

AuctionHandler is a distributed web-app in which users can sell their goods by creating Online Auctions. Registered users have the possibility to join an ongoing auction in order to buy a good in case they beat the concurrence by setting an higher offer on a given limited time.

1.1 Use Cases

An *Unregistered User* can:

- Register to the service

A *Unlogged User* can:

- Login to the service

A *Logged User* can:

- View the list of ongoing Auctions
- Create a new Auction
- Join an ongoing Auction
- Logout
- After Joining an Auction:
  - Make an offer
  - View list of participants
  - View past offer history
  - View the remaining time
  - Wait until the end of the Auction and then exit
  - View Auction result

The *System* must:

- Remember registered users
- Remember ongoing auctions
- Remember auction participants
- Choose in a unique way the auction winner
- Remember offers history
- Synchronize the remaining time, the auction participants, the offer history for an auction for each user
- Synchronize the list of ongoing auctions for each user

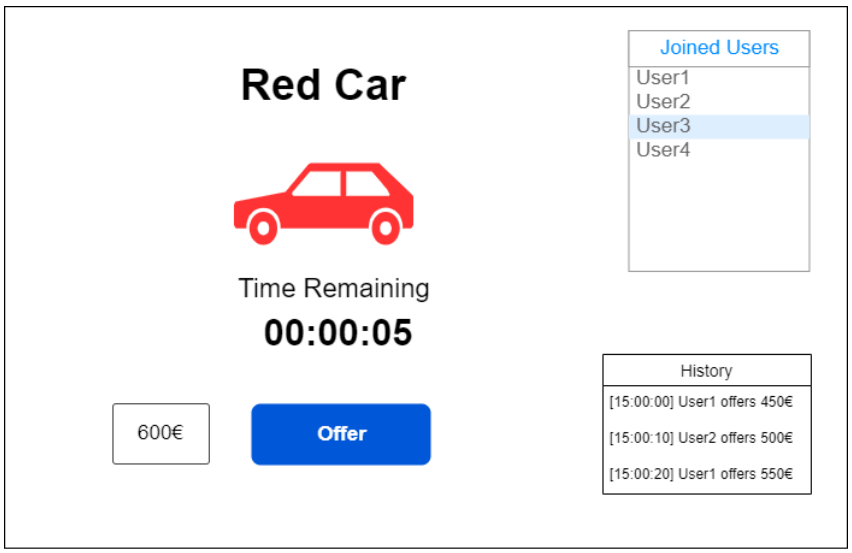

Figure 1: Mock-up of the main interface of the auction

1.2 Synchronization and Communication Issues

On the application we have the following synchronization and communication issues:

- Client nodes need to be synchronized with the same remaining time of the auction, the same offer history for a given auction, the same list of joined users on a given auction and the same list of available ongoing auctions.
- In case a client makes a valid offer, the server will be in charge of communicating to other clients nodes the information regarding the made offer.
- In case a client creates a new auction, the server will be in charge of communicating to other clients the information regarding the newly created auction.

1.3 Design Ideas

We were thinking of implementing the system in the following way:

- **Client Nodes**: User Interface via HTML/CSS, generated via Java Servlets and JSP. Each client node will have a dedicated Erlang node for communicating with the server
- **Server**: Made entirely in Erlang. The responsibility of the server will be performing persistent data storage and handling the communication and synchronization between different client nodes

2 — System Architecture

A graphical representation of the system architecture can be seen in figure 2.

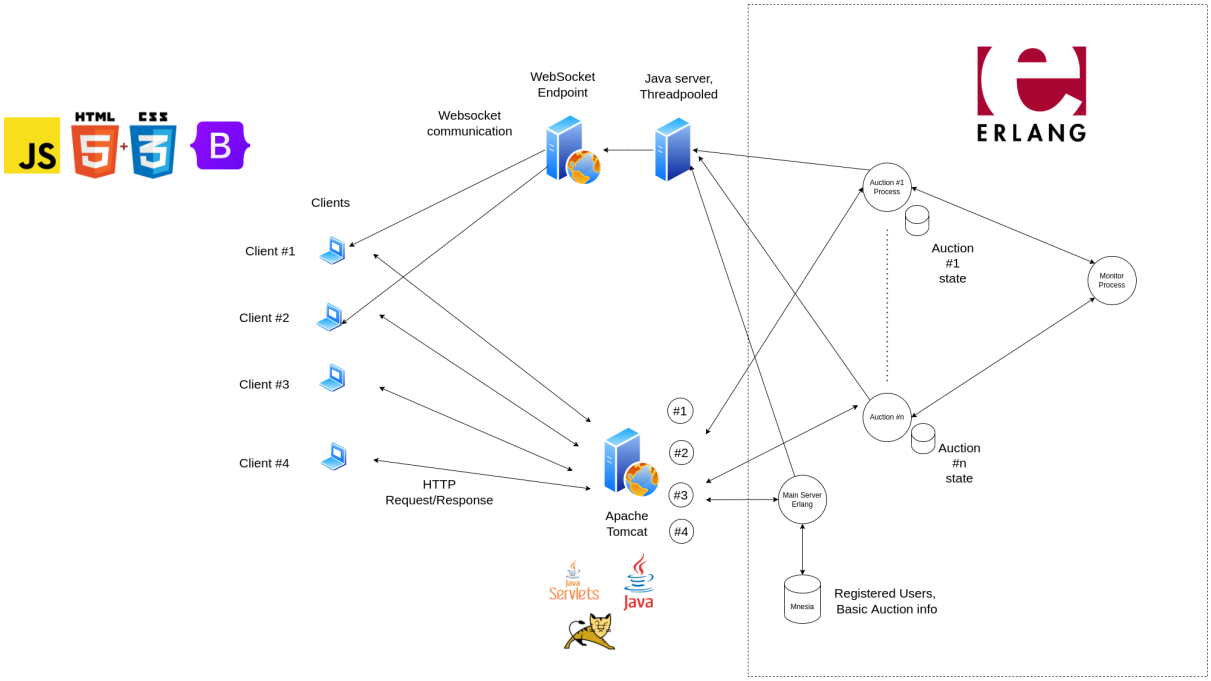

Figure 2: System Architecture graphical representation

We can divide the overall system in two part:

- The server side part: developed in Erlang, it is in charge of handling the request coming from the users and it has to handle the auctions and to maintain the global view of an auctions consistent among the users.
- The client side part: depeloved in Java, using EJB e JSP, and in Javascript for what regards the websocket. It is in charge of retrieving information from the server, create the GUI, update the GUI in order to let the user have a consistent view of the state of the auction and of the overall system.

2.1 Server Side

2.1.1 Main Server

2.1.2 Auction Handler

2.1.3 Monitor And Supervisor

2.2 MnesiaDB

2.3 Client Side

2.4 Synchronizations Issues