Documentation: AstaLaVista

Stefano Carletto - 10892219

Esercizio 1

Indice

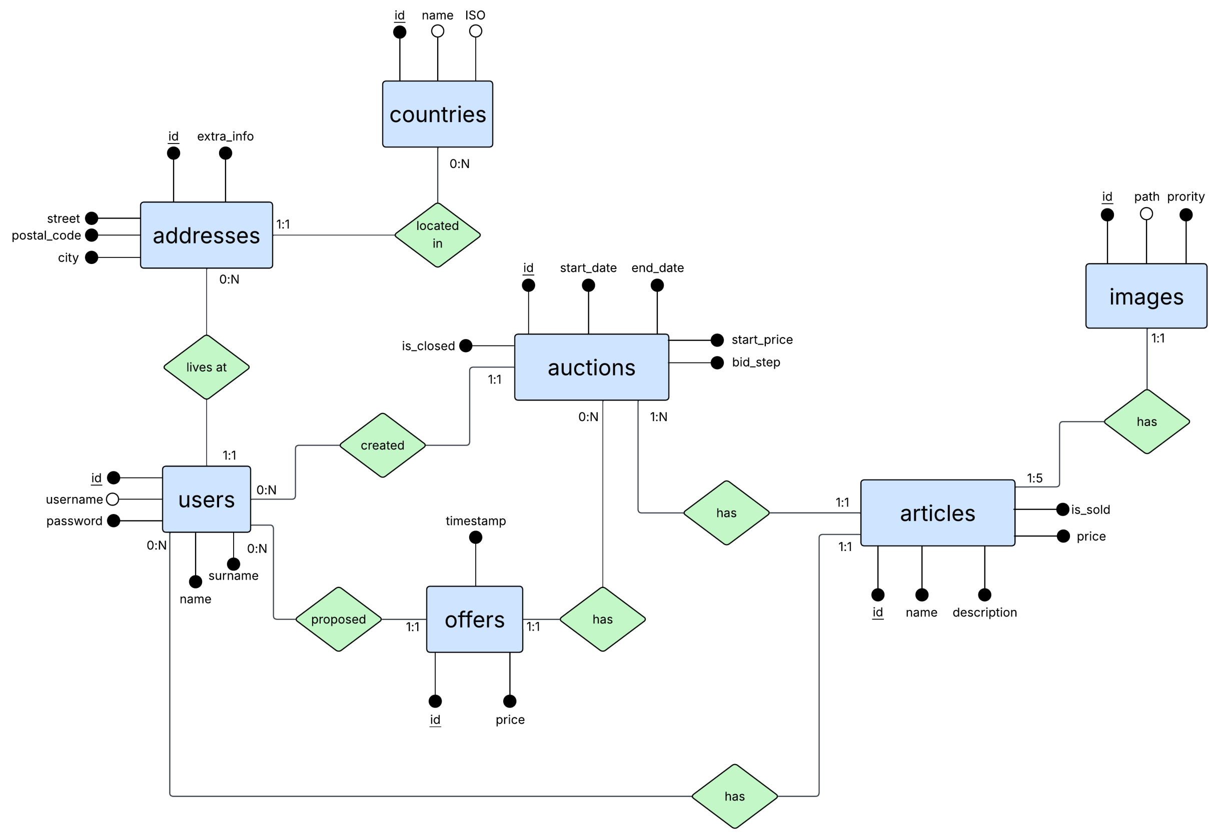
1 Introduction

This project was developed using Spring Boot as the backend framework. The choice of Spring Boot was motivated by its modern and opinionated approach to building web applications, which reduces boilerplate code and speeds up development. Its strong integration with Spring Security, simplified configuration, and production-ready features make it well-suited for scalable and secure applications.

For data persistence, a MySQL relational database was used, providing a reliable and widely supported solution for structured data management.

2 Database

2.1 ER Diagram



2.2 Database DDL (Data Definition Language)

2.2.1 Users Table

Listing 1: Users table

```
CREATE TABLE ‘users‘ (
        ‘id‘ INT(11) NOT NULL AUTO_INCREMENT,
        ‘username‘ VARCHAR(100) NOT NULL COLLATE 'utf8mb4_general_ci',
        ‘password‘ VARCHAR(100) NOT NULL COLLATE 'utf8mb4_general_ci',
        ‘name‘ VARCHAR(100) NOT NULL COLLATE 'utf8mb4_general_ci',
        ‘surname‘ VARCHAR(100) NOT NULL COLLATE 'utf8mb4_general_ci',
        ‘address_id‘ INT(11) NULL DEFAULT NULL,
        PRIMARY KEY (‘id‘) USING BTREE,
        UNIQUE INDEX ‘Users_unique‘ (‘username‘) USING BTREE,
        INDEX ‘FK_users_addresses‘ (‘address_id‘) USING BTREE,
        CONSTRAINT ‘FK_users_addresses‘ FOREIGN KEY (‘address_id‘) REFERENCES ‘
            addresses‘ (‘id‘) ON UPDATE RESTRICT ON DELETE SET NULL
)
```

2.2.2 Addresses Table

Listing 2: Addresses table

```sql
CREATE TABLE ‘addresses‘ (
        ‘id‘ INT(11) NOT NULL AUTO_INCREMENT,
        ‘street‘ VARCHAR(50) NOT NULL DEFAULT ’Via Roma 11’ COLLATE ’utf8mb4_general_ci’,
        ‘postal_code‘ VARCHAR(50) NOT NULL DEFAULT ’22040’ COLLATE ’utf8mb4_general_ci’,
        ‘city‘ VARCHAR(50) NOT NULL DEFAULT ’Alzate Brianza’ COLLATE ’utf8mb4_general_ci’,
        ‘extra_info‘ VARCHAR(50) NULL DEFAULT NULL COLLATE ’utf8mb4_general_ci’,
        ‘country_id‘ INT(11) NULL DEFAULT NULL,
        PRIMARY KEY (‘id‘) USING BTREE,
        INDEX ‘FK_addresses_countries‘ (‘country_id‘) USING BTREE,
        CONSTRAINT ‘FK_addresses_countries‘ FOREIGN KEY (‘country_id‘) REFERENCES ‘countries‘ (‘id‘) ON UPDATE RESTRICT ON DELETE SET NULL
)
```

2.2.3 Countries Table

Listing 3: Countries table

```sql
CREATE TABLE ‘countries‘ (
        ‘id‘ INT(11) NOT NULL AUTO_INCREMENT,
        ‘name‘ CHAR(100) NOT NULL COLLATE ’utf8mb4_general_ci’,
        ‘ISO‘ CHAR(2) NOT NULL COLLATE ’utf8mb4_general_ci’,
        PRIMARY KEY (‘id‘) USING BTREE,
        UNIQUE INDEX ‘name‘ (‘name‘) USING BTREE,
        UNIQUE INDEX ‘ISO‘ (‘ISO‘) USING BTREE
)
```

2.2.4 Articles Table

Listing 4: Articles table

```sql
CREATE TABLE ‘articles‘ (
        ‘id‘ INT(11) NOT NULL AUTO_INCREMENT,
        ‘name‘ VARCHAR(50) NOT NULL DEFAULT ’bomba nucleare’ COLLATE ’utf8mb4_general_ci’,
        ‘description‘ TEXT NOT NULL COLLATE ’utf8mb4_general_ci’,
        ‘price‘ FLOAT NOT NULL DEFAULT ’10’,
        ‘is_sold‘ ENUM(’Y’,’N’) NOT NULL DEFAULT ’N’ COLLATE ’utf8mb4_general_ci’,
        ‘user_id‘ INT(11) NOT NULL DEFAULT ’0’,
        ‘auction_id‘ INT(11) NULL DEFAULT NULL,
        PRIMARY KEY (‘id‘) USING BTREE,
        INDEX ‘FK_articles_users‘ (‘user_id‘) USING BTREE,
        INDEX ‘FK_articles_auctions‘ (‘auction_id‘) USING BTREE,
        CONSTRAINT ‘FK_articles_auctions‘ FOREIGN KEY (‘auction_id‘) REFERENCES ‘auctions‘ (‘id‘) ON UPDATE RESTRICT ON DELETE RESTRICT,
        CONSTRAINT ‘FK_articles_users‘ FOREIGN KEY (‘user_id‘) REFERENCES ‘users‘ (‘id‘) ON UPDATE RESTRICT ON DELETE SET NULL
)
```

2.2.5 Offers Table

Listing 5: Offers table

```
CREATE TABLE `offers` (
        `id` INT(11) NOT NULL AUTO_INCREMENT,
        `price` FLOAT NOT NULL DEFAULT '0',
        `timestamp` DATETIME NOT NULL,
        `user_id` INT(11) NOT NULL DEFAULT '0',
        `auction_id` INT(11) NULL DEFAULT NULL,
        PRIMARY KEY (`id`) USING BTREE,
        INDEX `FK__users` (`user_id`) USING BTREE,
        INDEX `FK_offers_auctions` (`auction_id`) USING BTREE,
        CONSTRAINT `FK__users` FOREIGN KEY (`user_id`) REFERENCES `users` (`id`) ON
            UPDATE RESTRICT ON DELETE RESTRICT,
        CONSTRAINT `FK_offers_auctions` FOREIGN KEY (`auction_id`) REFERENCES `
            auctions` (`id`) ON UPDATE RESTRICT ON DELETE CASCADE
)
```

2.2.6 Auctions Table

Listing 6: Auctions table

```
CREATE TABLE `auctions` (
        `id` INT(11) NOT NULL AUTO_INCREMENT,
        `start_date` DATETIME NULL DEFAULT NULL,
        `end_date` DATETIME NULL DEFAULT NULL,
        `start_price` FLOAT NULL DEFAULT '0',
        `bid_step` INT(11) NULL DEFAULT '1',
        `is_closed` ENUM('Y','N') NULL DEFAULT 'N' COLLATE 'utf8mb4_general_ci',
        `user_id` INT(11) NULL DEFAULT NULL,
        PRIMARY KEY (`id`) USING BTREE,
        INDEX `FK_auctions_users` (`user_id`) USING BTREE,
        CONSTRAINT `FK_auctions_users` FOREIGN KEY (`user_id`) REFERENCES `users` (`id
            `) ON UPDATE RESTRICT ON DELETE SET NULL
)
```

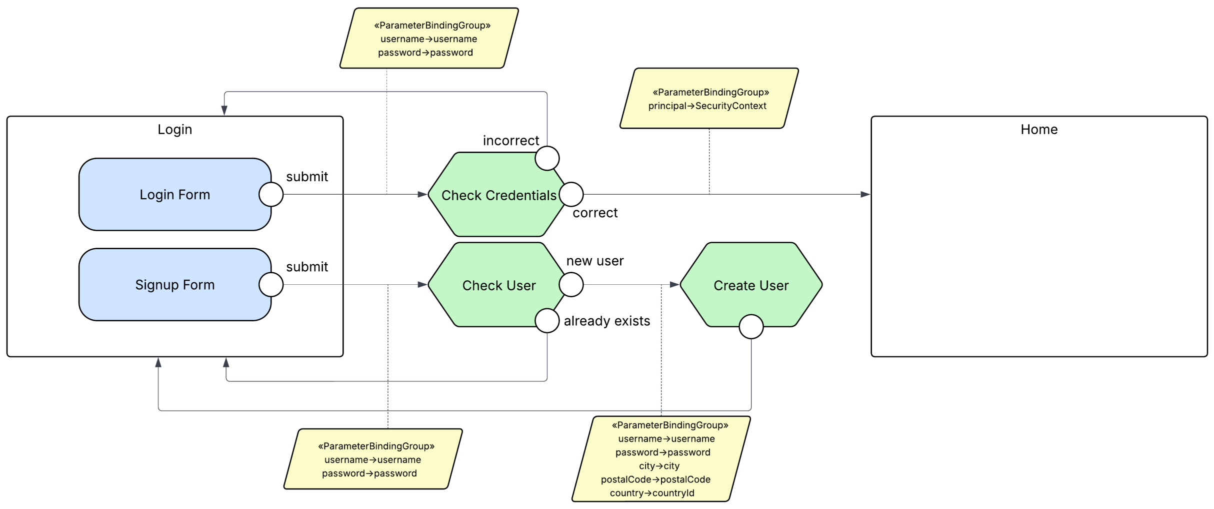
2.2.7 Images Table

Listing 7: Auctions table

```
CREATE TABLE `images` (
        `id` INT(11) NOT NULL AUTO_INCREMENT,
        `path` TEXT NULL DEFAULT NULL COLLATE 'utf8mb4_general_ci',
        `priority` TINYINT(4) NOT NULL DEFAULT '0',
        `article_id` INT(11) NOT NULL DEFAULT '0',
        PRIMARY KEY (`id`) USING BTREE,
        INDEX `path` (`path`(768)) USING BTREE,
        INDEX `FK__articles` (`article_id`) USING BTREE,
        CONSTRAINT `FK__articles` FOREIGN KEY (`article_id`) REFERENCES `articles` (`
            id`) ON UPDATE RESTRICT ON DELETE CASCADE
)
```

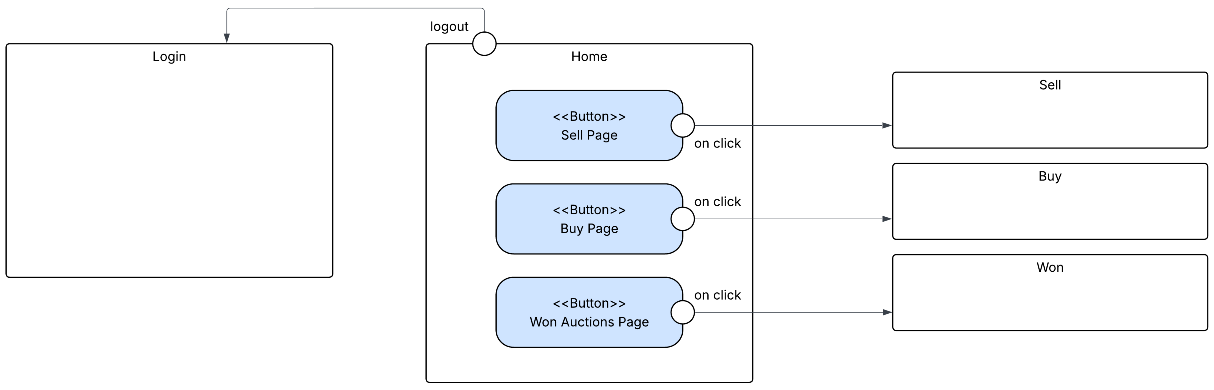
3 Application Design

3.1 Login and Register

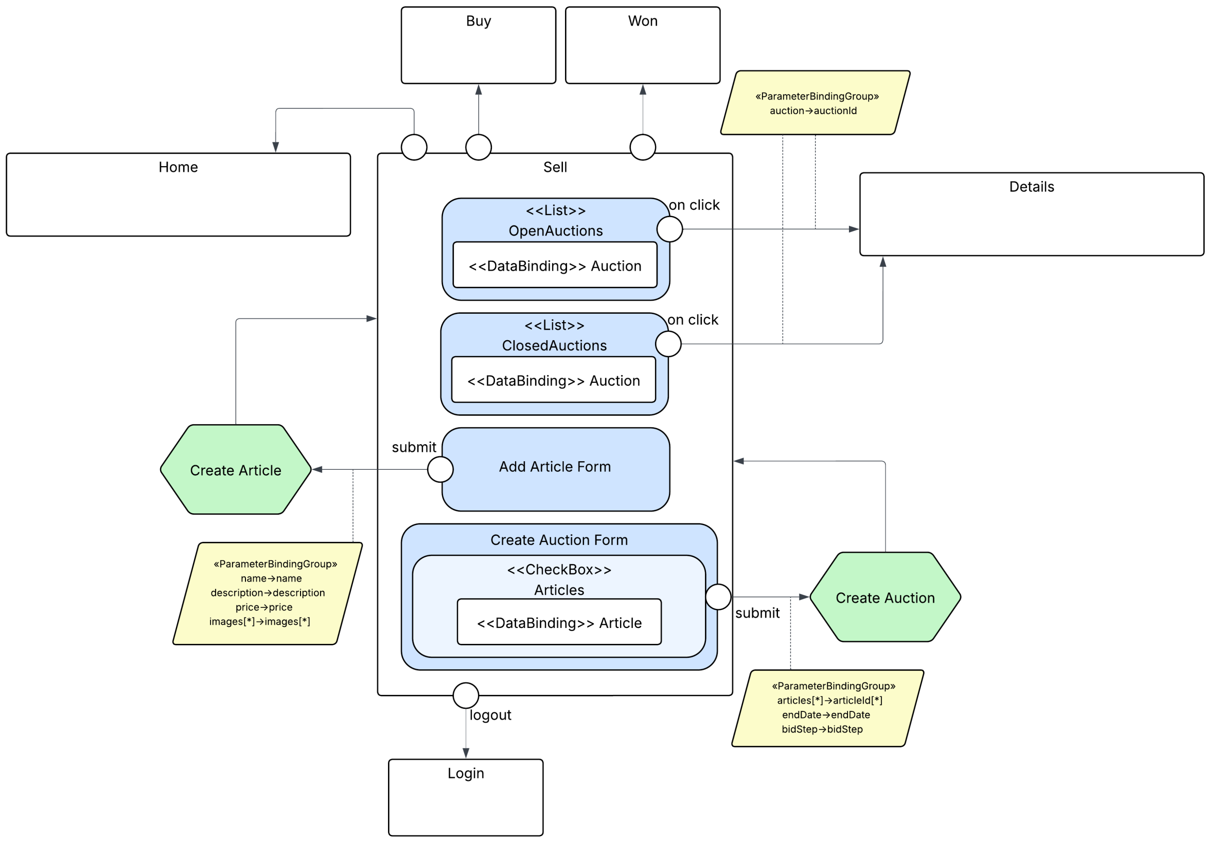


*On successful login, the authenticated user (principal) is stored in the Spring Security **SecurityContext**, which is bound to the **HttpSession**. This allows subsequent requests to access the user identity through the **@AuthenticationPrincipal** annotation or the **SecurityContextHolder**.*

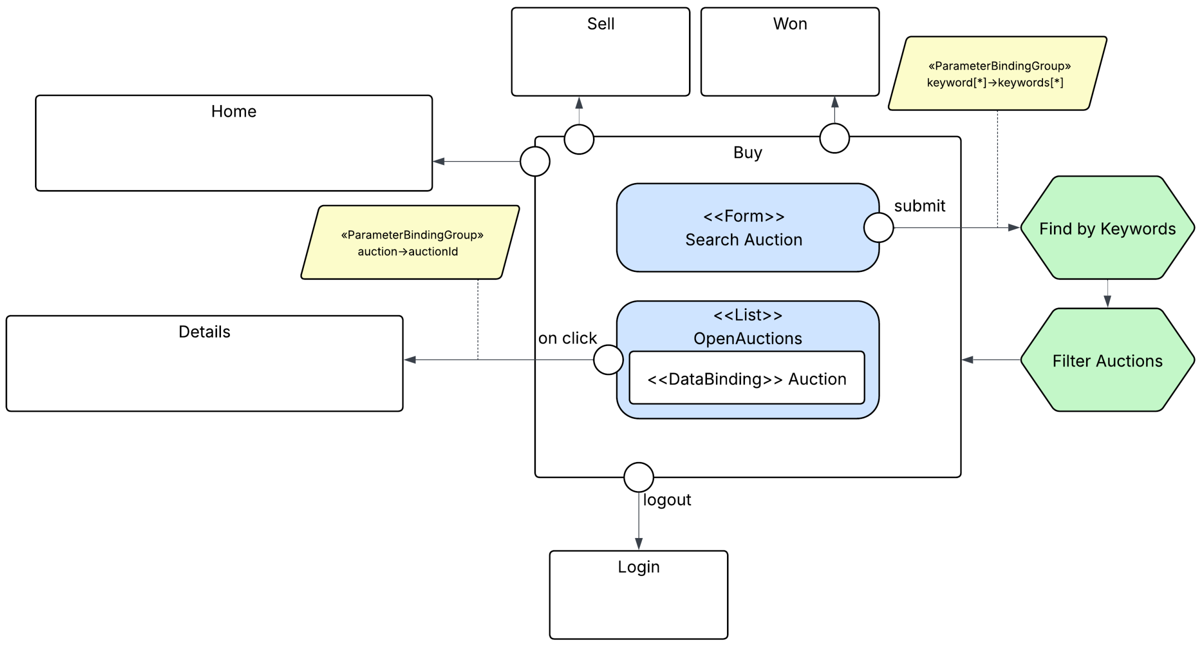
3.2 Home



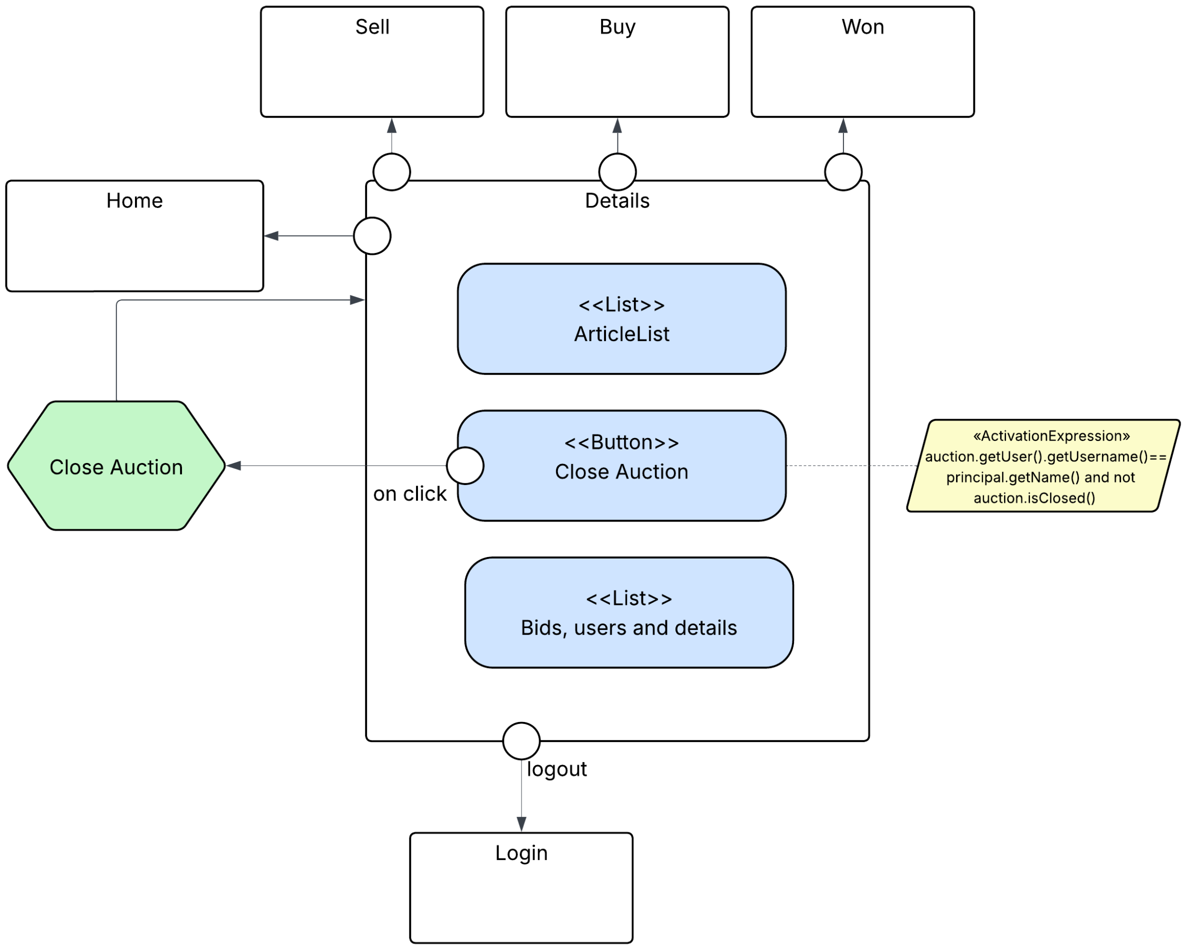
3.3 Sell


Buy
Won
«ParameterBindingGroup»
auction→auctionId
Home
Sell
Details
<<List>>
OpenAuctions
<<DataBinding>> Auction
on click
<<List>>
ClosedAuctions
<<DataBinding>> Auction
on click
submit
Add Article Form
Create Article
«ParameterBindingGroup»
name→name
description→description
price→price
images[*]→images[*]
Create Auction Form
<<CheckBox>>
Articles
<<DataBinding>> Article
submit
Create Auction
«ParameterBindingGroup»
articles[*]→articleId[*]
endDate→endDate
bidStep→bidStep
logout
Login

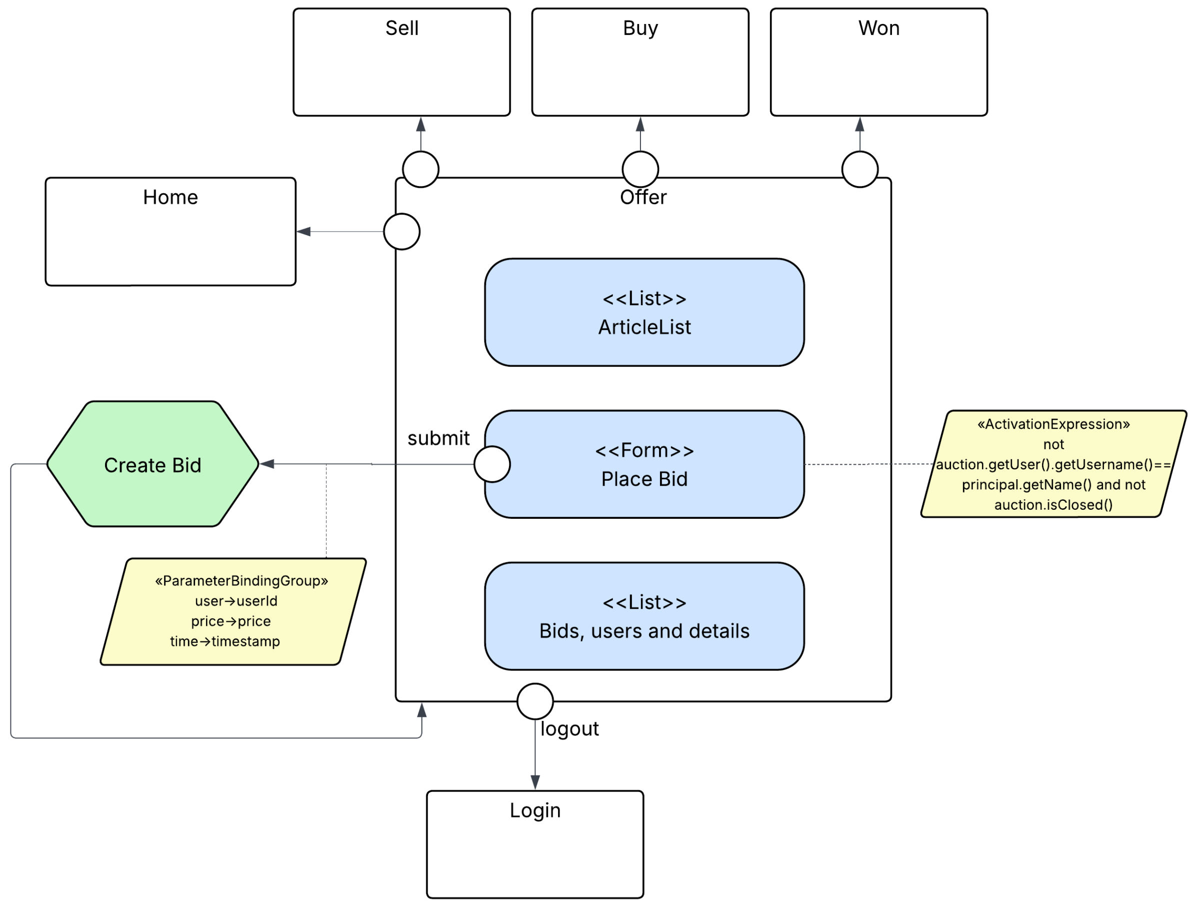
3.4 Buy


Sell
Won
«ParameterBindingGroup»
keyword[*]→keywords[*]
Home
Buy
<<Form>>
Search Auction
submit
Find by Keywords
«ParameterBindingGroup»
auction→auctionId
Details
on click
<<List>>
OpenAuctions
<<DataBinding>> Auction
Filter Auctions
logout
Login

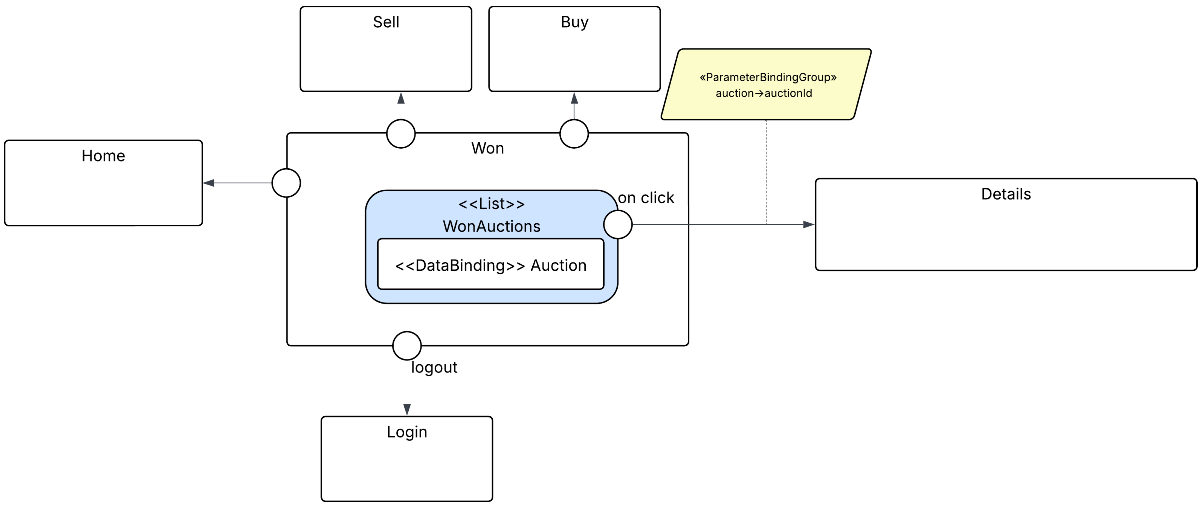
3.5 Details


Sell
Buy
Won
Home
Details
<<List>>
ArticleList
<<Button>>
Close Auction
on click
Close Auction
«ActivationExpression»
auction.getUser().getUsername()==
principal.getName() and not
auction.isClosed()
<<List>>
Bids, users and details
logout
Login

3.6 Offer


Sell
Buy
Won
Home
Offer
<<List>>
ArticleList
Create Bid
submit
<<Form>>
Place Bid
«ActivationExpression»
not
auction.getUser().getUsername()==
principal.getName() and not
auction.isClosed()
«ParameterBindingGroup»
user→userId
price→price
time→timestamp
<<List>>
Bids, users and details
logout
Login

3.7 Won



4 Components

4.1 Model Objects

- `Address`
- `Article`
- `Auction`
- `Country`
- `Image`
- `Offer`
- `User`

4.2 Repositories

- `AddressRepository`
- `ArticleRepository`
- `AuctionRepository`
- `CountryRepository`
- `ImageRepository`
- `OfferRepository`
- `UserRepository`

4.3 Controllers

- `BuyController`
- `DetailsController`
- `HomeController`
- `LoginController`
- `OfferController`
- `SellController`
- `UserController`
- `WonAuctionsController`

4.4 Services

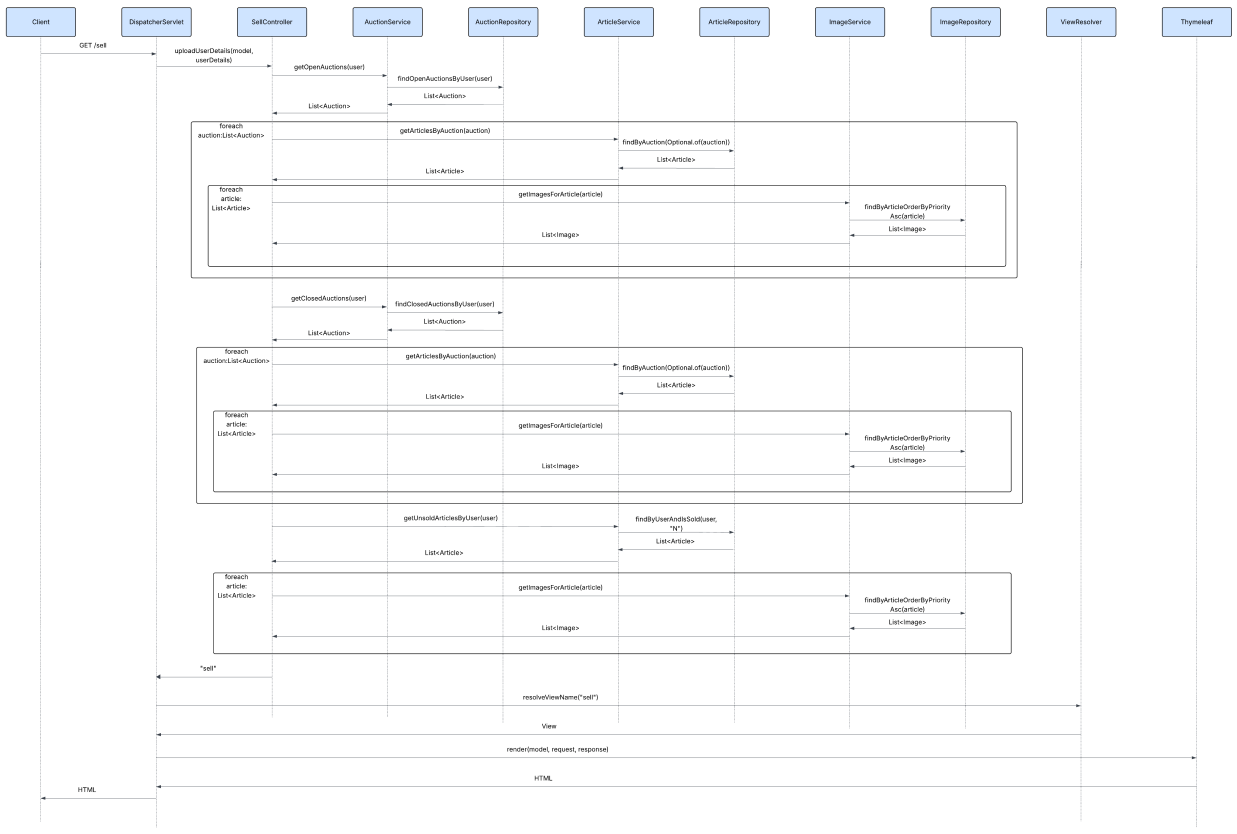
- `ArticleService`
- `AuctionService`
- `CountryService`
- `CustomUserDetailsService`
- `ImageService`
- `LoginService`
- `OfferService`
- `UserService`

5 Events: Sequence Diagrams

5.1 SPA: general case

In SPA mode, the server no longer returns server-rendered HTML but JSON; the UI is updated on the client. Routes like `GET /page` become `GET /api/....` `POST` requests do not issue server redirects; they reply with 200/201, and the client-side router changes the view without a full page reload. Authentication still uses the session cookie; the client calls `GET /api/users/me` at bootstrap.

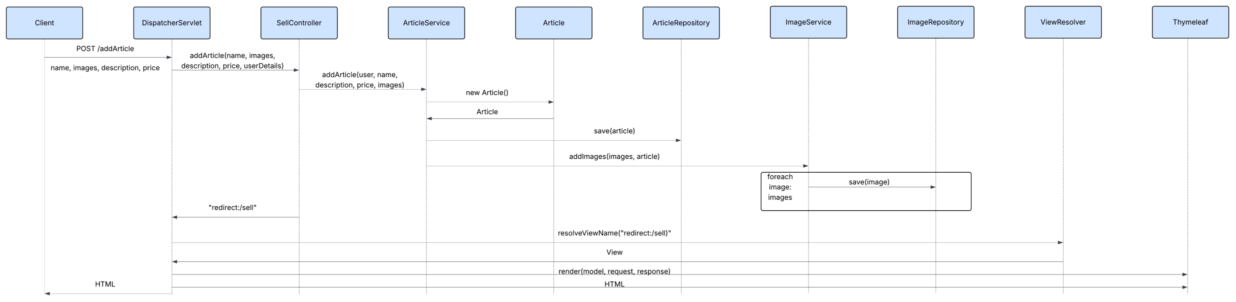
5.2 Sell: loading user details


Client
DispatcherServlet
SellController
AuctionService
AuctionRepository
ArticleService
ArticleRepository
ImageService
ImageRepository
ViewResolver
Thymeleaf
GET /sell
uploadUserDetails(model, userDetails)
getOpenAuctions(user)
findOpenAuctionsByUser(user)
List<Auction>
List<Auction>
foreach auction:List<Auction>
getArticlesByAuction(auction)
findByAuction(Optional.of(auction))
List<Article>
List<Article>
foreach article: List<Article>
getImagesForArticle(article)
findByArticleOrderByPriority Asc(article)
List<Image>
List<Image>
getClosedAuctions(user)
findClosedAuctionsByUser(user)
List<Auction>
List<Auction>
foreach auction:List<Auction>
getArticlesByAuction(auction)
findByAuction(Optional.of(auction))
List<Article>
List<Article>
foreach article: List<Article>
getImagesForArticle(article)
findByArticleOrderByPriority Asc(article)
List<Image>
List<Image>
getUnsoldArticlesByUser(user)
findByUserAndIsSold(user, "N")
List<Article>
List<Article>
foreach article: List<Article>
getImagesForArticle(article)
findByArticleOrderByPriority Asc(article)
List<Image>
List<Image>
"sell"
resolveViewName("sell")
View
render(model, request, response)
HTML
HTML

5.2.1 SPA case

`GET /sell` no longer used; the SPA view mounts and calls `GET /api/users/me`

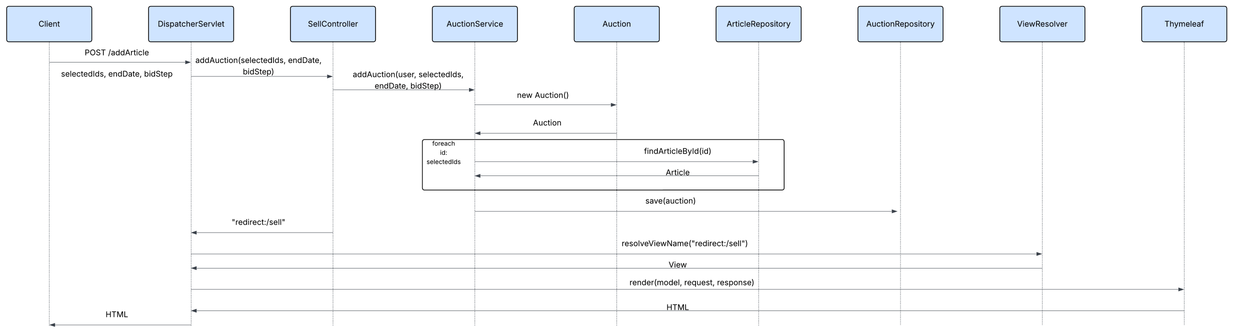
5.3 Sell: uploading new article



5.3.1 SPA case

`201 JSON` with the created article; no `redirect:/sell`.

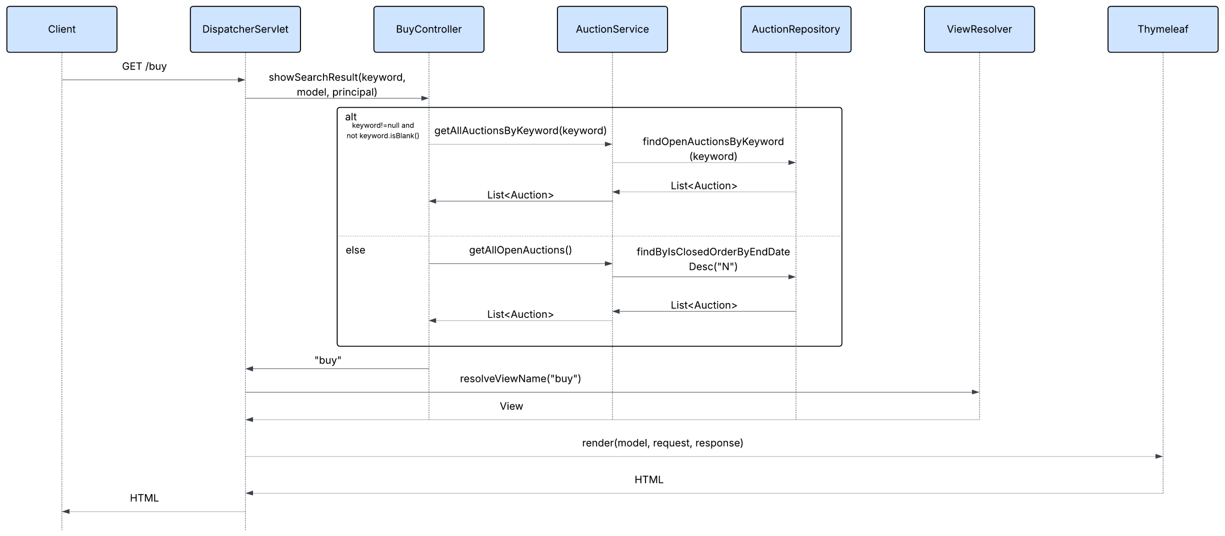
5.4 Sell: creating new auction



5.4.1 SPA case

`201 JSON` with the created article; no `redirect:/sell`.

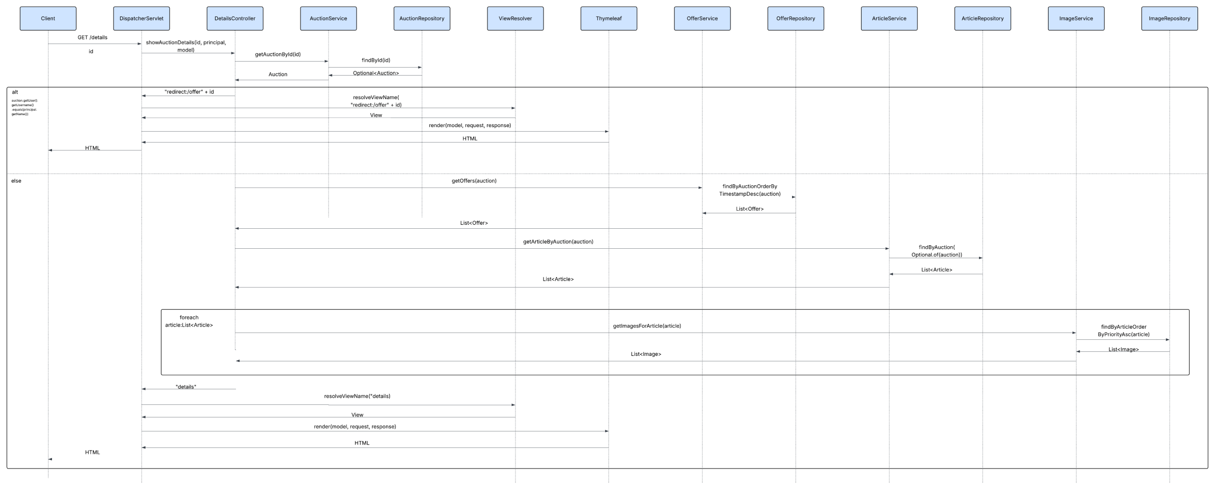
5.5 Buy: loading auctions and keyword search


Client
DispatcherServlet
BuyController
AuctionService
AuctionRepository
ViewResolver
Thymeleaf
GET /buy
showSearchResult(keyword, model, principal)
alt
keyword!=null and not keyword.isBlank()
getAllAuctionsByKeyword(keyword)
findOpenAuctionsByKeyword (keyword)
List<Auction>
List<Auction>
else
getAllOpenAuctions()
findByIsClosedOrderByEndDate Desc("N")
List<Auction>
List<Auction>
"buy"
resolveViewName("buy")
View
render(model, request, response)
HTML
HTML

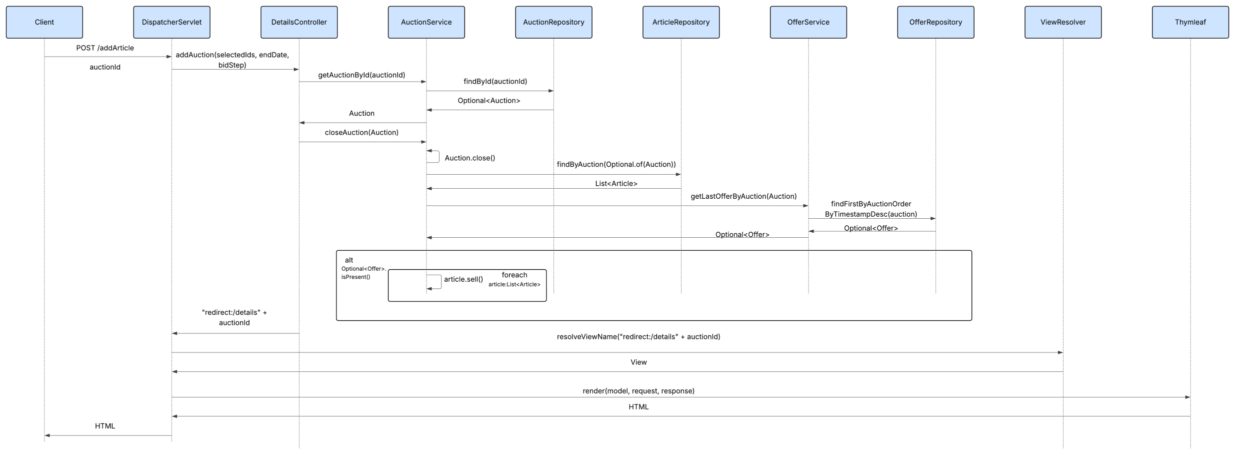
5.5.1 SPA case

No full page reload: `GET /api/auctions?...`

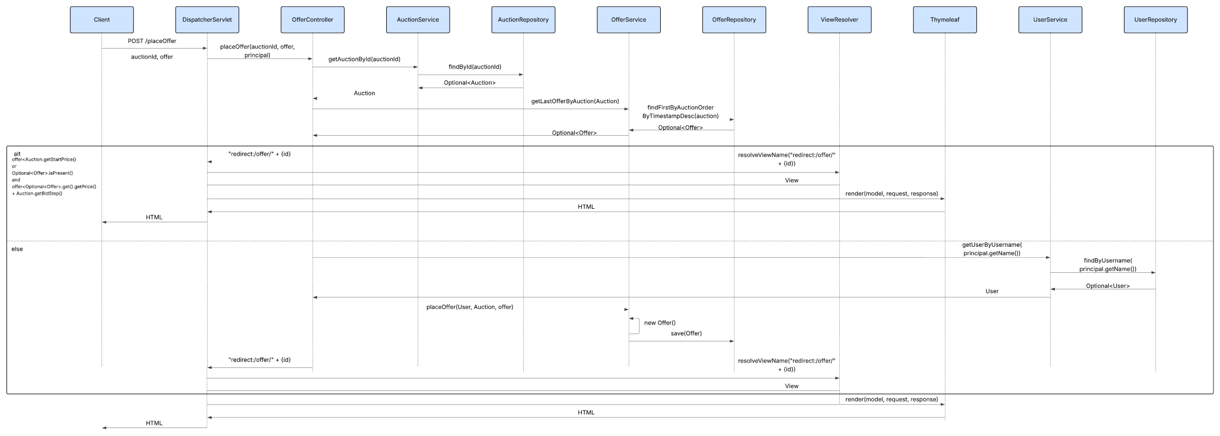
5.6 Details: loading auction details


Client
DispatcherServlet
DetailsController
AuctionService
AuctionRepository
ViewResolver
Thymeleaf
OfferService
OfferRepository
ArticleService
ArticleRepository
ImageService
ImageRepository
GET /details
id
showAuctionDetails(id, principal, model)
getAuctionById(id)
findById(id)
Optional<Auction>
Auction
alt
auction.getUser().getUsername().equals(principal.getName())
"redirect:/offer" + id
resolveViewName("redirect:/offer" + id)
View
render(model, request, response)
HTML
HTML
else
getOffers(auction)
findByAuctionOrderByTimestampDesc(auction)
List<Offer>
List<Offer>
getArticleByAuction(auction)
findByAuction(Optional.of(auction))
List<Article>
List<Article>
foreach article:List<Article>
getImagesForArticle(article)
findByArticleOrderByPriorityAsc(article)
List<Image>
List<Image>
"details"
resolveViewName("details)
View
render(model, request, response)
HTML
HTML

5.7 Details: closing auction


Client
DispatcherServlet
DetailsController
AuctionService
AuctionRepository
ArticleRepository
OfferService
OfferRepository
ViewResolver
Thymeleaf
POST /addArticle
auctionId
addAuction(selectedIds, endDate, bidStep)
getAuctionById(auctionId)
findById(auctionId)
Optional<Auction>
Auction
closeAuction(Auction)
Auction.close()
findByAuction(Optional.of(Auction))
List<Article>
getLastOfferByAuction(Auction)
findFirstByAuctionOrderByTimestampDesc(auction)
Optional<Offer>
Optional<Offer>
alt
Optional<Offer>.isPresent()
foreach article:List<Article>
article.sell()
"redirect:/details" + auctionId
resolveViewName("redirect:/details" + auctionId)
View
render(model, request, response)
HTML
HTML

5.8 Offer: place offer


Client
DispatcherServlet
OfferController
AuctionService
AuctionRepository
OfferService
OfferRepository
ViewResolver
Thymeleaf
UserService
UserRepository
POST /placeOffer
auctionId, offer
placeOffer(auctionId, offer, principal)
getAuctionById(auctionId)
findById(auctionId)
Optional<Auction>
Auction
getLastOfferByAuction(Auction)
findFirstByAuctionOrder ByTimestampDesc(auction)
Optional<Offer>
Optional<Offer>
alt
offer<Auction.getStartPrice()
or
Optional<Offer>.isPresent()
and
offer<Optional<Offer>.get().getPrice()
+ Auction.getBidStep()
"redirect:/offer/" + {id}
resolveViewName("redirect:/offer/" + {id})
View
render(model, request, response)
HTML
HTML
else
getUserByUsername(principal.getName())
findByUsername(principal.getName())
Optional<User>
User
placeOffer(User, Auction, offer)
new Offer()
save(Offer)
"redirect:/offer/" + {id}
resolveViewName("redirect:/offer/" + {id})
View
render(model, request, response)
HTML
HTML

6 Security Model

- **Session**: `JSESSIONID` with Spring Security (same-origin).
- **CSRF**: `CookieCsrfTokenRepository` issues an `XSRF-TOKEN`; the client sends it back as `X-XSRF-TOKEN` on `POST/PATCH/DELETE`.
- **CORS**: not required in same-origin.
- **Passwords**: uses PasswordEncoder (`BCryptPasswordEncoder`) to securely hash and store passwords in the database.