POLITECNICO
DI MILANO

Project

Tecnologie Informatiche per il Web

a.a. 2024-2025

Prof. Fraternali Piero

Cristian Summa **Mathias Rodigari**

Sommario

Specification

Pure HTML version

A web application allows you to manage online auctions.

Users log via login and can sell and buy articles sold in auctions.

Each user has a username, password, first name, last name and address (used to ship the bought articles).

Each article has a code, name, description, image and price.

The HOME page contains two links, one to access the SELL page and one to access the BUY page.

The SELL page shows a list of auctions created by the user and not closed yet, a list of auctions created by him and closed and two forms, one to create a new article and one to create a new auction to sell the user's articles.
The first form allows to insert an article into the database with all its data, which are all mandatory.
The second form shows a list of the articles in the database and available to be sold and allows to select more than one to create an auction that contains them.

An auction is made of one or more articles being sold, the initial price of the set of articles, the minimum increment of each offer (as an integer number of euros) and an expiration date (date and time, ex. 19-04-2021 at 24:00).
The auction's initial price is obtained as the sum of the comprised articles' prices in the offer.
The same article cannot be included in different auctions.
Once sold, an article cannot be available to be inserted into other auctions.

The list of auctions in the SELL page is ordered by date+time in ascending order.
The list shows: code and name of the articles inside the auction, maximum offer, remaining time (number of days and hours) between the moment (date and time) of the login and the auction's closing date and time.

Clicking on an auction inside the list an AUCTION DETAILS page appears, which shows for an open auction all the auction's data and the list of offers (username, offered price, date and time of the offer) ordered by date+time descending.

A CLOSE button allows the user to close the auction if the expiration date is passed (ignore the case of expired auctions that have not been closed by the user and don't implement an automatic closing of auctions after the expiration date).
If the auction is closed, the page shows all the auction's data, the buyer's name, the final price and the (fixed) shipping address of the user.

The BUY page contains a form to search for keyword.
When the buyer sends a keyword the BUY page is refreshed and shows a list of open auctions (whose expiration date is after the date and time of submission) for which the keyword appears inside the name or description of at least one of the auction's articles.
The list is ordered in a descending order based on the time (number of days and hours) remaining until closure.

Clicking on an open auction an OFFER page appears, showing all the articles' data, the list of the arrived offers in descending date+time order and an input field to send their own offer, which must be higher than the current max offer by an amount equal at least to the minimum increment.
After the submission the OFFER page shows the updated offers list.

The BUY page also contains a list of the offers won by the user with the article's data and the final price.

JavaScript version

Build a client server web application that extends and/or modifies the previous specifications as follows:

- After login, the entire application is built in a single page
- If the user logs in for the first time the application shows the content of the BUY page. If the user has already used the application, it shows the content of the SELL page if the last action performed by the user was creating an auction; otherwise, it shows the content of the BUY page with the list (even if empty) of the auction that the user had previously clicked and that are still open.
  The information about the last performed action and of the visited auctions is saved on the client-side for a month.
- Every user interaction is handled without completely reloading the page; instead produces the asynchronous invocation of the server and the modification only of the content to be updated after the event.

Specifications analysis

Data analysis

Legend:

- Entity
- Attribute
- Relation

A web application allows you to manage online auctions.

Users log via login and can sell and buy articles sold in auctions.

Each user has a username, password, first name, last name and address (used to ship the bought articles).

Each article has a code, name, description, image and price.

The HOME page contains two links, one to access the SELL page and one to access the BUY page.

The SELL page shows a list of auctions created by the user and not closed yet, a list of auctions created by him and closed and two forms, one to create a new article and one to create a new auction to sell the user's articles.
The first form allows to insert an article into the database with all its data, which are all mandatory.
The second form shows a list of the articles in the database and available to be sold and allows to select more than one to create an auction that contains them.

An auction is made of one or more articles being sold, the initial price of the set of articles, the minimum increment of each offer (as an integer number of euros) and an expiration date (date and time, ex. 19-04-2021 at 24:00).
The auction's initial price is obtained as the sum of the comprised articles' prices in the offer.
The same article cannot be included in different auctions.
Once sold, an article cannot be available to be inserted into other auctions.

The list of auctions in the SELL page is ordered by date+time in ascending order.
The list shows: code and name of the articles inside the auction, maximum offer, remaining time (number of days and hours) between the moment (date and time) of the login and the auction's closing date and time.

Clicking on an auction inside the list an AUCTION DETAILS page appears, which shows for an open auction all the auction's data and the list of offers (username, offered price, date and time of the offer) ordered by date+time descending.

A CLOSE button allows the user to close the auction if the expiration date is passed (ignore the case of expired auctions that have not been closed by the user and don't implement an automatic closing of auctions after the expiration date).
If the auction is closed, the page shows all the auction's data, the buyer's name, the final price and the (fixed) shipping address of the user.

The BUY page contains a form to search for keyword.
When the buyer sends a keyword the BUY page is refreshed and shows a list of open auctions (whose expiration date is after the date and time of submission) for which the keyword appears inside the name or description of at least one of the auction's articles.
The list is ordered in a descending order based on the time (number of days and hours) remaining until closure.

Clicking on an open auction an OFFER page appears, showing all the articles' data, the list of the arrived offers in descending date+time order and an input field to send their own offer, which must be higher than the current max offer by an amount equal at least to the minimum increment.
After the submission the OFFER page shows the updated offers list.

The BUY page also contains a list of the offers won by the user with the article's data and the final price.

Note: During the design phase, the *Image* attribute of the *Article* entity was refactored into a separate entity, establishing an explicit relationship between the two.

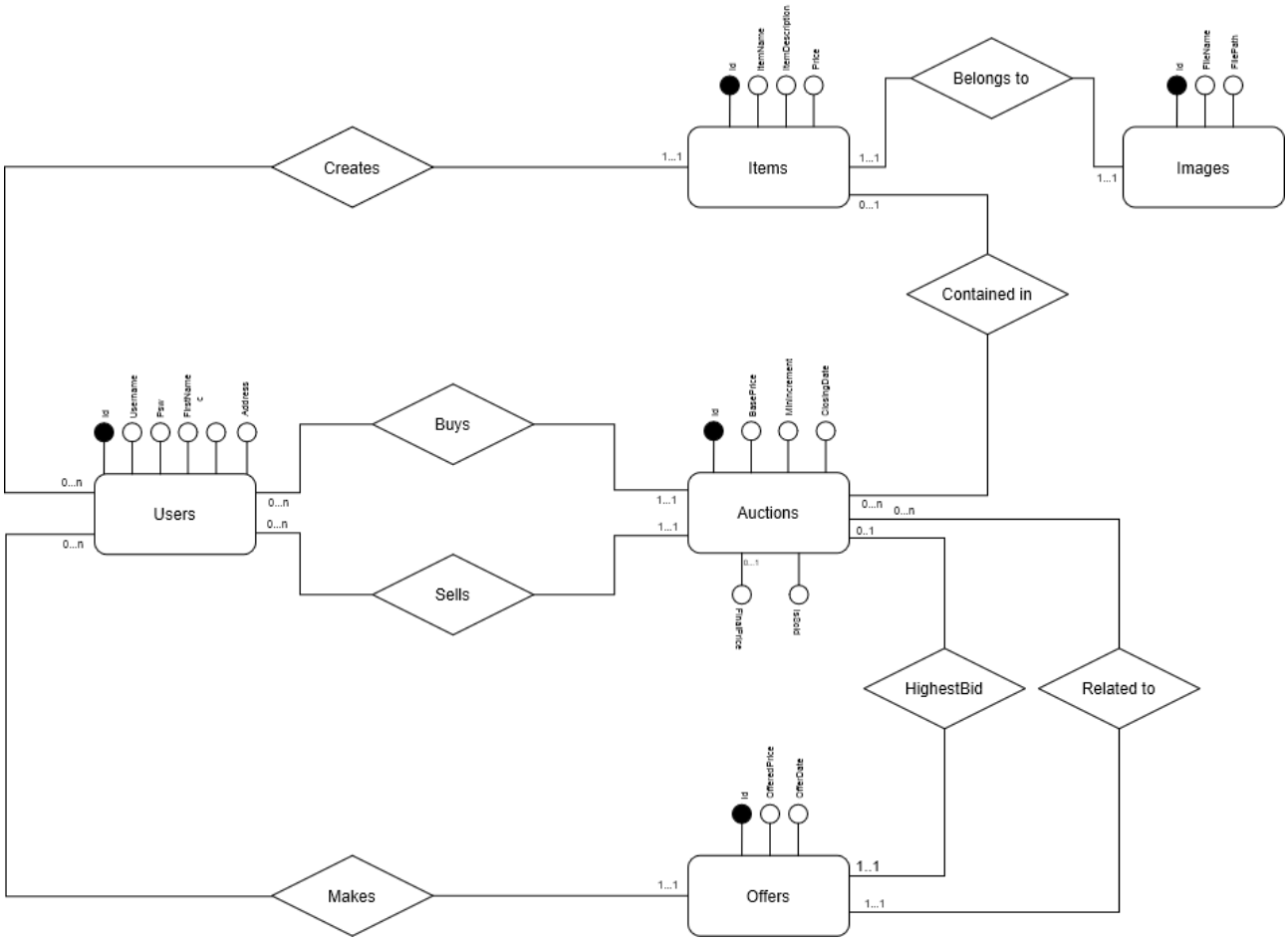
This decision was motivated by the need to prevent potential issues in handling image storage and retrieval on the server side.

In particular, the separation allowed for a clear distinction between two relationships: one linking each *Article* to its associated Image, and another connecting the Image entry in the database to the corresponding physical file in the filesystem.

This structure simplifies future modifications: if the storage method or location changes, it will be sufficient to update the entries in the Images table without altering the Articles table.

Although this separation turned out to be only partially necessary in the specific context of this project, we chose to keep it in place to ensure a more robust and scalable architecture for potential future extensions.

Database design



Database schemas

```
CREATE DATABASE IF NOT EXISTS Progetto_Tiw;
USE Progetto_Tiw;

-- Stores User's informations and login credentials
CREATE TABLE Users (
    Id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    -- Login Username
    Username VARCHAR(255) NOT NULL UNIQUE CHECK (LENGTH(TRIM(Username)) > 0),
    -- Password
    Psw VARCHAR(255) NOT NULL CHECK (LENGTH(TRIM(Psw)) > 0),
    -- Anagraphics
    FirstName VARCHAR(255) NOT NULL CHECK (LENGTH(TRIM(FirstName)) > 0),
    LastName VARCHAR(255) NOT NULL CHECK (LENGTH(TRIM(LastName)) > 0),
    -- User Address, used for shipping Auctioned Items
    Address VARCHAR(255) NOT NULL CHECK (LENGTH(TRIM(Address)) > 0)
);
```

```sql
-- Stores the File names and paths for Image resources of Items
CREATE TABLE Images (
    Id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    -- Name of the image file
    FileName VARCHAR(255) NOT NULL CHECK (LENGTH(TRIM(FileName)) > 0),
    -- Path inside the FileSystem to locate the file
    FilePath VARCHAR(255) NOT NULL UNIQUE CHECK (LENGTH(TRIM(FilePath)) > 0)
);

-- Stores all the Auctions created by Users, with all related data
CREATE TABLE Auctions (
    Id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    -- Starting Price for the Auction, calculated as the Sum of all the Item's Prices
    BasePrice INT NOT NULL CHECK (BasePrice > 0),
    -- Minimum Allowed Increment for Offers
    MinIncrement INT NOT NULL CHECK (MinIncrement > 0),
    -- The ID of the current Highest Offer for this Auction
    -- NULL if no Offers have been made yet
    HighestBidId INT,
    -- It will be added at the end of the SCRIPT via an ALTER TABLE
    -- Ending Date for the Auction
    ClosingDate DATETIME NOT NULL,
    -- The ID of the User who created the Auction
    SellerId INT NOT NULL,
    -- A Boolean value to determine whether the Auction has been closed or not
    IsSold BOOLEAN DEFAULT FALSE NOT NULL,
    -- The ID of the User who won the Auction
    BuyerId INT,
    -- The Price of the winning Offer
    FinalPrice INT CHECK (FinalPrice IS NULL OR FinalPrice >= 0),
    -- Foreign Keys
    FOREIGN KEY (BuyerId) REFERENCES Users(Id)
    -- If the buyer User is deleted, the Auction is not deleted
      ON DELETE SET NULL
      ON UPDATE CASCADE,
    FOREIGN KEY (SellerId) REFERENCES Users(Id)
       -- Users with Auctions can't delete their profile
       ON DELETE RESTRICT
       ON UPDATE CASCADE
    -- FOREIGN KEY (HighestBidId) REFERENCES Offers(Id)
       -- In case the Offer is deleted, the Auction's reference is set to NULL
       -- Added later via ALTER TABLE
       -- ON DELETE SET NULL
       -- ON UPDATE CASCADE
);
```

```sql
-- Stores the Offers made by the Users inside Auctions
CREATE TABLE Offers (
    Id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    -- The ID of the User who made the Offer
    UserId INT NOT NULL,
    -- The ID of the Auction where the Offered has been made
    AuctionId INT NOT NULL,
    -- The Price offered by the User
    OfferedPrice INT NOT NULL CHECK (OfferedPrice > 0),
    -- The Date and Time the Offer has been made
    OfferDate DATETIME NOT NULL,
    -- Foreign Keys
    FOREIGN KEY (UserId) REFERENCES Users(Id)
       ON DELETE CASCADE
       ON UPDATE CASCADE,
    FOREIGN KEY (AuctionId) REFERENCES Auctions(Id)
       ON DELETE CASCADE
       ON UPDATE CASCADE
);

/*
    Stores the Items created by the Users
    Items can be part of an Auction or not
*/
CREATE TABLE Items (
    Id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    -- The Name of the Item
    ItemName VARCHAR(255) NOT NULL CHECK (LENGTH(TRIM(ItemName)) > 0),
    -- The Description of the Item
    ItemDescription VARCHAR(1023) NOT NULL CHECK (LENGTH(TRIM(ItemDescription)) > 0),
    -- The Price of the Item
    Price INT NOT NULL CHECK (Price > 0),
    -- The ID of the Image of the Item, used to retrieve the image FilePath
    ImageId INT,
    -- The ID of the User who created the Item
    CreatorId INT NOT NULL,
    /*
       The ID of the Auction in which the Item is being sold.
       An Item can only reference 1 Auction, so no duplicates allowed.
       It can also be NULL, meaning the Item does not belong to any Auction yet.
    */
    AuctionId INT,
    -- Foreign Keys
    FOREIGN KEY (ImageId) REFERENCES Images(Id)
       -- Images can't be deleted if they belong to an Item
       ON DELETE RESTRICT
       ON UPDATE CASCADE,
    FOREIGN KEY (CreatorId) REFERENCES Users(Id)
       ON DELETE CASCADE
       ON UPDATE CASCADE,
    FOREIGN KEY (AuctionId) REFERENCES Auctions(Id)
       /*
           If an Auction is Deleted, the Items are not deleted,
           but marked as not in an Auction
       */
       ON DELETE SET NULL
       ON UPDATE CASCADE
);
```

```sql
-- Add Auctions.HighestBid
ALTER TABLE Auctions
    ADD CONSTRAINT fk_highest_bid
    FOREIGN KEY (HighestBidId) REFERENCES Offers(Id)
       ON DELETE SET NULL
       ON UPDATE CASCADE;

-- Indexes for performance
CREATE INDEX idx_offers_auction ON Offers(AuctionId);
CREATE INDEX idx_offers_user ON Offers(UserId);
CREATE INDEX idx_items_auction ON Items(AuctionId);
CREATE INDEX idx_items_creator ON Items(CreatorId);
CREATE INDEX idx_items_image ON Items(ImageId);
CREATE INDEX idx_auctions_seller ON Auctions(SellerId);
CREATE INDEX idx_auctions_buyer ON Auctions(BuyerId);
```

Functional analysis

Pure HTML version

Analysis

Legend:

- Pages
- Components (View)
- Actions
- Events

A web application allows you to manage online auctions.

Users log via login and can sell and buy articles sold in auctions.

Each user has a username, password, first name, last name and address (used to ship the bought articles).

Each article has a code, name, description, image and price.

The HOME page contains two links, one to access the SELL page and one to access the BUY page.

The SELL page shows a list of auctions created by the user and not closed yet, a list of auctions created by him and closed and two forms, one to create a new article and one to create a new auction to sell the user's articles.
The first form allows to insert an article into the database with all its data, which are all mandatory.
The second form shows a list of the articles in the database and available to be sold and allows to select more than one to create an auction that contains them.

An auction is made of one or more articles being sold, the initial price of the set of articles, the minimum increment of each offer (as an integer number of euros) and an expiration date (date and time, ex. 19-04-2021 at 24:00).
The auction's initial price is obtained as the sum of the comprised articles' prices in the offer.
The same article cannot be included in different auctions.
Once sold, an article cannot be available to be inserted into other auctions.

The list of auctions in the SELL page is ordered by date+time in ascending order.
The list shows: code and name of the articles inside the auction, maximum offer, remaining time (number of days and hours) between the moment (date and time) of the login and the auction's closing date and time.

Clicking on an auction inside the list an AUCTION DETAILS page appears, which shows for an open auction all the auction's data and the list of offers (username, offered price, date and time of the offer) ordered by date+time descending.

A CLOSE button allows the user to close the auction if the expiration date is passed (ignore the case of expired auctions that have not been closed by the user and don't implement an automatic closing of auctions after the expiration date).
If the auction is closed, the page shows all the auction's data, the buyer's name, the final price and the (fixed) shipping address of the user.

The BUY page contains a form to search for keyword.
When the buyer sends a keyword the BUY page is refreshed and shows a list of open auctions (whose expiration date is after the date and time of submission) for which the keyword appears inside the name or description of at least one of the auction's articles.
The list is ordered in a descending order based on the time (number of days and hours) remaining until closure.

Clicking on an open auction an OFFER page appears, showing all the articles' data, the list of the arrived offers in descending date+time order and an input field to send their own offer, which must be higher than the current max offer by an amount equal at least to the minimum increment.
After the submission the OFFER page shows the updated offers list.

The BUY page also contains a list of the offers won by the user with the article's data and the final price.

Completion of specifications

- The LOGIN page contains a form.
- The LOGIN page contains a link to navigate to a SIGN UP page with a form where the user can create an account.
- The SIGN UP page contains a link to return to the LOGIN page.
- The HOME, BUY, SELL, AUCTION DETAILS and OFFERS pages contain a button that allows the user, when clicked, to logout from the website.
- The BUY and SELL pages contain a link that allows to return to the HOME page.
- The lists of Auctions contain a reusable component, which shows all the auction's data. This component also contains a link that, when clicked, shows the AUCTION DETAILS or OFFERS page.
- The AUCTION DETAILS and OFFERS pages contain a link that allows to return to the previous page.

Summary and completing of the functional analysis

Pages and interface components:

- Login
  - Login form
  - Login button
  - Link to the SIGN UP page
- Sign Up *
  - Sign up form *
  - Submit button
  - Link to return to the LOGIN page
- Home
  - Link to access the SELL page
  - Link to access the BUY page
  - Link to logout and return to the LOGIN page
- Sell
  - Link to return to the HOME page
  - Link to logout and return to the LOGIN page
  - List of the auctions created by the user and still open
  - List of the auctions created by the user and closed
  - Form to create a new Article
  - Form to create a new Auction
- Buy
  - Link to return to the HOME page
  - Link to logout and return to the LOGIN page
  - Form to perform a search
  - List of the auctions found by the search **
  - List of the auctions won by the user
- Auction Details (Auction Details and Offers) ***
  - Link to return to the previous page
  - Link to logout and return to the LOGIN page
  - Auction's data
  - Buyer user's data (optional)
  - List of the Auction's Articles
  - Button to close the Auction (optional) ****
  - List of the Offers to the Auction (optional)
  - Form to send a new Offer (optional)

* For *User Experience* reasons we decided to split the registration phase into two separate pages, so the form was split in two parts too.

** For *UX* reasons we decided to hide the section for the search results if nothing has been searched yet.
The moment the *query* parameter is found inside the URL this component is shown too, but if it's missing, we assume that the user has loaded the page from scratch and so he hasn't made any search yet.
The form that serves as a search bar, however, is always visible.

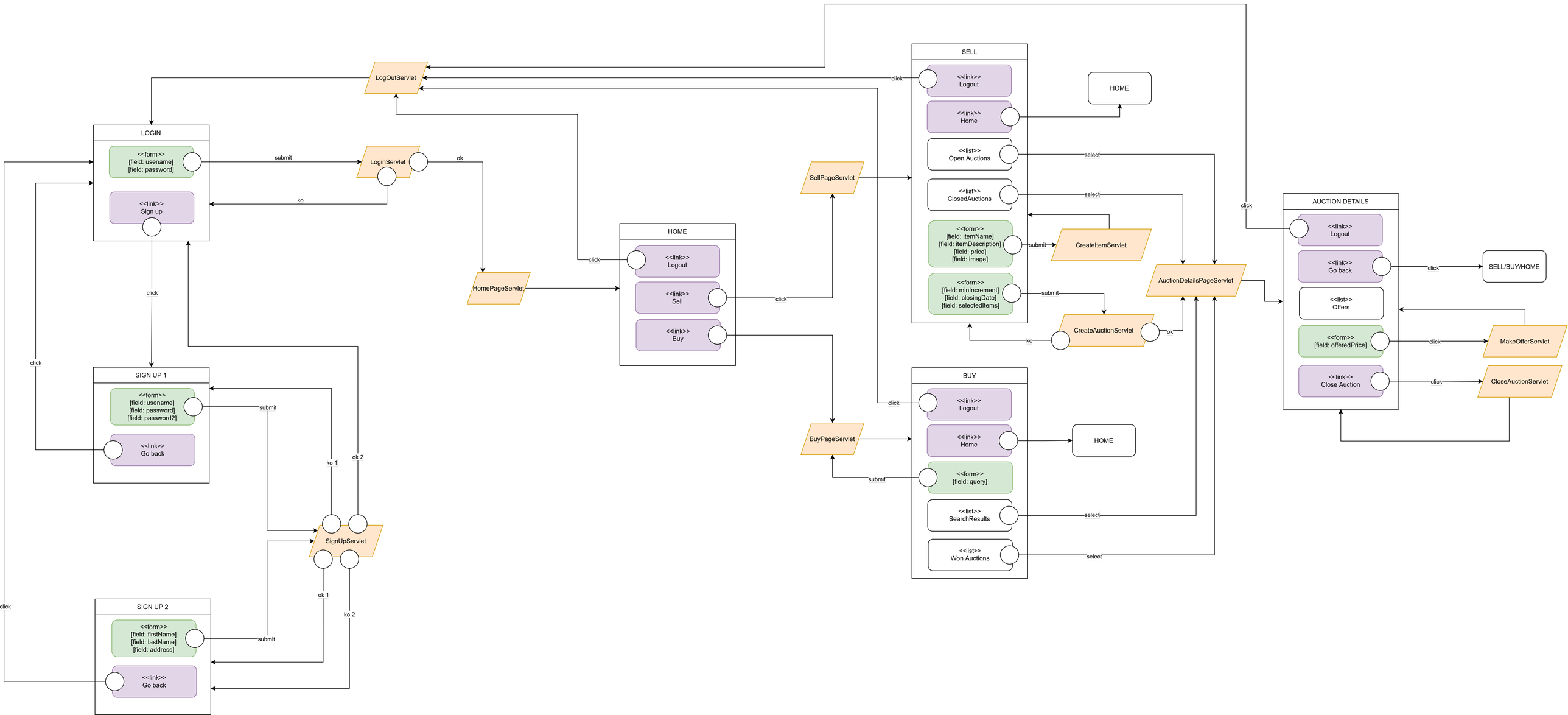
***For *UX* reasons and for a better code management the two pages AUCTION DETAILS and OFFERS have been merged into one.
These two pages should show the same content, the auction's data, with the exception of the second one, which also shows the list of Offers sent to that auction, and the form to allow the user to send his own offer.
This section of the UI can be shown and hidden programmatically to obtain one or the other view from the same source page.

**** Only shown if the user is the auction's creator.

Events and actions:

- Click on the login button (or form submit)
    - Credentials check and authentication
- Click on the “Sign Up” button under the Login form
    - Redirect to the SignUp 1 page
- Click on the “Continue” button in the first “Sign Up” form (or form submit)
    - Check availability for the username and password validation
    - Redirect to the SignUp 2 page
- Click on the “Sign Up” button in the second SignUp form (or form submit)
    - Check availability of the username and password validation *
    - Redirect to the Login page
- Click on the “Go back” button in the two SigUp pages
    - Redirect to the Login page
- Click on the “Logout” button in the Home, Sell, Buy and AuctionDetails pages
    - Session invalidation
    - Redirect to the Login page
- Click on the “Sell” button in the Home page
    - Redirect to the Sell page
- Click on the “Buy” button in the Home page
    - Redirect to the Buy page
- Click on the “Home” button in the Sell and Buy pages
    - Redirect to the Home page
- Click on the “Create Item” button in the Sell page (or form submit)
    - Sends data to CreateItemServlet
    - Refresh the Sell page with the new Article or the error message
- Click the “Create Auction” button in the Sell page (or form submit)
    - Sends the data to CreateAuctionServlet
    - Refresh the Sell page with the new Auction or the error message
- Click on the “Search” button in the Buy page (or form submit)
    - Redirect to the Buy page (same page) with the query attribute in the request
    - Refresh the page with the search results (either the list of auctions or the signal message)
- Click on the “See more” button in any of the lists of Auctions in the Sell and Buy pages
    - Redirect to the AuctionDetails page
- Click on the “Send” button in the form to send offers in the AuctionDetails page (or form submit)
    - Sends the data to MakeOfferServlet
    - Refresh the page AuctionDetails with the new Offer or the error message
- Click on the “Close Auction” button in the AuctionDetails page
    - Sends the data to CloseAuctionServlet
    - Refresh the AuctionDetails page with the new UI or the error message.

* After the first check the username and password (both) data are inserted into some hidden fields in the second form, and are re-sent with the second submit.
To avoid the user editing those credentials during this second step without harming the database or the system, credentials are checked and validated once again, and if there is anything wrong with them, the user would be redirected to the first SignUp form once again, resetting the registration attempt and forcing him to restart from scratch.
The proper error message would also be shown alongside the form.
We want to point out that this second validation step does not keep memory of what was already checked and validated previously. So it's entirely possible that the user modifies his own credentials in the hidden fields of the second form with some other credentials just as valid, and these would be accepted by the system. The goal of this second check is not to guarantee a registration process impossible to tamper with, rather to protect the system from the insertion of credentials that are not conforming to the specifications or are duplicate usernames, which would compromise the security of the Login system.
Because of that, only the credentials in the second submit are actually considered and inserted into the Database.


LOGIN
<<form>>
[field: usename]
[field: password]
<<link>>
Sign up
submit
LoginServlet
ok
ko
click
SIGN UP 1
<<form>>
[field: usename]
[field: password]
[field: password2]
<<link>>
Go back
submit
ko 1
ok 2
SignUpServlet
SIGN UP 2
<<form>>
[field: firstName]
[field: lastName]
[field: address]
<<link>>
Go back
submit
ok 1
ko 2
click
LogOutServlet
HOME
<<link>>
Logout
<<link>>
Sell
<<link>>
Buy
click
HomePageServlet
click
SellPageServlet
BuyPageServlet
SELL
<<link>>
Logout
<<link>>
Home
<<list>>
Open Auctions
<<list>>
ClosedAuctions
<<form>>
[field: itemName]
[field: itemDescription]
[field: price]
[field: image]
<<form>>
[field: minIncrement]
[field: closingDate]
[field: selectedItems]
HOME
select
select
submit
CreateItemServlet
submit
CreateAuctionServlet
ko
ok
BUY
<<link>>
Logout
<<link>>
Home
<<form>>
[field: query]
<<list>>
SearchResults
<<list>>
Won Auctions
HOME
submit
select
select
click
AuctionDetailsPageServlet
click
AUCTION DETAILS
<<link>>
Logout
<<link>>
Go back
<<list>>
Offers
<<form>>
[field: offeredPrice]
<<link>>
Close Auction
SELL/BUY/HOME
click
MakeOfferServlet
click
CloseAuctionServlet
click

Components

Model Objects (Beans):

- User
- Item (Article)
- Auction
- Offer
- Image

Data Access Objects (DAOs):

- UserDAO:
  - User checkCredentials(String username, String password)
  - void validateCredentials(String username, String password1, String password2)
  - User getUserById(int userId)
  - void insert(User user, String password)
- ItemDAO:
  - int insert(Item item)
  - void updateItemsAuctionId(List<Item> items, int auctionId)
  - void addImageToItem(int itemId, int imageId)
  - List<Item> getItemsInAuction(int auctionId)
  - List<Item> getItemsByIDs(List<Integer> itemIds)
  - List<Item> getAvailableItemsForUserId(int userId)
- AuctionDAO:
  - int insert(Auction auction)
  - void markAuctionAsClosed(Auction auction)
  - void updateAuctionsHighestBid(int auctionId, int offerId)
  - Auction getAuctionById(int id, long loginTime)
  - List<Auction> getAuctionsCreatedBy(User user, boolean closed, long loginTime)
  - List<Auction> getAuctionsNotCreatedBy(User user, long loginTime)
  - List<Auction> getAuctionsWonBy(User user, long loginTime)
  - List<Auction> getAuctionsForKeywords(String[] keywords, int userId, long loginTime)
- OfferDAO:
  - int insert(Offer offer)
  - Offer getOfferById(int offerId)
  - List<Offer> getOffersForAuction(int auctionId)
- ImageDAO:
  - Image getImageById(int id)
  - int insert(String fileName, String filePath)
  - void delete(int imageId)

Controller or Servlet:

- LoginServlet
- SignUpServlet
- LogOutServlet
- CloseAuctionServlet
- CreateAuctionServlet
- CreateItemServlet
- FrontServlet
- MakeOfferServlet

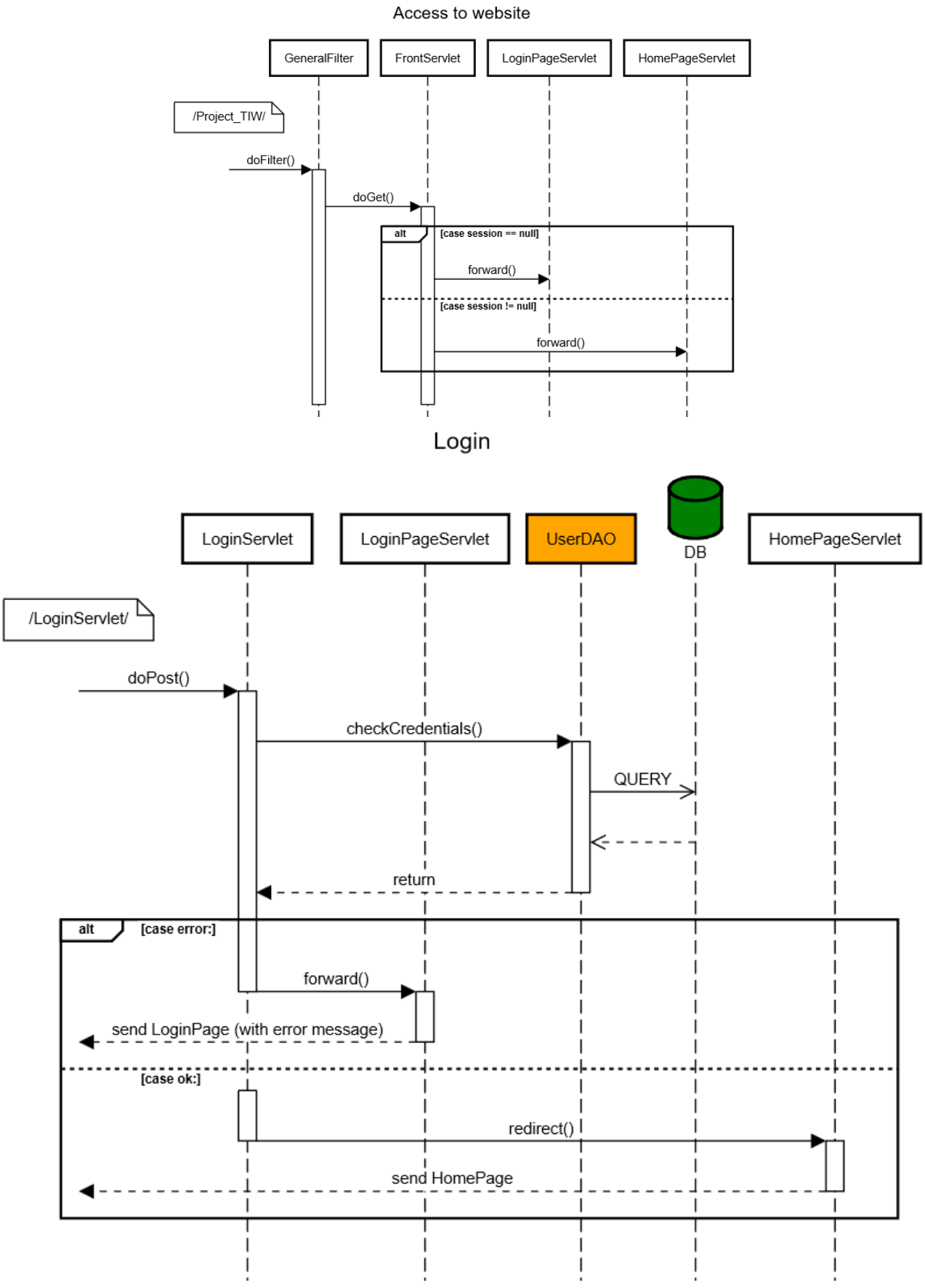
Servlet dedicated to pages (PageServlet):

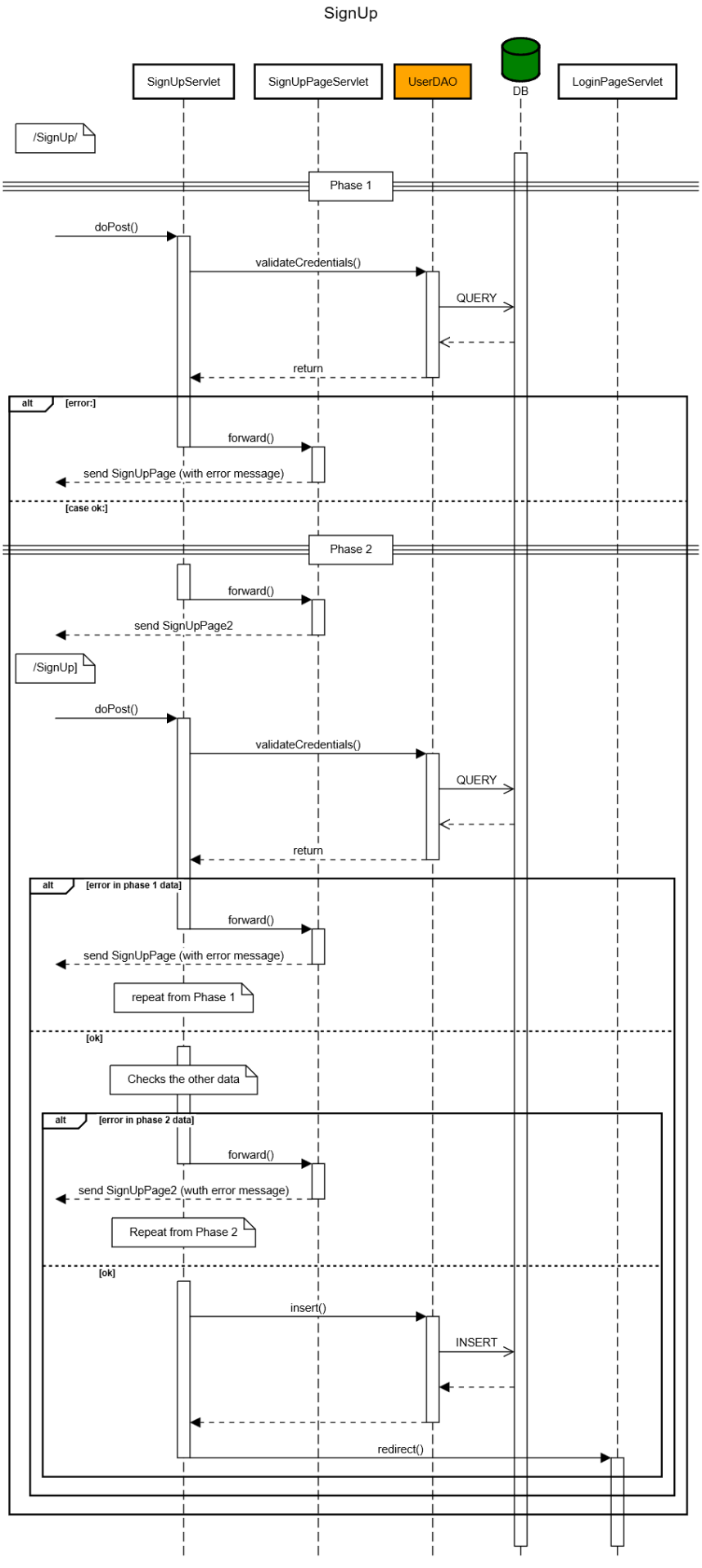
- AuctionDetailsPageServlet
- BuyPageServlet
- ErrorPageServlet
- HomePageServlet
- LoginPageServlet
- SellPageServlet
- SignUpServlet

Pages (template):

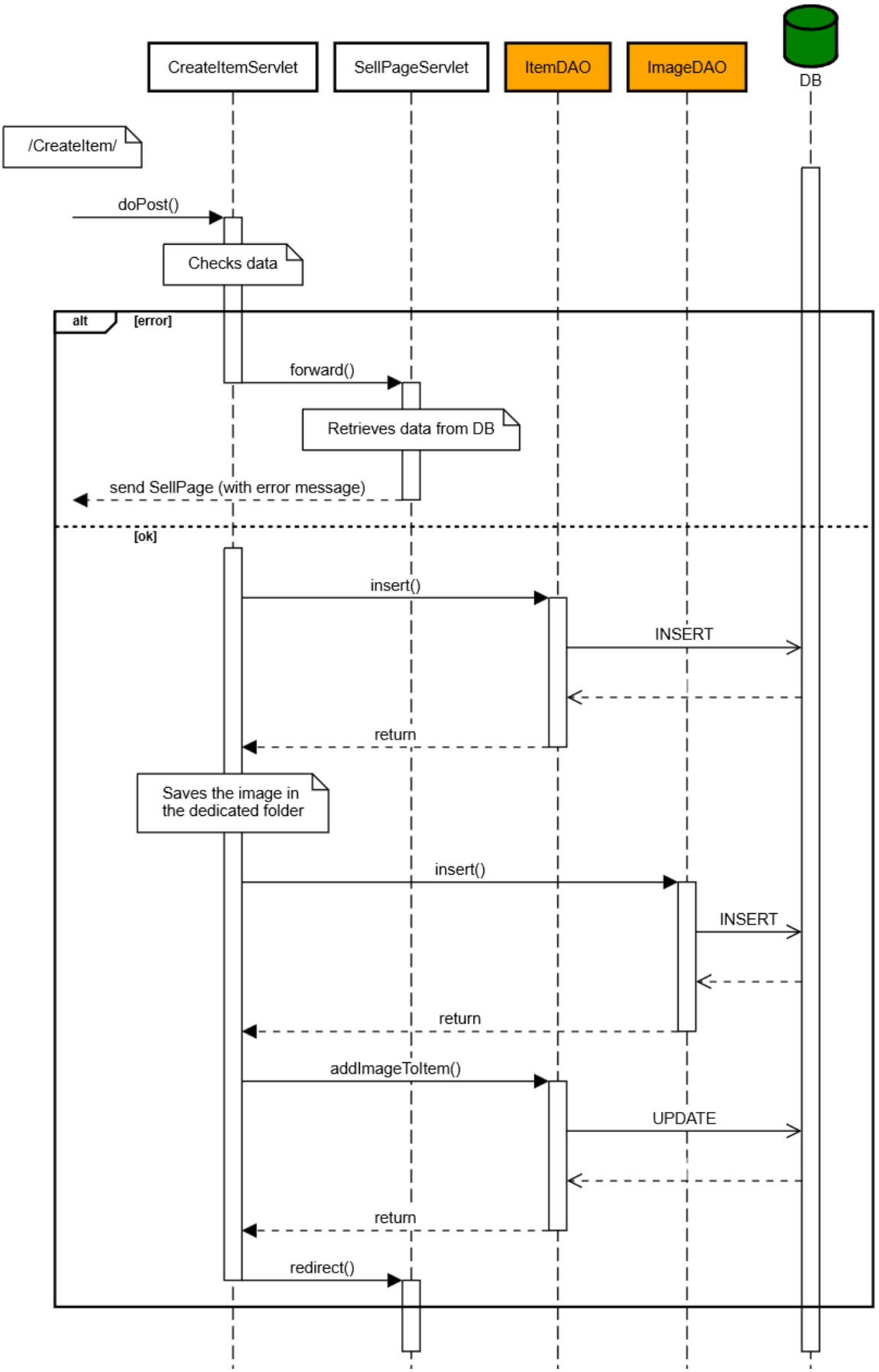
- AuctionDetailsPage
- BuyPage
- ErrorPage
- HomePage
- LoginPage
- SellPage
- SignUpPage
- SignUpPage2

Sequence Diagrams

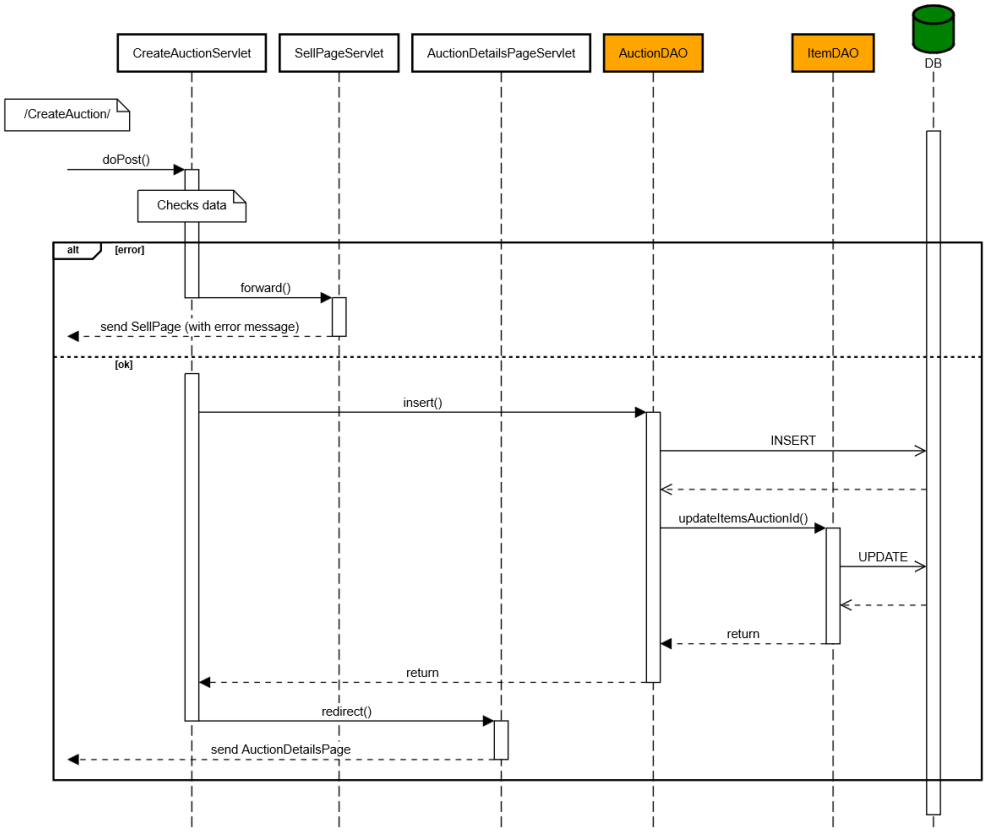



SignUp
SignUpServlet
SignUpPageServlet
UserDAO
DB
LoginPageServlet
/SignUp/
Phase 1
doPost()
validateCredentials()
QUERY
return
alt
[error:]
forward()
send SignUpPage (with error message)
[case ok:]
Phase 2
forward()
send SignUpPage2
/SignUp]
doPost()
validateCredentials()
QUERY
return
alt
[error in phase 1 data]
forward()
send SignUpPage (with error message)
repeat from Phase 1
[ok]
Checks the other data
alt
[error in phase 2 data]
forward()
send SignUpPage2 (wuth error message)
Repeat from Phase 2
[ok]
insert()
INSERT
redirect()

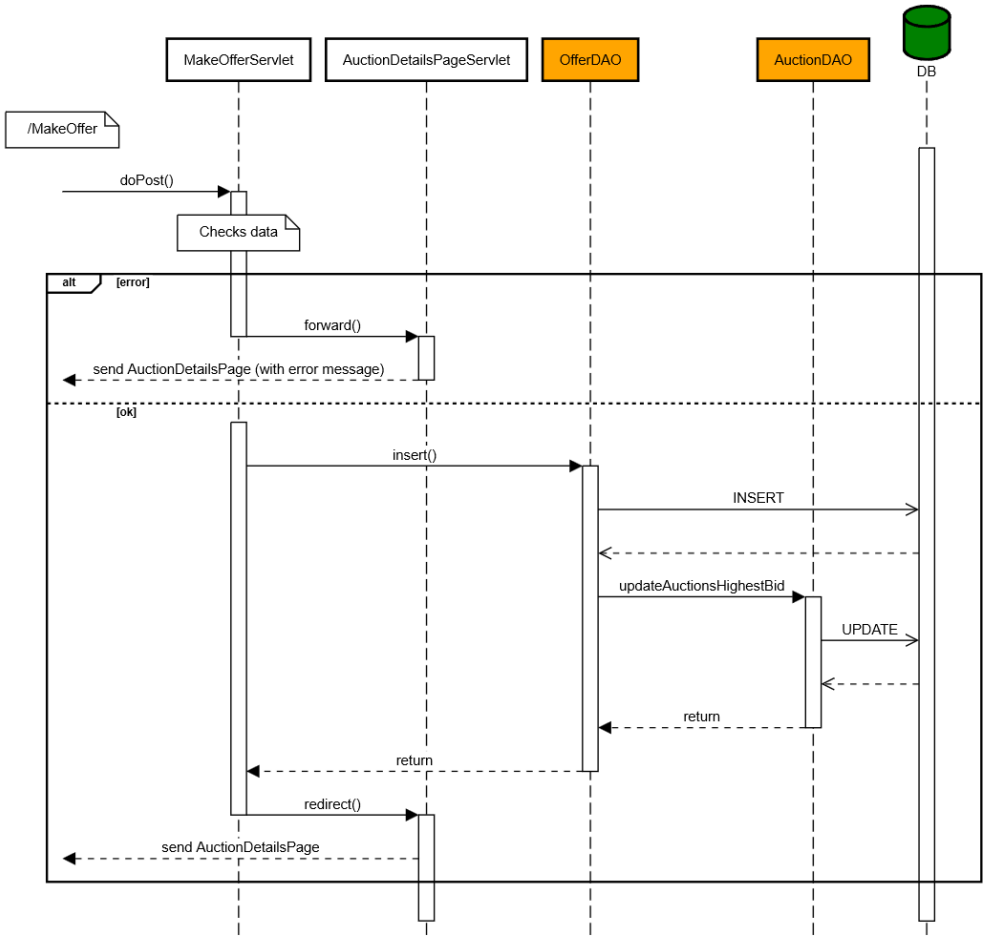
Creating an Item



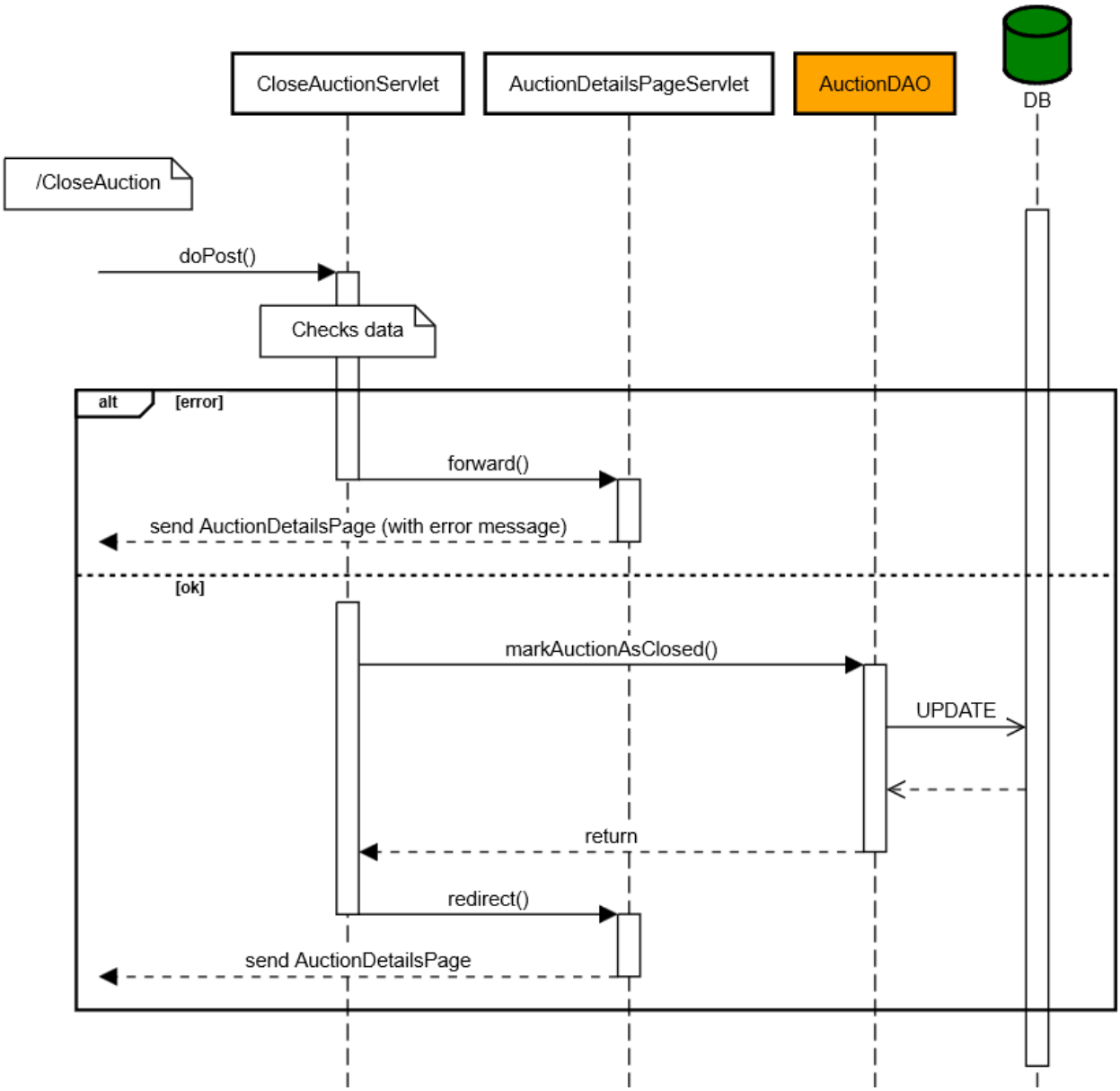
Creating an Auction



Making an Offer



Closing an Auction



View and interface

For this version of the project it was sufficient to implement the various requested pages in individual web pages, so the following pages have been implemented:

- **LoginPage**: contains a form in which the user can insert its username and password to authenticate.
- **SignUpPage**: contains the form in which the user inserts its username and password (password is inserted twice for confirmation) for its new account.
- **SignUpPage2**: contains a form in which the user inserts its name, last name and address to complete the account creation process.
- **HomePage**: contains two links, one to SellPage and one to BuyPage.
- **SellPage**: contains the list of Auctions created by the user and still open, the list of Auctions created by the user and closed, a form in which he can create a new Article and one in which he can create a new Auction.
- **BuyPage**: contains a search field, the list of search results and the list of Auctions won by the user
- **AuctionDetailsPage**: Shows all the details of an Auction, adapting its UI to the Auction's status.

We decided to use *Thymeleaf* to assist in the process of personalizing the pages.
Therefore, all these pages contain numerous tags from this framework and able to adapt the presentation and the content of the pages based on the data inside the request.

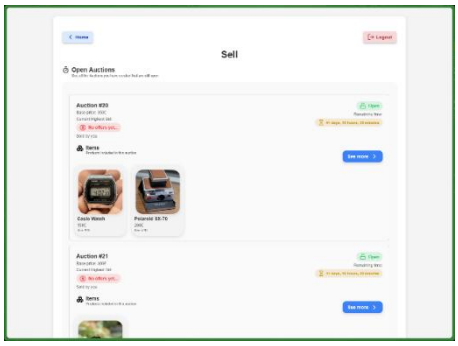
To do so, however, it's necessary that the server calls the `process()` method from *Thymeleaf 's TemplateEngine* to render an HTML page before sending it to the client. A set of dedicated Servlets have been implemented, one for each page.
These Servlets have been appropriately named with the suffix "*PageServlet*" for coherency, and they are:

- LoginPageServlet (/login)
- SignUpPageServlet (for both pages) (/signup)
- HomePageServlet (/home)
- SellPageServlet (/sell)
- BuyPageServlet (/buy)
- AuctionDetailsPageServlet (/auction-details)

To these Servlets have been assigned some simple and distinctive URLs, since they are meant to be usable by the user.

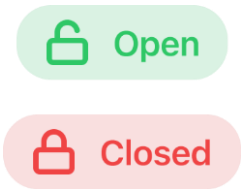
Inside the individual pages we find a relatively similar layout, with a re-use of many CSS classes to guarantee continuity and coherence.

In here there are a mix of elements statically build along with some dynamically built ones, like the components that represent the single Auctions into the various lists, the cards that represent the Auction's Articles or the bubbles that contain the Offers.



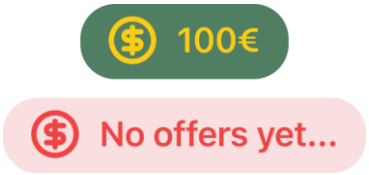
Example of page: SellPage

In this versione of the project, these elements have been written once HTML and iterated through an `th:each` from the *Thymeleaf TemplateEngine*.
For the content whose presence depends on the data represented we used `th:if` and `th:unless`.



Auction's status indicators

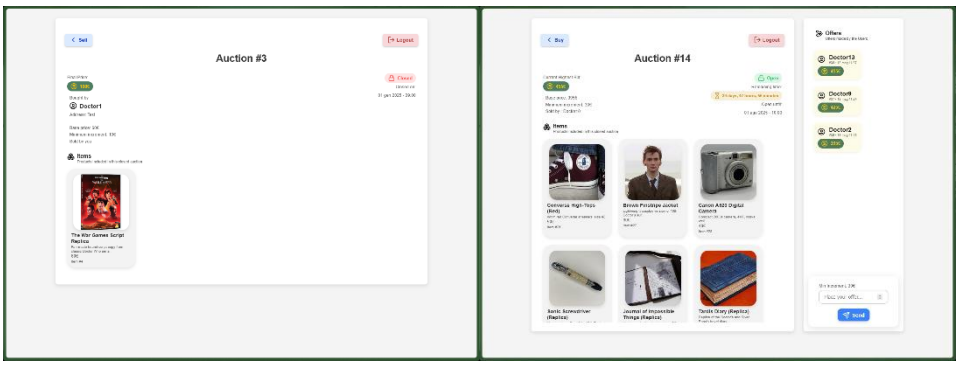
This allows to alternatively show one or the other indicator for open and closed Auctions, or to show the max offer indicator or the final price one instead of the one that signals the absence of offers.



At the top an examlpe of the highest offer indicator, at the bottom the one that signals the absence of Offers

Also, the `th:if` instruction has allowed us to reduce the number of pages from what is expressed into the specifications to what is actually implemented: according to the specs, opening the details of an open auction should show a page containing all the auction's data and the list of offers, while opening a closed auctions should show a page with only the auction's details.

Given that the two pages share the vast majority of their content, we decided to implement a single page, made up by a container with all the auction's details, whose presence, absence and content is controlled by the data from the Auction, and a second container on the right hand side dedicated to the Offers, whose presence or absence is controlled by a `th:if` on the container itself.



JavaScript version

Analysis

Legenda:

- Pages
- Components (View)
- Actions
- Events

A web application allows you to manage online auctions.

Users log via login and can sell and buy articles sold in auctions.

Each user has a username, password, first name, last name and address (used to ship the bought articles).

Each article has a code, name, description, image and price.

The HOME page contains two links, one to access the SELL page and one to access the BUY page.

The SELL page shows a list of auctions created by the user and not closed yet, a list of auctions created by him and closed and two forms, one to create a new article and one to create a new auction to sell the user's articles.
The first form allows to insert an article into the database with all its data, which are all mandatory.
The second form shows a list of the articles in the database and available to be sold and allows to select more than one to create an auction that contains them.

An auction is made of one or more articles being sold, the initial price of the set of articles, the minimum increment of each offer (as an integer number of euros) and an expiration date (date and time, ex. 19-04-2021 at 24:00).
The auction's initial price is obtained as the sum of the comprised articles' prices in the offer.
The same article cannot be included in different auctions.
Once sold, an article cannot be available to be inserted into other auctions.

The list of auctions in the SELL page is ordered by date+time in ascending order.
The list shows: code and name of the articles inside the auction, maximum offer, remaining time (number of days and hours) between the moment (date and time) of the login and the auction's closing date and time.

Clicking on an auction inside the list an AUCTION DETAILS page appears, which shows for an open auction all the auction's data and the list of offers (username, offered price, date and time of the offer) ordered by date+time descending.

A CLOSE button allows the user to close the auction if the expiration date is passed (ignore the case of expired auctions that have not been closed by the user and don't implement an automatic closing of auctions after the expiration date).
If the auction is closed, the page shows all the auction's data, the buyer's name, the final price and the (fixed) shipping address of the user.

The BUY page contains a form to search for keyword.
When the buyer sends a keyword the BUY page is refreshed and shows a list of open auctions (whose expiration date is after the date and time of submission) for which the keyword appears inside the name or description of at least one of the auction's articles.
The list is ordered in a descending order based on the time (number of days and hours) remaining until closure.

Clicking on an open auction an OFFER page appears, showing all the articles' data, the list of the arrived offers in descending date+time order and an input field to send their own offer, which must be higher than the current max offer by an amount equal at least to the minimum increment.
After the submission the OFFER page shows the updated offers list.

The BUY page also contains a list of the offers won by the user with the article's data and the final price.

Build a client server web application that extends and/or modifies the previous specifications as follows:

- After login, the entire application is built in a single page
- If the user logs in for the first time the application shows the content of the BUY page. If the user has already used the application, it shows the content of the SELL page if the last action performed by the user was creating an auction; otherwise, it shows the content of the BUY page with the list (even if empty) of the auction that the user had previously clicked and that are still open.
  The information about the last performed action and of the visited auctions is saved on the client-side for a month.

Every user interaction is handled without completely reloading the page; instead produces the asynchronous invocation of the server and the modification only of the content to be updated after the event.

Completion of specifications

- The LOGIN page contains a form.
- The LOGIN sectio in the page contains a button to show a second section SIGN UP, with inside a form where the user can create an account.
- The SIGN UP section contains a button to show the LOGIN section of the page.
- The HOME page contains the sections BUY, SELL, AUCTION DETAILS and OFFERS.
- The HOME page contains a botton that allows the user, when clicked, to logout from the website.
- The HOME page contains a selector that allows to navigate through the sections SELL and BUY.
- The sections BUY and SELL contains a reusabke component, which shows all the auction's data. This component contains a button that, when clicked, shows the section AUCTION DETAILS or OFFERS.
- The sections AUCTION DETAILS and OFFERS contain a button that allows to return to the previous view.

Summary and completing of the functional analysis

Pages and interface components:

- Login
    - Login
        - Login form
        - Login button
        - Button to show the SIGN UP section
    - Sign Up *
        - Signup form *
        - Submit button
        - Button to show the LOGIN section
- Home
    - Link to logout and return to the LOGIN page
    - Selector to navigate between SELL and BUY sections
    - Sell
        - List of auctions created by the user and still open
        - List of the auctions created by the user and closed
        - Form to create a new Article
        - Form to create a new Auction
    - Buy
        - Form to perform a search
        - List of auctions found by the search **
        - List of auctions recently visited
        - List of auctions won by the user
    - Auction Details (Auction Details and Offers) ***
        - Button to hide the section and show the previous one
        - Auction's details
        - Buyer user's data (optional)
        - List of Articles inside the Auction
        - Button to close the Auction (optional) ****
        - List of Offers sent to the Auction (optional)
        - Form to send a new Offer (optional)

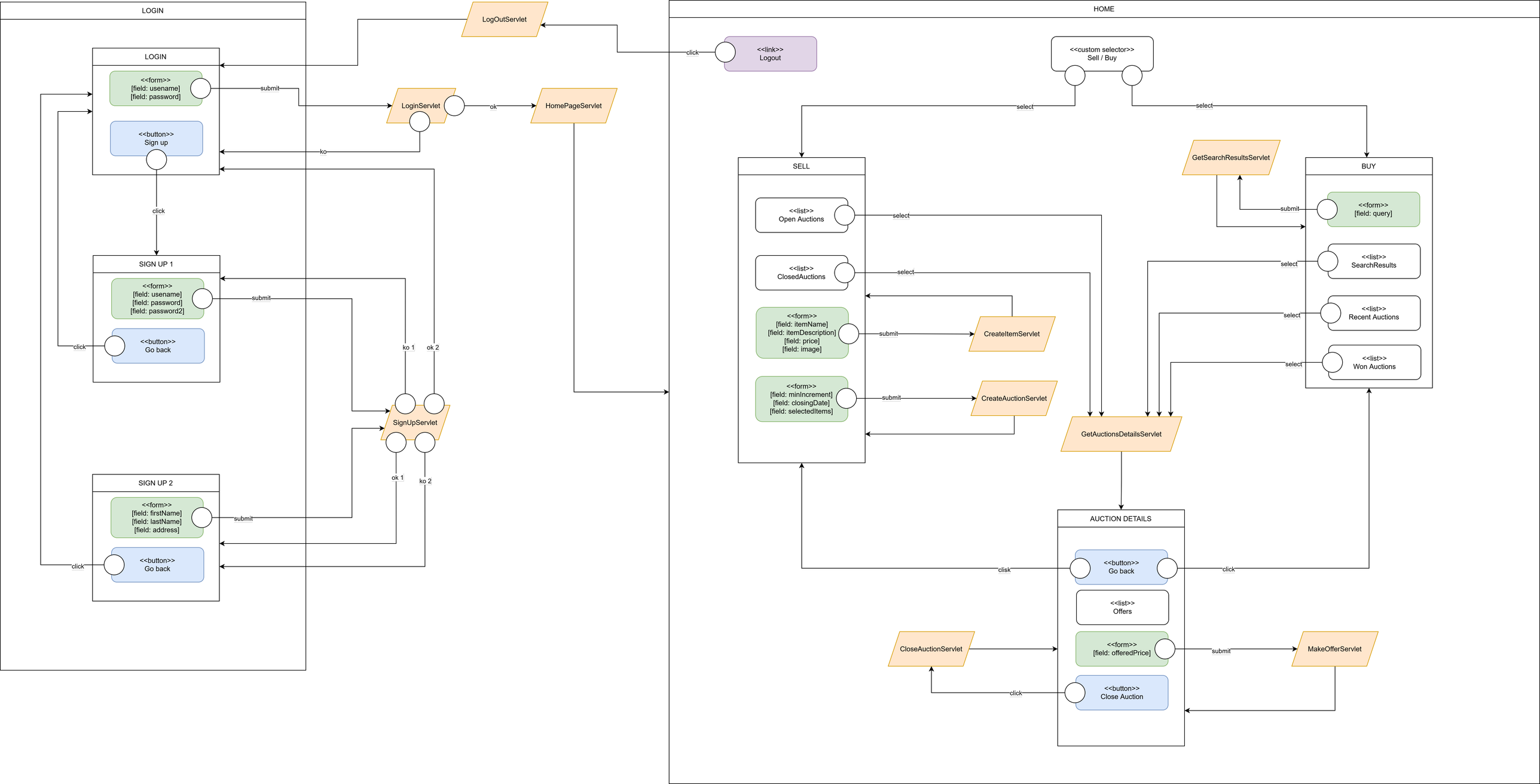
* For *User Experience* reasons we decided to split the registration phase into two separate sections, so the form was split in two forms too.

** For *UX* reasons we decided to hide the sections with the search results when the user has not performed any search yet.
Since we couldn't reach this section with a specific URL with a query inside, unlike the other version, we assume that the page is always "clean" at the first load, so without any search.

The page is automatically refreshed when the asynchronous request of the search returns the auctions, which are added into the page via JavaScript.

*** For *UX* reasons and to improve the overall code management the two sections AUCTION DETAILS and OFFERS have been merged into one.
These two sections should show the same content, the auction's data, with the exception of the second one, which also shows the list of Offers sent to that auction, and the form to allow the user to send his own offer.
This sub-section of the UI can be shown and hidden programmatically to obtain one or the other view from the same source section.

**** Only shown if the user is the auction's creator.


LOGIN
LOGIN
<<form>>
[field: usename]
[field: password]
submit
<<button>>
Sign up
LoginServlet
ok
HomePageServlet
ko
click
SIGN UP 1
<<form>>
[field: usename]
[field: password]
[field: password2]
submit
click
<<button>>
Go back
ko 1
ok 2
SignUpServlet
ok 1
ko 2
SIGN UP 2
<<form>>
[field: firstName]
[field: lastName]
[field: address]
submit
click
<<button>>
Go back
LogOutServlet
click
<<link>>
Logout
HOME
<<custom selector>>
Sell / Buy
select
select
SELL
<<list>>
Open Auctions
select
<<list>>
ClosedAuctions
select
<<form>>
[field: itemName]
[field: itemDescription]
[field: price]
[field: image]
submit
CreateItemServlet
<<form>>
[field: minIncrement]
[field: closingDate]
[field: selectedItems]
submit
CreateAuctionServlet
GetSearchResultsServlet
BUY
<<form>>
[field: query]
submit
<<list>>
SearchResults
select
<<list>>
Recent Auctions
select
<<list>>
Won Auctions
select
GetAuctionsDetailsServlet
AUCTION DETAILS
<<button>>
Go back
clisk
click
<<list>>
Offers
CloseAuctionServlet
<<form>>
[field: offeredPrice]
submit
MakeOfferServlet
<<button>>
Close Auction
click

Components

Model Objects (Beans):

- User
- Item (Article)
- Auction
- Offer
- Image

Data Access Objects (DAOs):

- UserDAO:
  - User checkCredentials(String username, String password)
  - void validateCredentials(String username, String password1, String password2)
  - User getUserById(int userId)
  - void insert(User user, String password)
- ItemDAO:
  - int insert(Item item)
  - void updateItemsAuctionId(List<Item> items, int auctionId)
  - void addImageToItem(int itemId, int imageId)
  - List<Item> getItemsInAuction(int auctionId)
  - List<Item> getItemsByIDs(List<Integer> itemIds)
  - List<Item> getAvailableItemsForUserId(int userId)
- AuctionDAO:
  - int insert(Auction auction)
  - void markAuctionAsClosed(Auction auction)
  - void updateAuctionsHighestBid(int auctionId, int offerId)
  - Auction getAuctionById(int id, long loginTime)
  - List<Auction> getAuctionsCreatedBy(User user, boolean closed, long loginTime)
  - List<Auction> getAuctionsNotCreatedBy(User user, long loginTime)
  - List<Auction> getAuctionsWonBy(User user, long loginTime)
  - List<Auction> getAuctionsForKeywords(String[] keywords, int userId, long loginTime)
- OfferDAO:
  - int insert(Offer offer)
  - Offer getOfferById(int offerId)
  - List<Offer> getOffersForAuction(int auctionId)
- ImageDAO:
  - Image getImageById(int id)
  - int insert(String fileName, String filePath)
  - void delete(int imageId)

Controller or Servlet:

- LoginServlet
- SignUpServlet
- LogOutServlet
- CloseAuctionServlet
- CreateAuctionServlet
- CreateItemServlet
- FrontServlet
- GetAuctionsDetailsServlet
- GetAvailableItemsServlet
- GetClosedAuctionsServlet
- GetOpenAuctionsServlet
- GetSearchResultsServlet
- GetUserInfoServlet
- GetVisitedAuctionsServlet
- GetWonAuctionsServlet
- MakeOfferServlet

Servlet dedicated to pages (PageServlet) *:

- LoginPageServlet
- HomePageServlet

Pages:

- LoginPage
- HomePage

* In this version of the project we decided to keep the PageServlet mechanism, the Servlets entirely dedicated to send HTML files, rather than allow direct access to files, so that we can use two simpler URLs (/login and /home).

Events and actions

N	Client-side		Server-side	
	Event	Action	Event	Action
1	Login -> login form -> submit	Login	POST (username, password)	Credential check and authentication
2	Login -> button Sign Up	Show Sign Up 1 form	-	-
3	SignUp1 -> signUp form -> submit	Credential check + Show SignUp 2 form	POST (username, password, password2)	Check username availability
4	SignUp 2 -> signUp form -> submit	Data check + Show Login form	POST (username, password, password2, firstName, lastName, address)	Check username availability + registration
5	SignUp 1/2 -> button Go Back	Show Login form	-	-
6	HomePage -> link Logout	Logout	GET()	Session invalidation + redirect to LoginPage
7	HomePage -> load	Retrieve ID and username	GET()	Send user data
8	HomePage->load	Retrieve available items for new auction	GET()	Send items data
9	HomePage -> load	Retrieve open auctions, closed auctions, won auctions	GET()	Send auctions data
10	HomePage->load	Show recent auctions	POST(AuctionIds)	Send auctions data
11	HomePage -> load	Read first access / last performed action and show the correct section	-	-
12	HomePage -> selection Buy/Sell	Show Buy/Sell section + refresh the associated lists	GET() (for refresh)	Send data
13	HomePage -> create item form -> submit	Send data to create the item + refresh available items in the auction creation form	POST(itemName, itemDescription, price, image) (item creation) + GET() (refresh)	Check data + insert in DB (item creation) + send data (refresh)

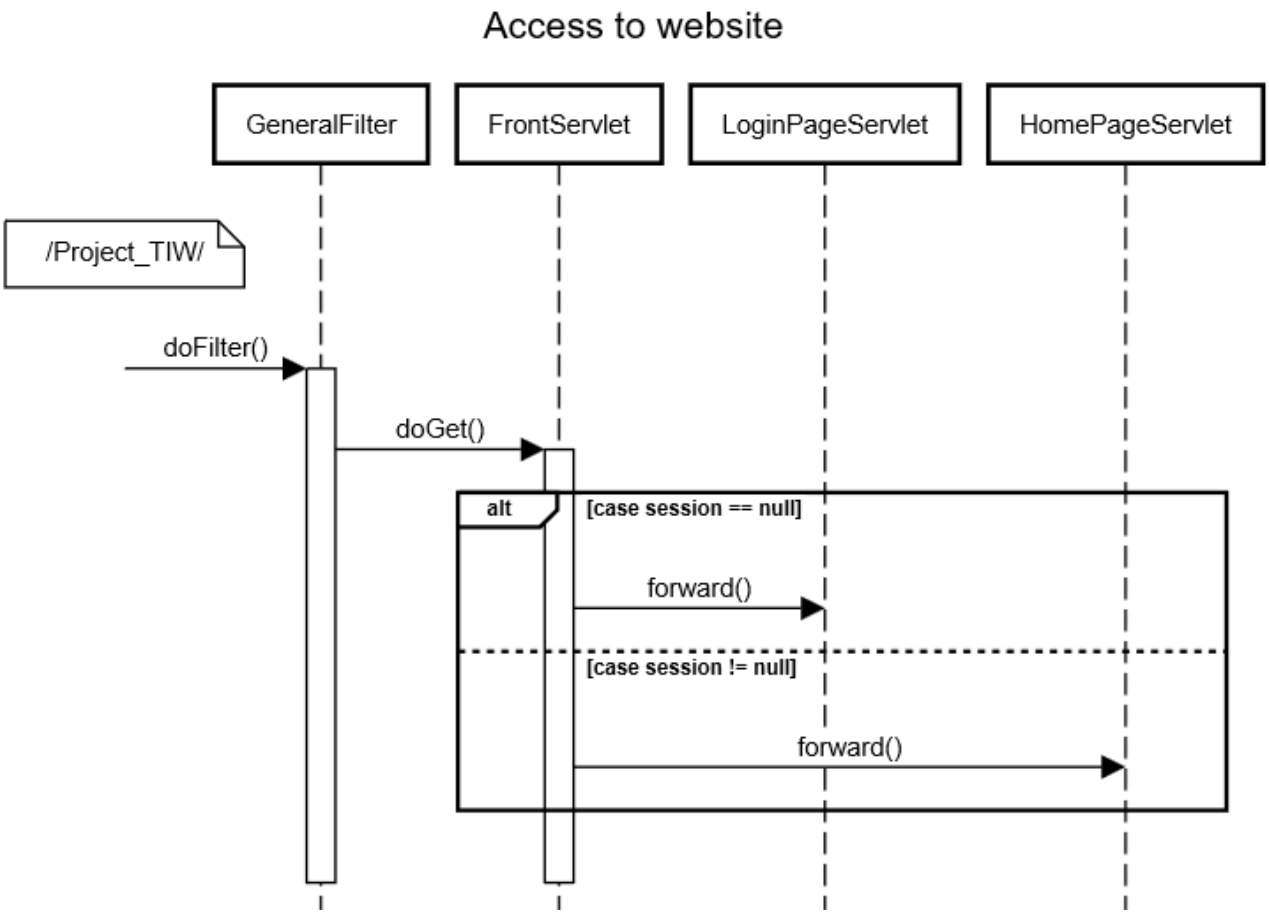
14	HomePage -> create auction form -> submit	Send data to create the auction + refresh open auctions and available items in the auction creation form	POST(minIncrement, closingDate, items) (auction creation) + GET() (refresh)	Check data + insert into DB (auction creation) + send data (refresh)
15	HomePage -> selection item in auction creation form checkbox -> check	Checkbox selection + aesthetic change of <label> (via CSS)	-	-
16	HomePage -> search form -> submit	Send query + refresh list of results (and show)	GET(query)	Send data
17	HomePage -> opening auction details (from any list) button -> click	Request updated data for the auction + show the popup + refresh the UI + save the ID in cookie	GET(id)	Send data
18	HomePage -> closing the auction details popup button -> click	Hide the popup	-	-
19	HomePage -> send offer form -> submit	Check data + send + refresh UI in auction details popup	POST(userId, auctionId, offerId) (send offer) + GET(id) (refresh)	Check data + insert in DB (send offer) + send data (refresh)
20	HomePage -> closing auction button -> click	Check + send + refresh UI in auction details popup	POST(userId, auctionId) (closing) + GET(id) (refresh)	Check data and validity + update DB (closing) + send data (refresh)

Controller end event handlers

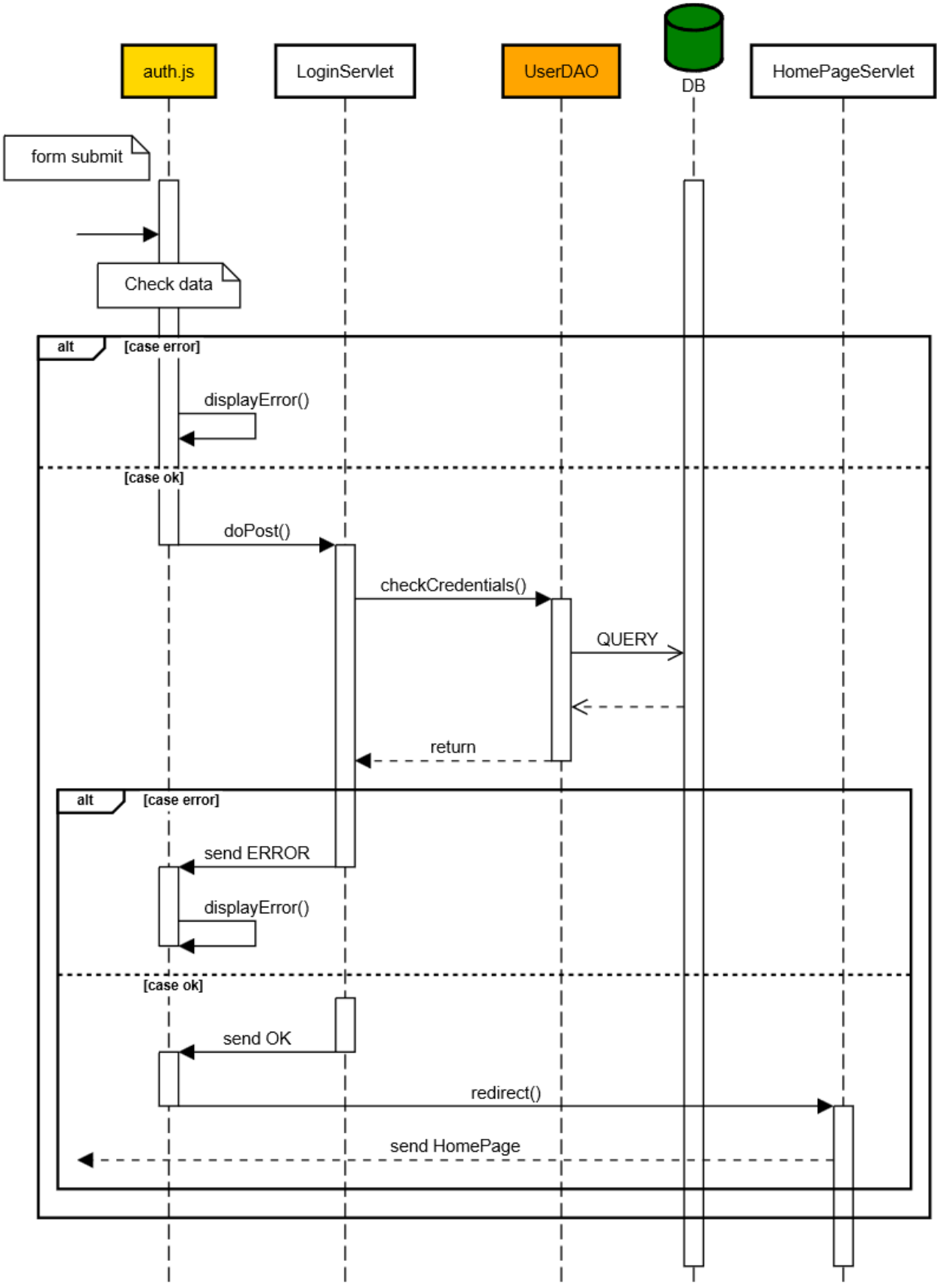
N	Client-side		Server-side	
	Event	Controller (function)	Evento	Controller (Servlet)
1	Login -> login form -> submit	form.eventHandler	POST (username, password)	LoginServlet
2	Login -> button Sign Up	showForm()	-	-
3	SignUp1 -> signUp form -> submit	form.eventHandler	POST (username, password, password2)	SignUpServlet
4	SignUp 2 -> signUp form -> submit	form.eventHandler	POST (username, password, password2, firstName, lastName, address)	SignUpServlet
5	SignUp 1/2 -> button Go Back	showForm()	-	-
6	HomePage -> link Logout	-	GET()	LogOutServlet
7	HomePage -> load	fetchUser()	GET()	GetUserInfoServlet
8	HomePage->load	refreshAvailableItems()	GET()	GetAvailableItemsServlet
9	HomePage -> load	refreshOpenAuctions(), refreshClosedAuctions(), refreshWonAuctions()	GET()	GetOpenAuctionsServlet, GetClosedAuctionsServlet, GetWonAuctionsServlet
10	HomePage->load	refreshVisitedAuctions()	POST(AuctionIds)	GetVisitedAuctionsServlet
11	HomePage -> load	initPillTabBar()	-	-
12	HomePage -> selection Buy/Sell	activateTab() (call the various functions refresh...())	GET() (for refresh)	The various Get...AuctionsServlet (for refresh)
13	HomePage -> create item form -> submit	form.eventHandler (set from initItemCreationForm() on load), which calls validateItemCreation() to check the data, refreshAvailableItems()	POST(itemName, itemDescription, price, image) (item creation) + GET() (refresh)	CreateItemServlet, GetAvailableItemsServlet

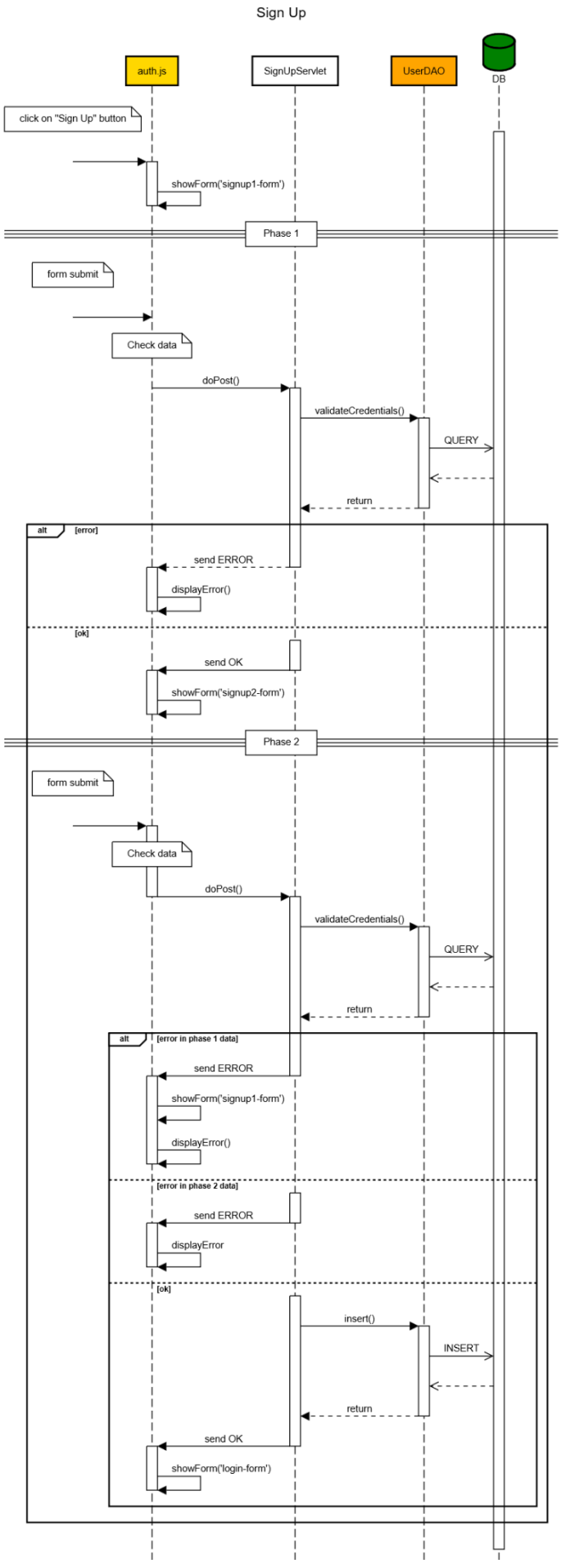
14	HomePage -> create auction form -> submit	form.eventHandler(set from initAuctionCreationForm()), which calls validateAuctionCreationForm() to check the data, refreshOpenAuctions(), refreshAvailableItems()	POST(minIncrement, closingDate, items) (auction creation) + GET() (refresh)	CreateAuctionsServlet, GetOpenAuctionsServlet, GetAvailableItemsServlet
15	HomePage -> selection of item in auction creation form checkbox -> check	-	-	-
16	HomePage -> search form -> submit	form.eventHandler	GET(query)	GetSearchResultsServlet
17	HomePage -> opening auction details (from any list) button -> click	showAuctionPopup(), updateAuctionPopup(), addAuctionToVisited()	GET(id)	GetAuctionDetailsServlet()
18	HomePage -> closing auction details popup button -> click	closePopup()	-	-
19	HomePage -> send offer form -> submit	form.eventHandler (set from initMakeOfferForm()), which calls validateOffer() to check the data, updateAuctionPopup() for refresh	POST(userId, auctionId, offerId) (send offer) + GET(id) (refresh)	MakeOfferServlet, GetAuctionsDetailsServlet
20	HomePage -> closing auction button -> click	button.eventHandler (set from initCloseAuctionButton()), which calls validateCloseAuction() for check, updateAuctionPopup() for refresh	POST(userId, auctionId) (closing) + GET(id) (refresh)	CloseAuctionServlet, GetAuctionsDetailsServlet

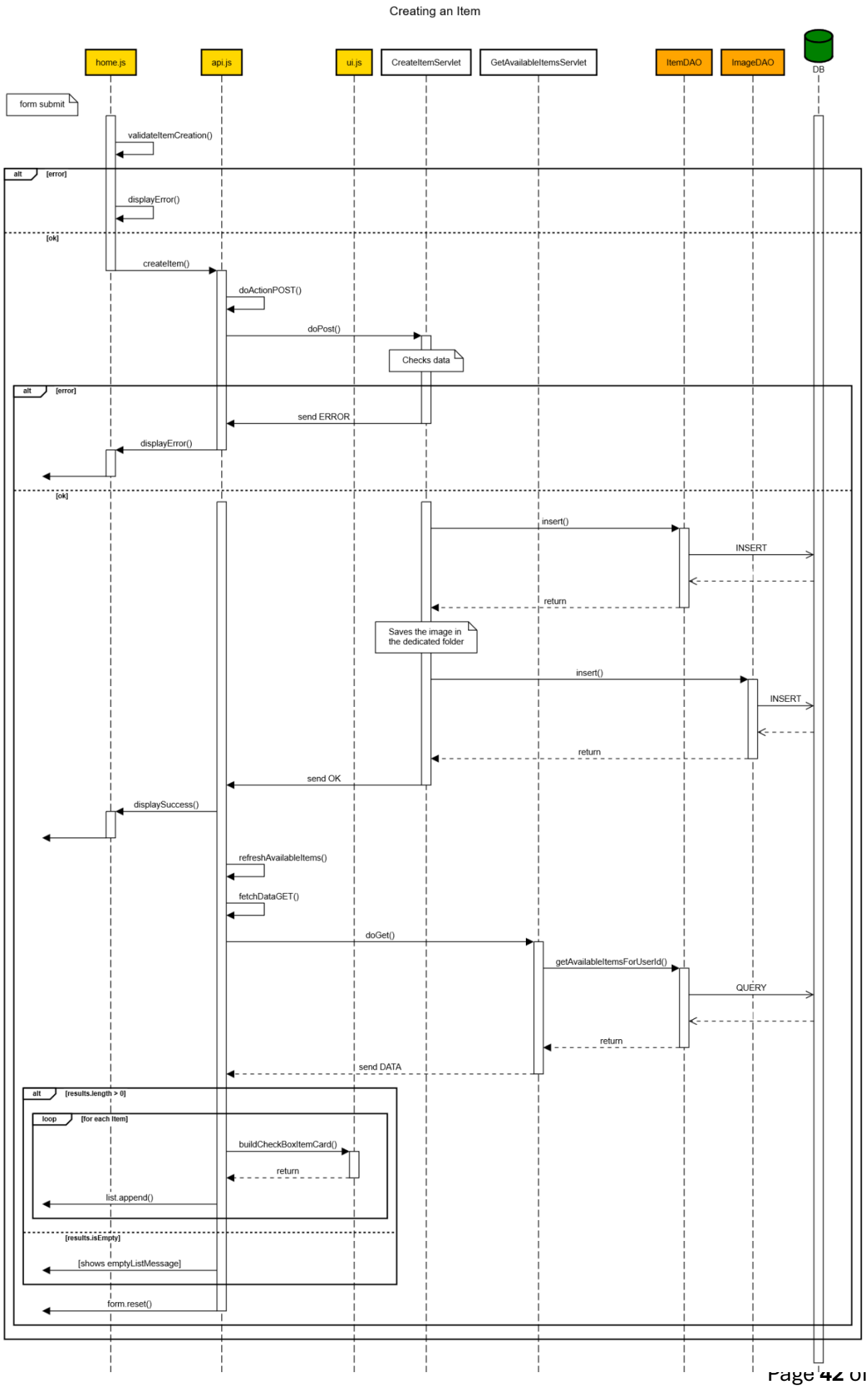
Sequence Diagrams



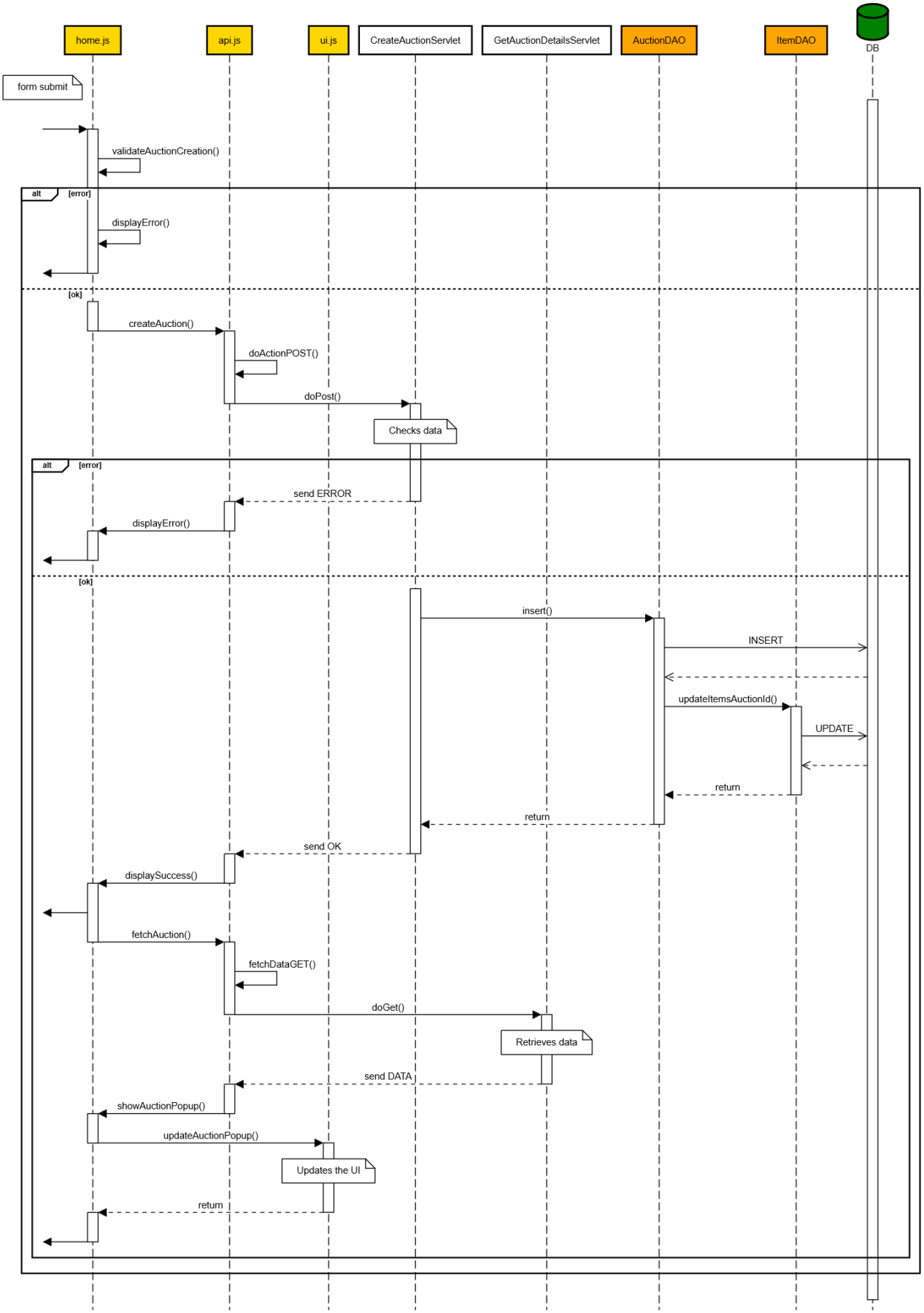
Login


auth.js
LoginServlet
UserDAO
DB
HomePageServlet
form submit
Check data
alt
[case error]
displayError()
[case ok]
doPost()
checkCredentials()
QUERY
return
alt
[case error]
send ERROR
displayError()
[case ok]
send OK
redirect()
send HomePage

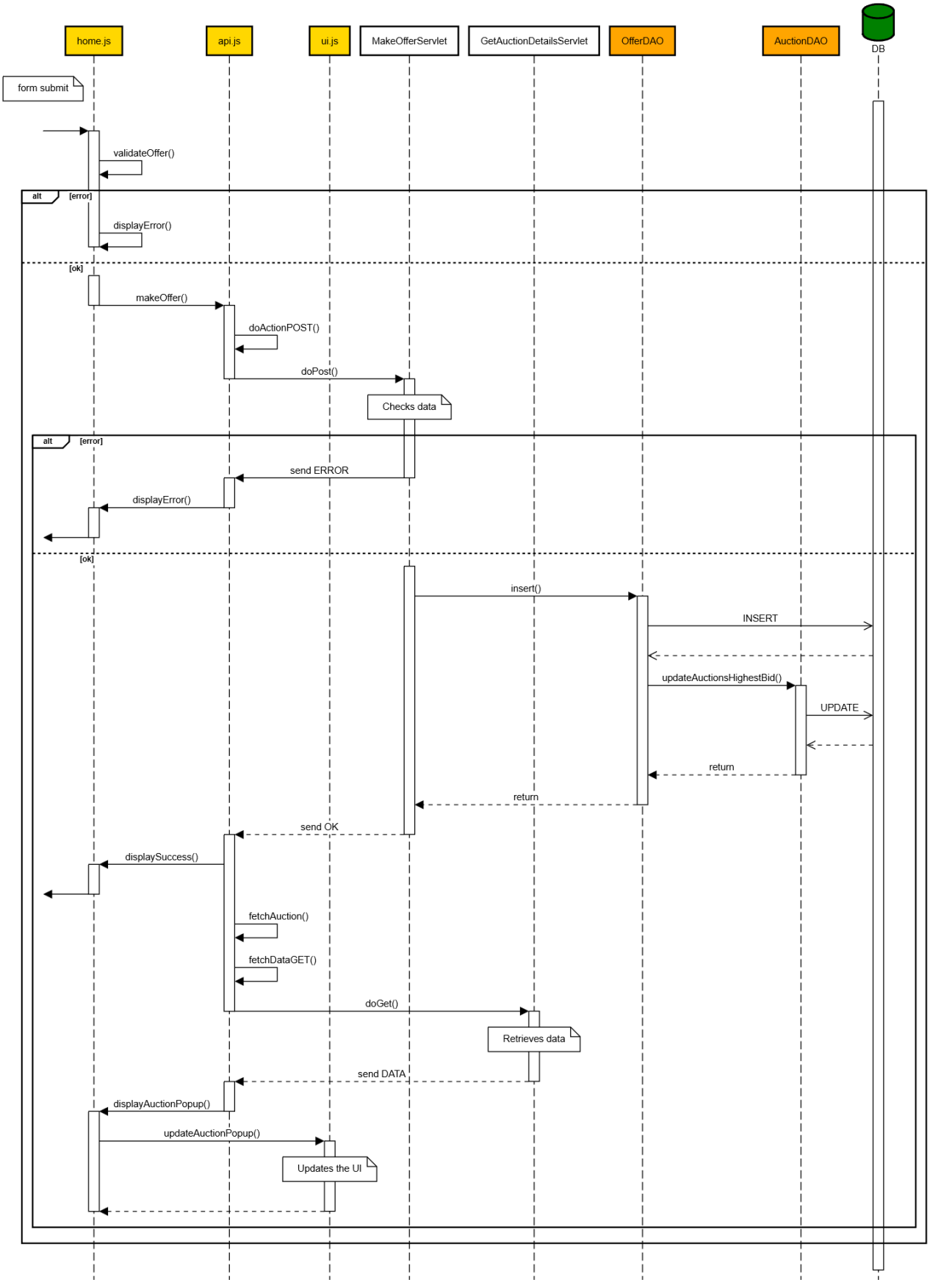

Sign Up
auth.js
SignUpServlet
UserDAO
DB
click on "Sign Up" button
showForm('signup1-form')
Phase 1
form submit
Check data
doPost()
validateCredentials()
QUERY
return
alt
[error]
send ERROR
displayError()
[ok]
send OK
showForm('signup2-form')
Phase 2
form submit
Check data
doPost()
validateCredentials()
QUERY
return
alt
[error in phase 1 data]
send ERROR
showForm('signup1-form')
displayError()
[error in phase 2 data]
send ERROR
displayError
[ok]
insert()
INSERT
return
send OK
showForm('login-form')


Creating an Item
home.js
api.js
ui.js
CreateItemServlet
GetAvailableItemsServlet
ItemDAO
ImageDAO
DB
form submit
validateItemCreation()
alt
[error]
displayError()
[ok]
createItem()
doActionPOST()
doPost()
Checks data
alt
[error]
send ERROR
displayError()
[ok]
insert()
INSERT
return
Saves the image in the dedicated folder
insert()
INSERT
return
send OK
displaySuccess()
refreshAvailableItems()
fetchDataGET()
doGet()
getAvailableItemsForUserId()
QUERY
return
send DATA
alt
[results.length > 0]
loop
[for each Item]
buildCheckBoxItemCard()
return
list.append()
[results.isEmpty]
[shows emptyListMessage]
form.reset()

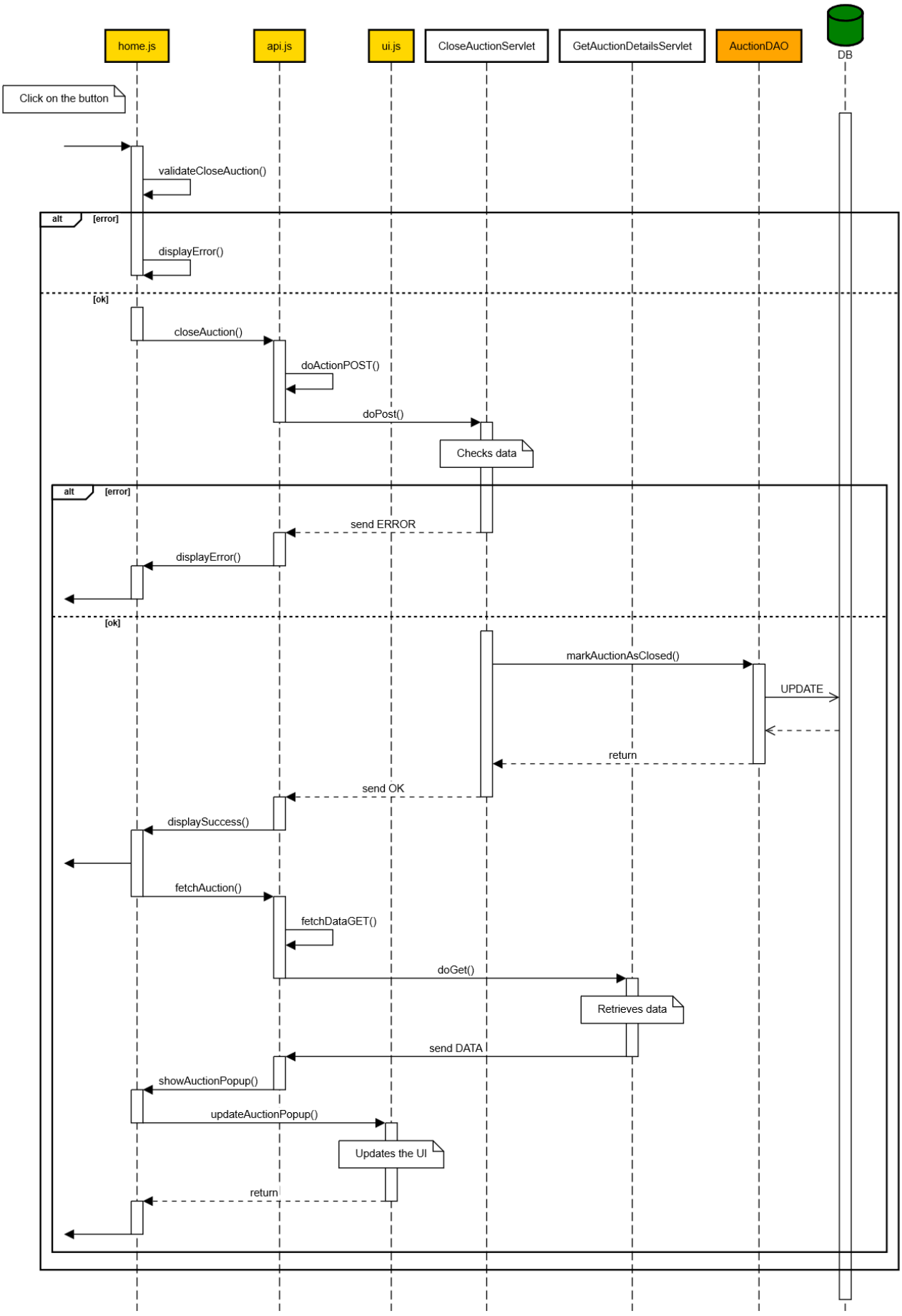
Creating an Auction



Making an Offer



Closing an Auction



View and interface

For this second version of the project, unlike the first one, it was explicitly requested that everything that comes after the Login phase must be handled in a single web page.
For this reason this version implements only two pages: LoginPage and HomePage.

Even though in this version of the project it would have been sufficient to allow direct access to HTML files for the two pages, given that there was no requirement to pre-process pages before sending them to the user anymore (in other words, we could have just exposed the file paths like "/HomePage.html"), we still decided to implement a similar approach to the first version of the project, with two Servlets (LoginPageServlet and HomePageServlet) whose goal is to retrieve the HTML files form the WEB-INF directory and send them to the client.

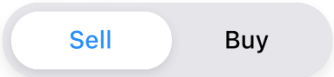
This was done for two reasons:

1. Avoiding exposing the internal structure of the server
2. Give users two comfortable URLs: "/login" and "/home", which we find easier and more intuitive to use.

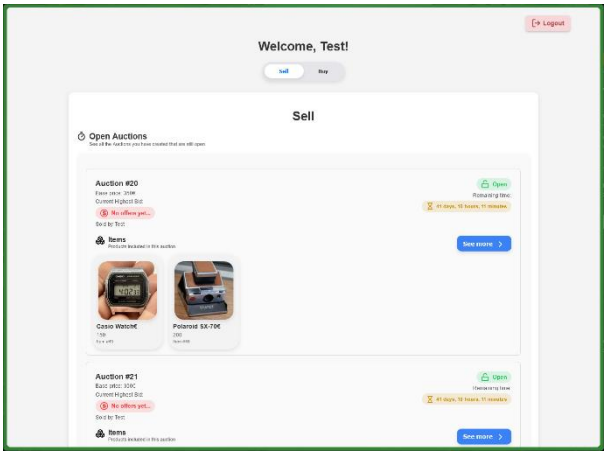
LoginPage handles internally both the login and the signup processes: it contains all the 3 forms that were previously divided into 3 separate pages, and shows and hides them dynamically via JavaScript.

In a similar way, HomePage handles all the roles that were previously split between SellPage, BuyPage and AuctionDetailsPage.
Unlike LoginPage, however, the two main flows of this screen are not exactly exclusive, therefore we didn't find appropriate to "hide" one of the sections behind a simple button at the bottom of the page.

The solution that has been implemented is a TabView, a component mutuated from mobile UIs and vastly used across the web to divide multiple independent buy equally important flows.



This component, located in the top portion of the page and immediately visible, control the appearing and disappearing of two macro-containers *div*, one containing all that previously was inside the SellPage, one containing the BuyPage..

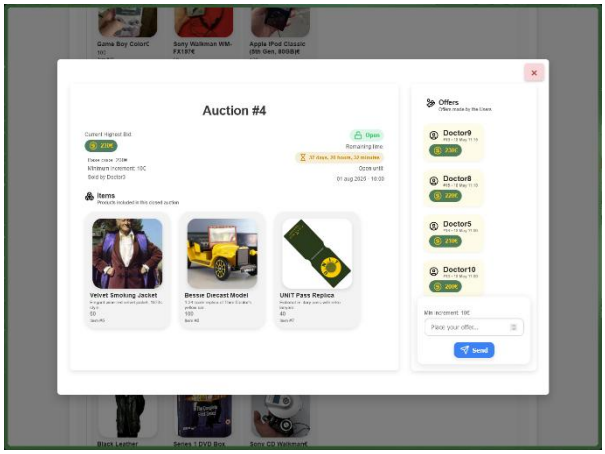


Example of page: SellPage

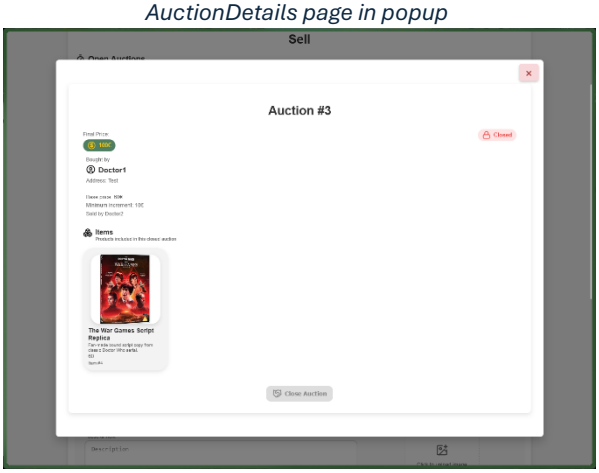
To show the AuctionDetailsPage, instead, we opted for a different solution: instead of showing it as an entire page it was added inside a popup, shown on top of the main view of the HomePage.

This allows to differentiate this particular section, which plays a secondary role in navigation compared to Sell and Buy sections, and also to unify and centralize the logic behind its appearance, refresh and disappearance.

Like the other version, the page shows a panel in the right hand side containing the Offers list, which is rendered onky in specific cases, as requested by specs. In this version, however, the appearance and disappearance of the panel is handled via JavaScrip on the client-side.



AuctionDetails page in popup



AuctionDetails popup without the list of Offers