PROGETTO TIW 2020/21

Marco D'Antini – Alfredo Landi – Luigi Piccoli

Esercizio 1: Aste online

Specification

Legenda: Entities, Attributes, pages, components, events, action

- Un'applicazione web consente la gestione di aste online. Gli utenti (username, password, indirizzo) accedono tramite login e possono vendere e acquistare all'asta. La HOME page contiene due link, uno per accedere alla pagina VENDO e uno per accedere alla pagina ACQUISTO.

- La pagina VENDO mostra una lista delle aste create dall'utente e non ancora chiuse, una lista delle aste da lui create e chiuse e una form per creare un nuovo articolo e una nuova asta per venderlo. L'asta comprende l'articolo da mettere in vendita (codice, nome, descrizione, immagine), prezzo iniziale, rialzo minimo di ogni offerta e una scadenza (data e ora, es 19-04-2021 alle 23:00). La lista delle aste è ordinata per data+ora crescente e riporta: codice e nome dell'articolo, offerta massima, tempo mancante (numero di giorni e ore) tra il momento del login e la data e ora di chiusura dell'asta. Cliccando su un'asta compare una pagina DETTAGLIO ASTA che riporta per un'asta aperta i dati dell'asta e la lista delle offerte (nome utente, prezzo offerto, data e ora dell'offerta) ordinata per data+ora decrescente. Un bottone CHIUDI permette all'utente di chiudere l'asta se è giunta l'ora della scadenza (si ignori il caso di aste scadute ma non chiuse). Se l'asta è chiusa, la pagina riporta i dati dell'asta, il nome dell'aggiudicatario, il prezzo finale e l'indirizzo di spedizione dell'utente.

- La pagina ACQUISTO contiene una form di ricerca per parola chiave. Quando l'acquirente invia una parola chiave la pagina ACQUISTO si ricarica e mostra un elenco di aste aperte (la cui scadenza è posteriore all'ora dell'invio) il cui articolo contiene la parola chiave nel nome o nella descrizione. La lista è ordinata in modo decrescente in base al tempo (numero di giorni. ore e minuti) mancante alla chiusura. Cliccando su un'asta aperta compare la pagina OFFERTA che mostra i dati dell'articolo, l'elenco delle offerte pervenute in ordine di data+ora decrescente e un campo di input per inserire la propria offerta, che deve essere superiore all'offerta massima corrente. Dopo l'invio dell'offerta la pagina OFFERTA mostra l'elenco delle offerte aggiornate. La pagina ACQUISTO contiene anche un elenco delle offerte aggiudicate all'utente con i dati dell'articolo e il prezzo finale.

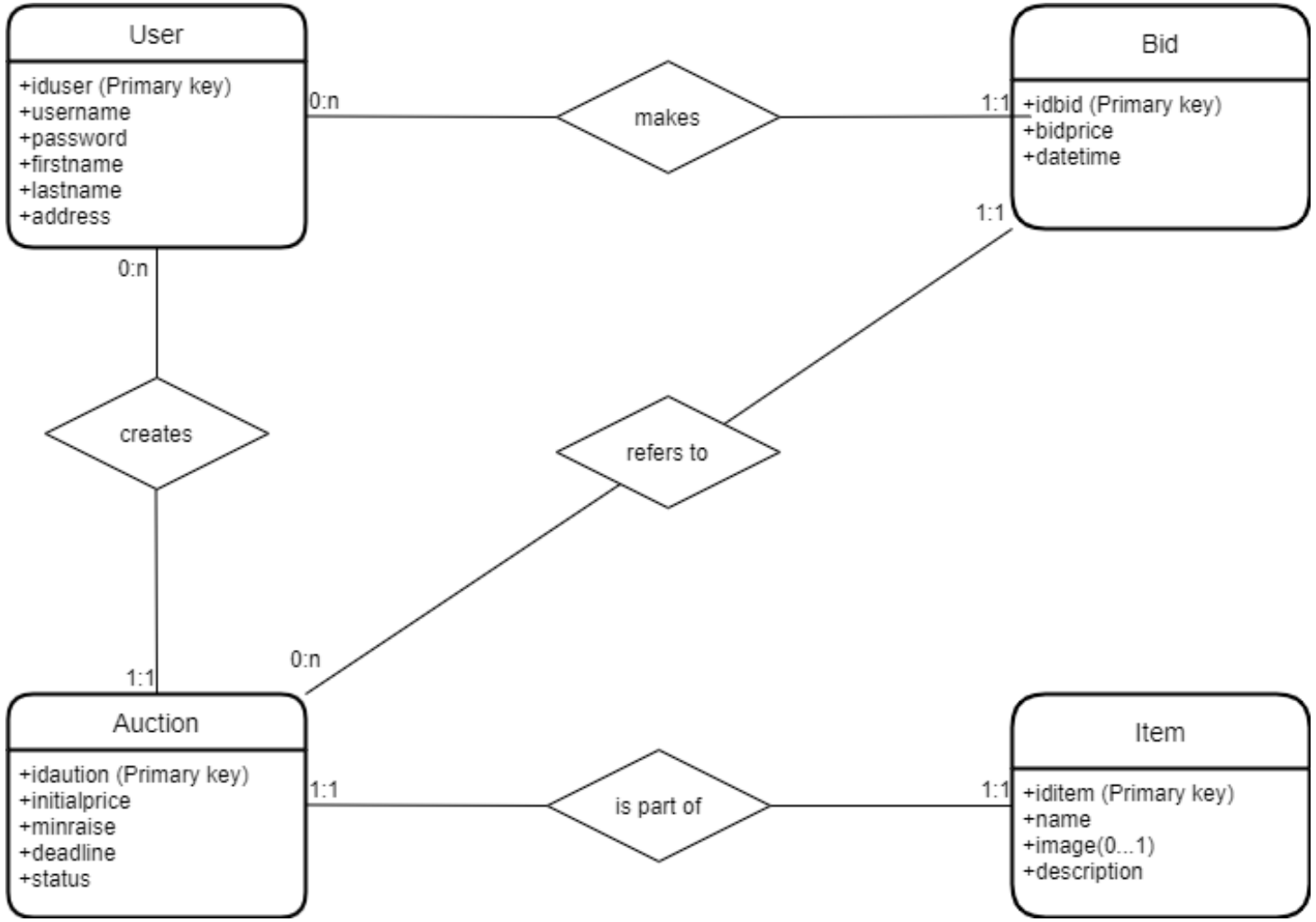
JavaScript extra specifications

- Dopo il login, l'intera applicazione è realizzata con un'unica pagina.
- Se l'utente accede per la prima volta l'applicazione mostra il contenuto della pagina ACQUISTO. Se l'utente ha già usato l'applicazione, questa mostra il contenuto della pagina VENDO se l'ultima azione dell'utente è stata la creazione di un'asta; altrimenti mostra il contenuto della pagina ACQUISTO con l'elenco (eventualmente vuoto) delle aste su cui l'utente ha cliccato in precedenza e che sono ancora aperte. L'informazione dell'ultima azione compiuta e delle aste visitate è memorizzata a lato client per la durata di un mese.
- Ogni interazione dell'utente è gestita senza ricaricare completamente la pagina, ma produce l'invocazione asincrona del server e l'eventuale modifica solo del contenuto da aggiornare a seguito dell'evento

Specification completion

- When creating an auction:
  - The initial price must be set between 1€ and 999999,99€.
  - The minimum raise must be set between 0.01€ and 500000€.
  - The deadline for the auction must be set between 24h and 2 weeks from the insertion.

Database design



Local/Remote database schema

```
CREATE TABLE `auction` (
  `idauction` int NOT NULL AUTO_INCREMENT,
  `initialprice` decimal(10,2) NOT NULL,
  `minraise` decimal(10,2) NOT NULL,
  `deadline` timestamp NOT NULL,
  `idcreator` int NOT NULL,
  `iditem` int NOT NULL,
  `status` int NOT NULL DEFAULT '0',
  PRIMARY KEY (`idauction`),
  KEY `idcreator_idx` (`idcreator`),
  KEY `iditem_idx` (`iditem`),
  CONSTRAINT `idcreator` FOREIGN KEY (`idcreator`) REFERENCES `user`
(`iduser`),
  CONSTRAINT `iditem` FOREIGN KEY (`iditem`) REFERENCES `item` (`iditem`)
```

Local/Remote database schema

```
CREATE TABLE `bid` (
  `idbid` int NOT NULL AUTO_INCREMENT,
  `bidprice` decimal(10,2) NOT NULL,
  `datetime` timestamp NOT NULL,
  `idbidder` int NOT NULL,
  `idauction` int NOT NULL,
  PRIMARY KEY (`idbid`),
  KEY `idbidder_idx` (`idbidder`),
  KEY `idauction_idx` (`idauction`),
  CONSTRAINT `idauction` FOREIGN KEY (`idauction`) REFERENCES `auction`
(`idauction`),
  CONSTRAINT `idbidder` FOREIGN KEY (`idbidder`) REFERENCES `user`
(`iduser`)
)
```

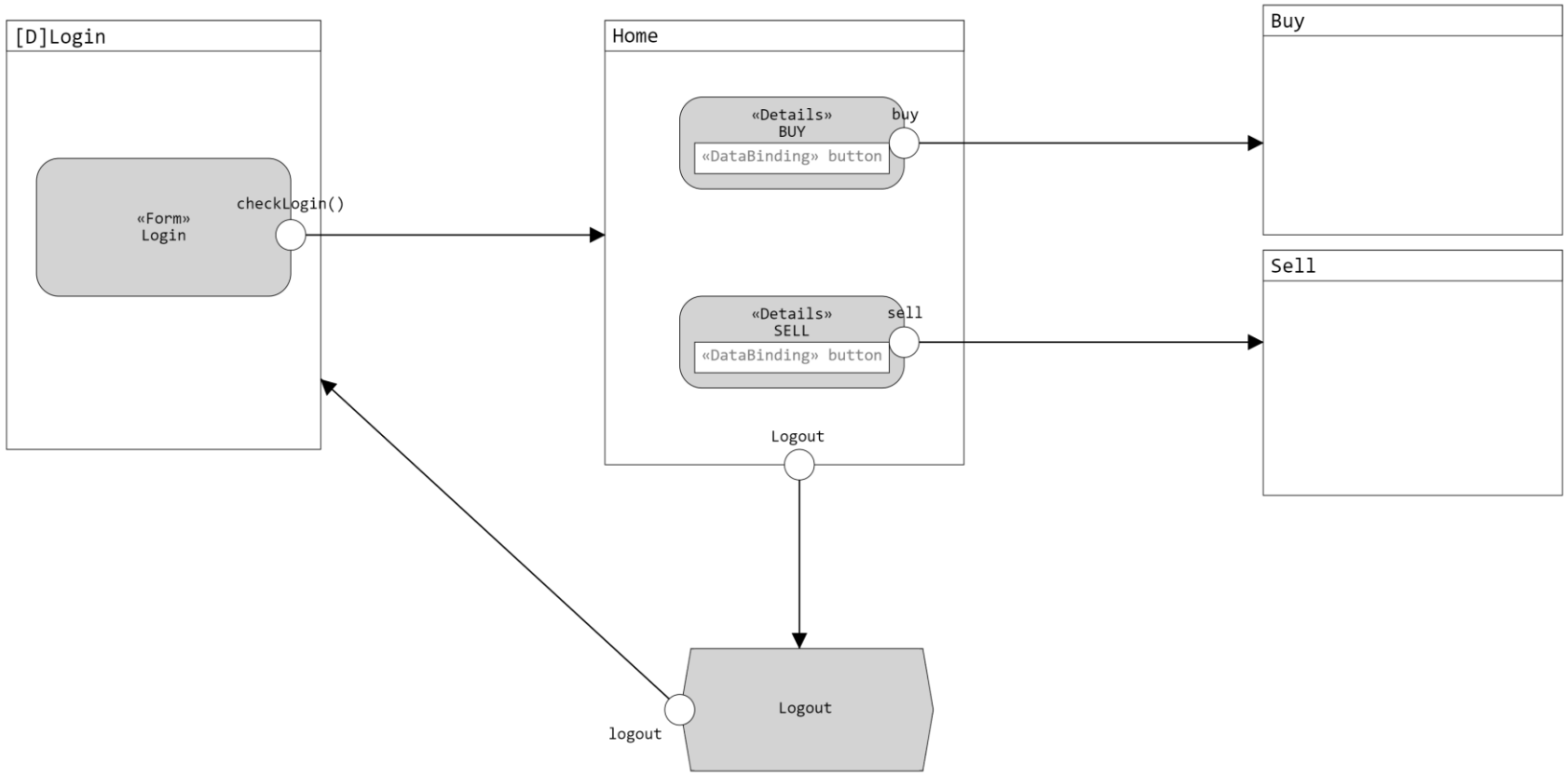
Local/Remote database schema

```
CREATE TABLE `item` (
  `iditem` int NOT NULL
AUTO_INCREMENT,
  `name` varchar(45) NOT NULL,
  `image` varchar(100) DEFAULT NULL,
  `description` varchar(255) DEFAULT
NULL,
  PRIMARY KEY (`iditem`)
)
```

```
CREATE TABLE `user` (
  `iduser` int NOT NULL
AUTO_INCREMENT,
  `username` varchar(45) NOT NULL,
  `password` varchar(45) NOT NULL,
  `firstname` varchar(45) NOT NULL,
  `lastname` varchar(45) NOT NULL,
  `address` varchar(255) NOT NULL,
  PRIMARY KEY (`iduser`)
)
```

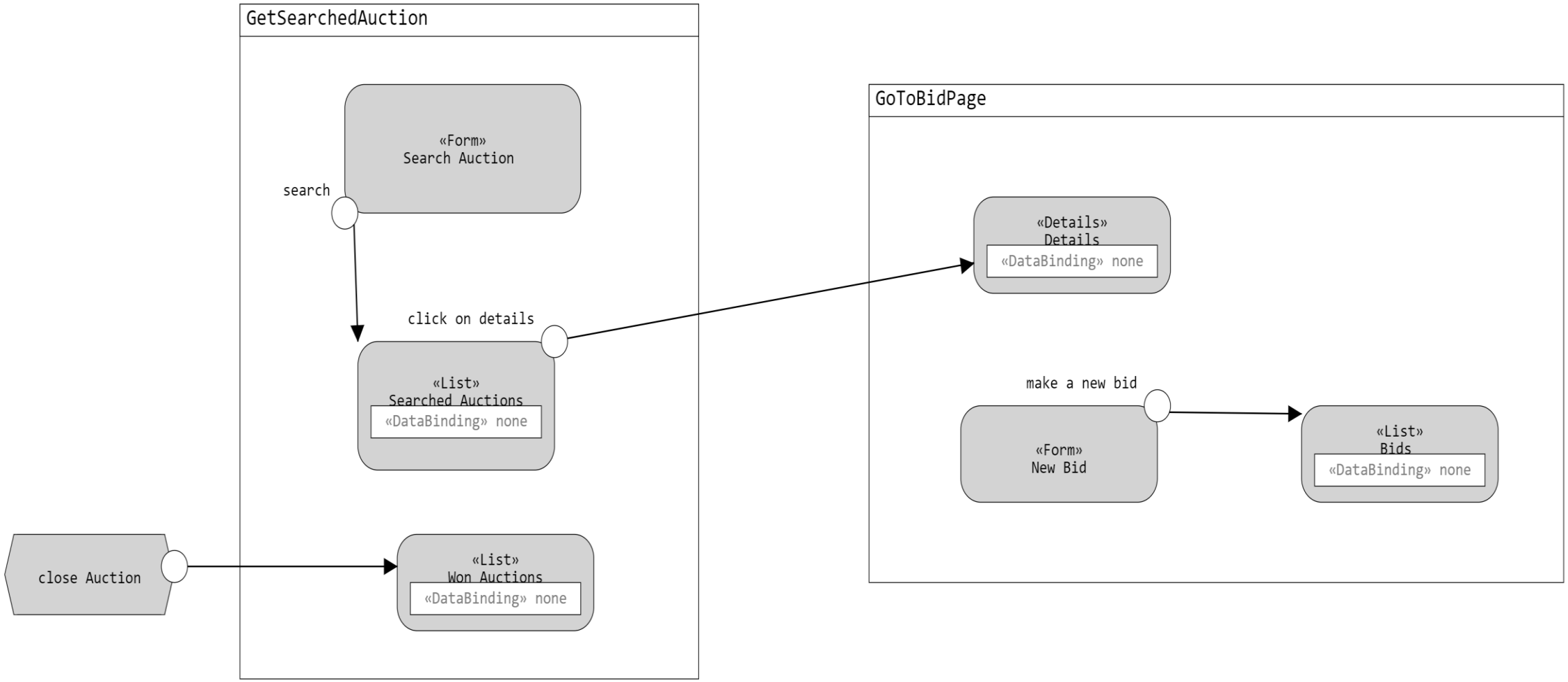
HTML pure application design (first part)

Login



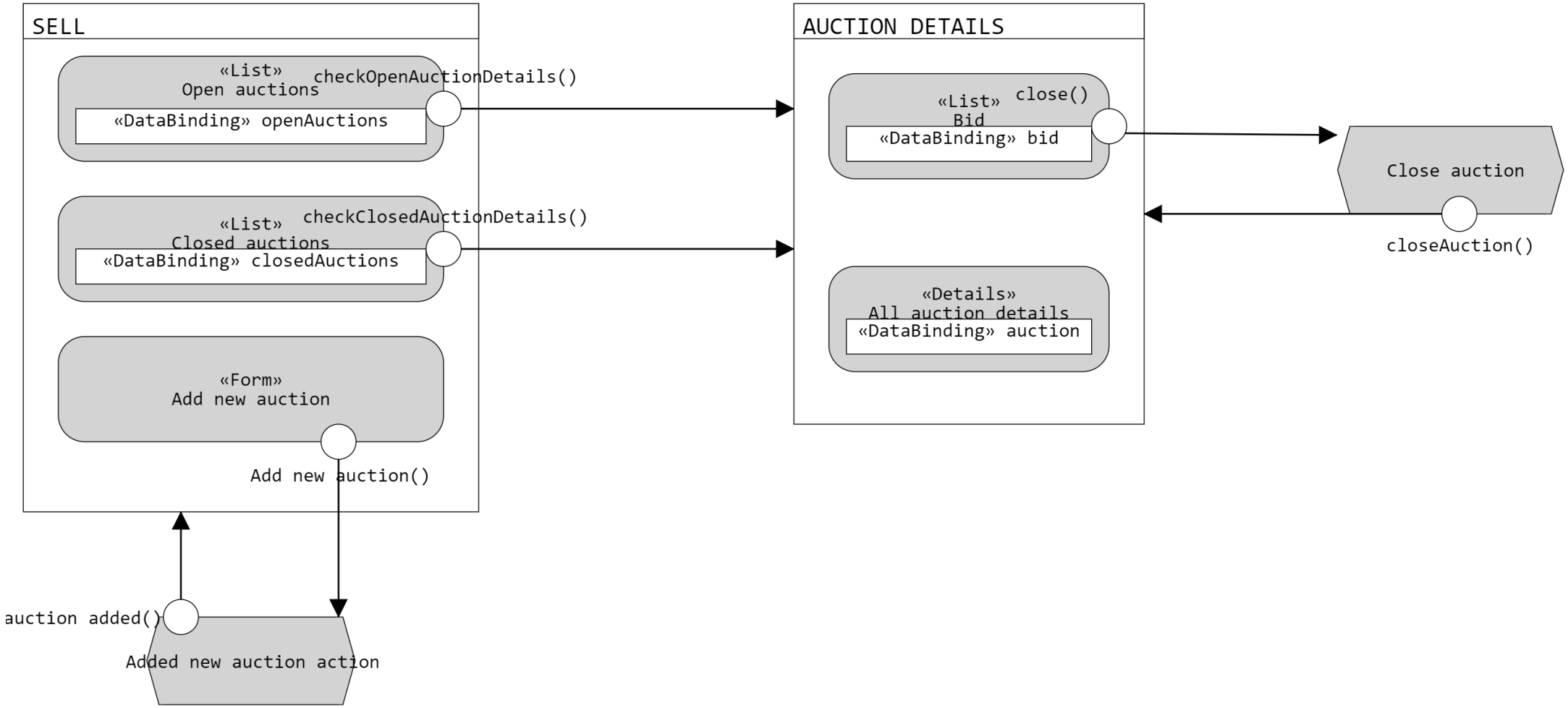
HTML pure application design (second part)

Buy



HTML pure application design (third part)

Sell



Components

- **Model Objects** (Beans)
  - Auction
  - AuctionStatus
  - Bid
  - ExtendedAuction
  - ExtendedBid
  - Item
  - User
- **Data Access Object** (DAO)
  - AuctionDAO
  - BidDAO
  - ItemDAO
  - UserDAO

- **Controllers** (Servlets)
  - AuctionDetailsServlet
  - AuctionDetailsServletHelper
  - CreateBid
  - GetSearchedAuction *Only in JavaScript*
  - GetUserData *Only in JavaScript*
  - GetWonAuction
  - GoToBidPage
  - GoToHomePage
  - ImageServlet
  - Login
  - Logout
  - SellHelperServlet
  - SellServlet

JavaScript

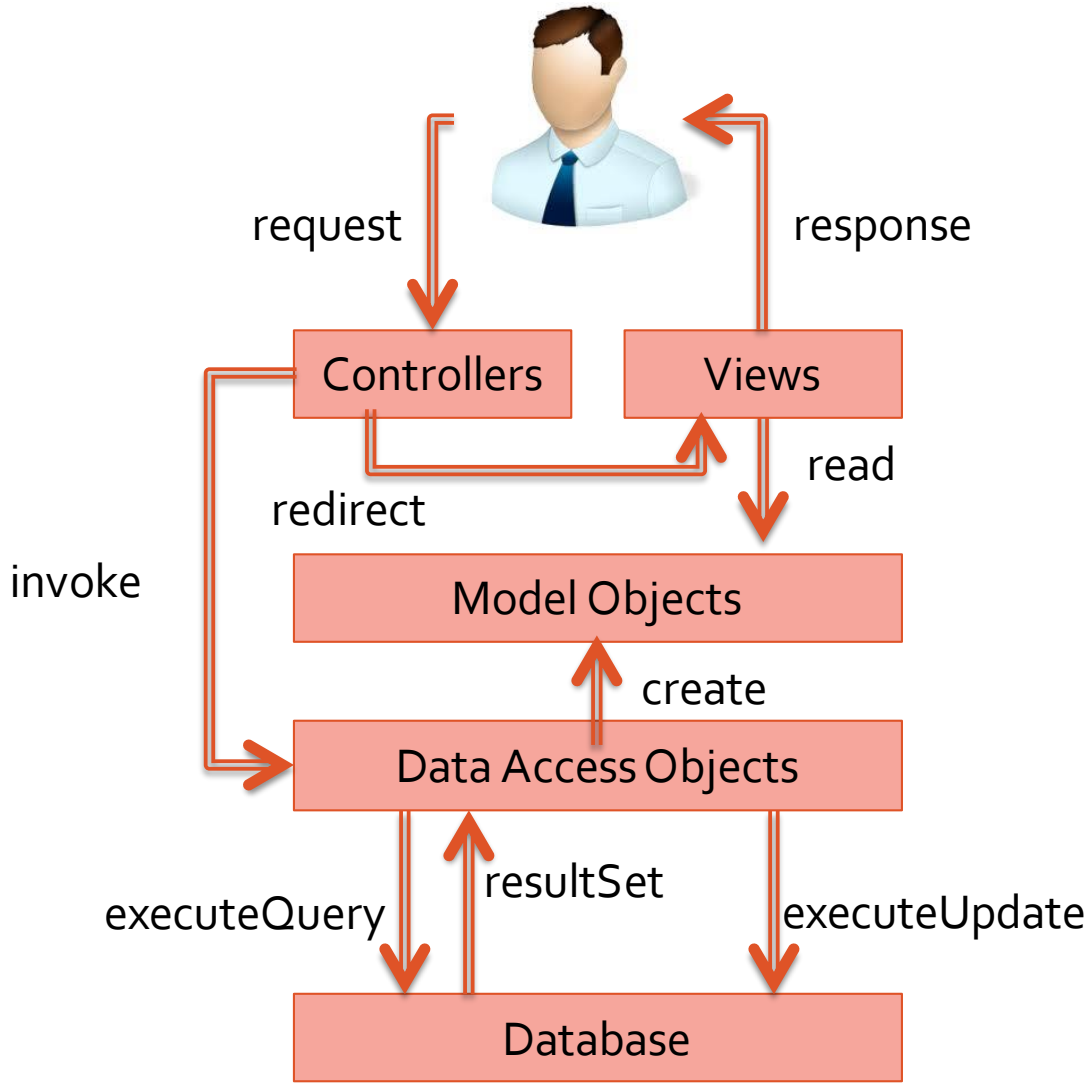
- **View** (Template)
  - HomeACT.html
  - Index.html
  - auctionManagement.js
  - buyViewManagement.js
  - loginManagement.js
  - sellViewManagement.js
  - util.js

HTML PURE

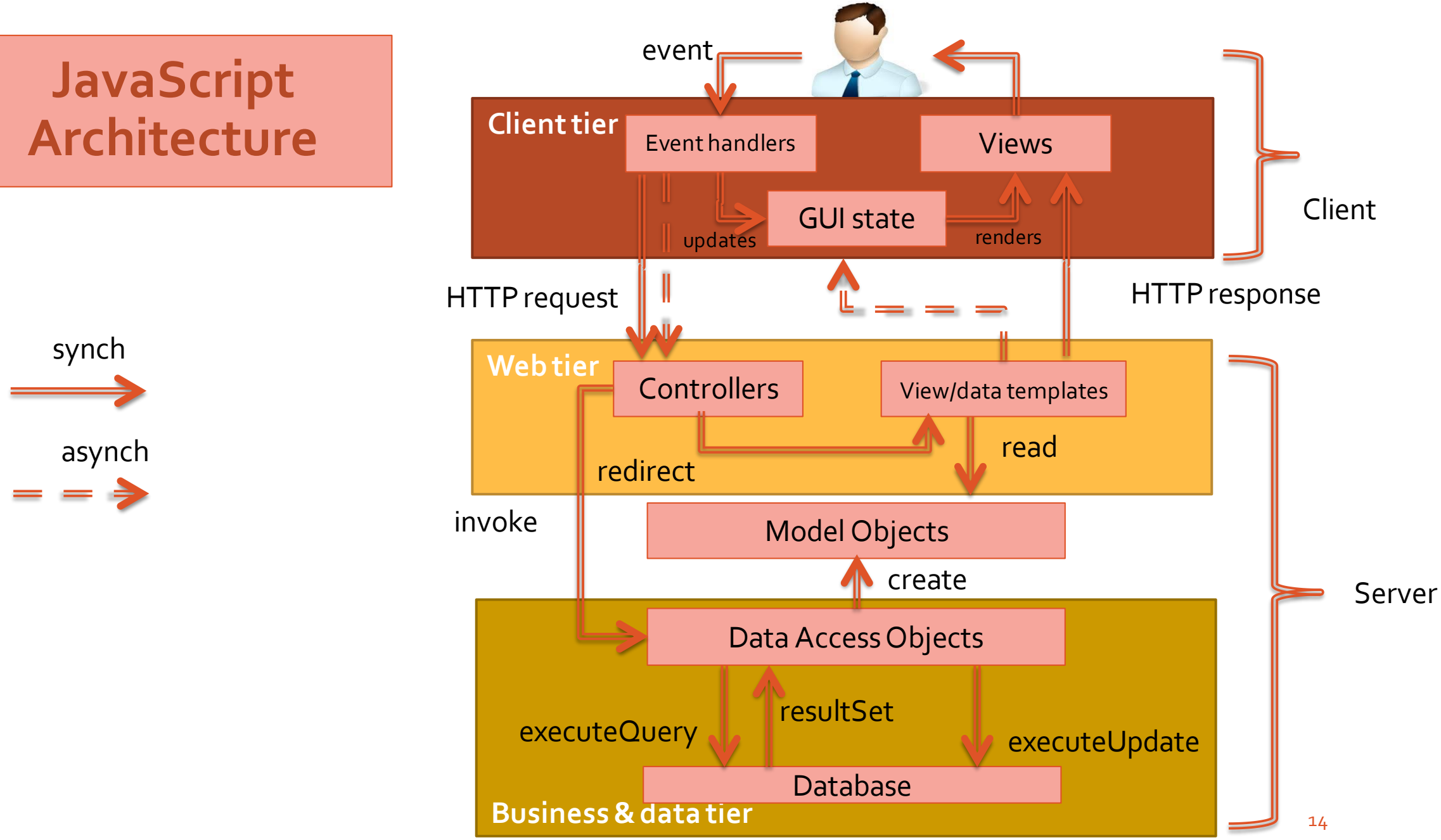
- **View** (Template)
  - AuctionDetails.html
  - GetSearchedAuction.html
  - GoToBidPage.html
  - Home.html
  - Sell.html
  - Index.html

HTML Pure Architecture

- Web tier
  - Model View Controller (MVC)
    - Controller: handles event
    - View: presents the interface
    - Model: store the application status
- Business & data tier
  - If complex business logic is required, a service layer can be interposed between the controllers and the DAO layer , to make DAO cope only with data access



JavaScript Architecture



HTML PURE
&
JavaScript

Server side: DAO methods

- AuctionDAO
  - findOpenAuction (keyword, userid)
  - findAuctionsByIdAndStatus (idUser, status)
  - insertNewAuction (itemName, itemImage, itemDescription, initialPrice, minRaise, deadLine, idCreator)
  - closeAuction (auctionId)
  - findAuctionById (auctionId)
  - findAuctionIdsByUsernameId (usernameId)
  - findAuctionDeadLineById (int auctionId)
  - findLegitIdBid (int idUser)

- ✓ UserDAO
  - ✓ checkCredentials (usrn, pwd)
  - ✓ getUserById (userId)

- BidDao
  - findBidsByIdAuction (auctionId)
  - findWonBids (idBidder)
  - insertNewBids (bidPrice, idBidder, idAuction)
  - findPriceForNewBid (auctionId)
  - findMinRaise (auctionId)
  - findWinnerIdByAuction (auctionId)

- ItemDAO
  - InsertNewItem (newItem)
  - getItemById (idAuction)

Events and actions JS

Client side		Server side	
Evento	**Azione**	**Evento**	**Azione**
Index → Login form → Submit	Check credentials	POST username password	Check credentials
Home page→ Load	Loads initial page elements (Decides whether to load buy or sell view)	GET (no parameters)	Retrieve user data
Home page→ Open Buy view	Updates data to show on Buy view	GET (last visited auction ids)	Retrieve won auctions Retrieve last visited auctions
Home page→ Open Sell view	Updates data to show on Sell view	GET (no parameters)	Retrieve open auctions by user Retrieve closed auctions by user
Buy view → Search for auction	Searches for an auction and displays result	GET keyword	Retrieve auctions that match keyword (if any)
Buy view → Auction list → Click auction details	Show auction details	GET idauction	Retrieve selected auction
Buy Auction details → Place bid form → submit	Place a bid	POST idauction, bidprice	Insert a new bid for a specific auction
Sell view → Create new auction → submit	Create new auction	POST (auction data)	Insert a new auction
Sell view → Click auction details	Show auction details	GET idauction	Retrieve selected auction
Sell Auction details → Close auction button	Close selected auction	POST idauction	Close selected auction
Logout		GET	Session end

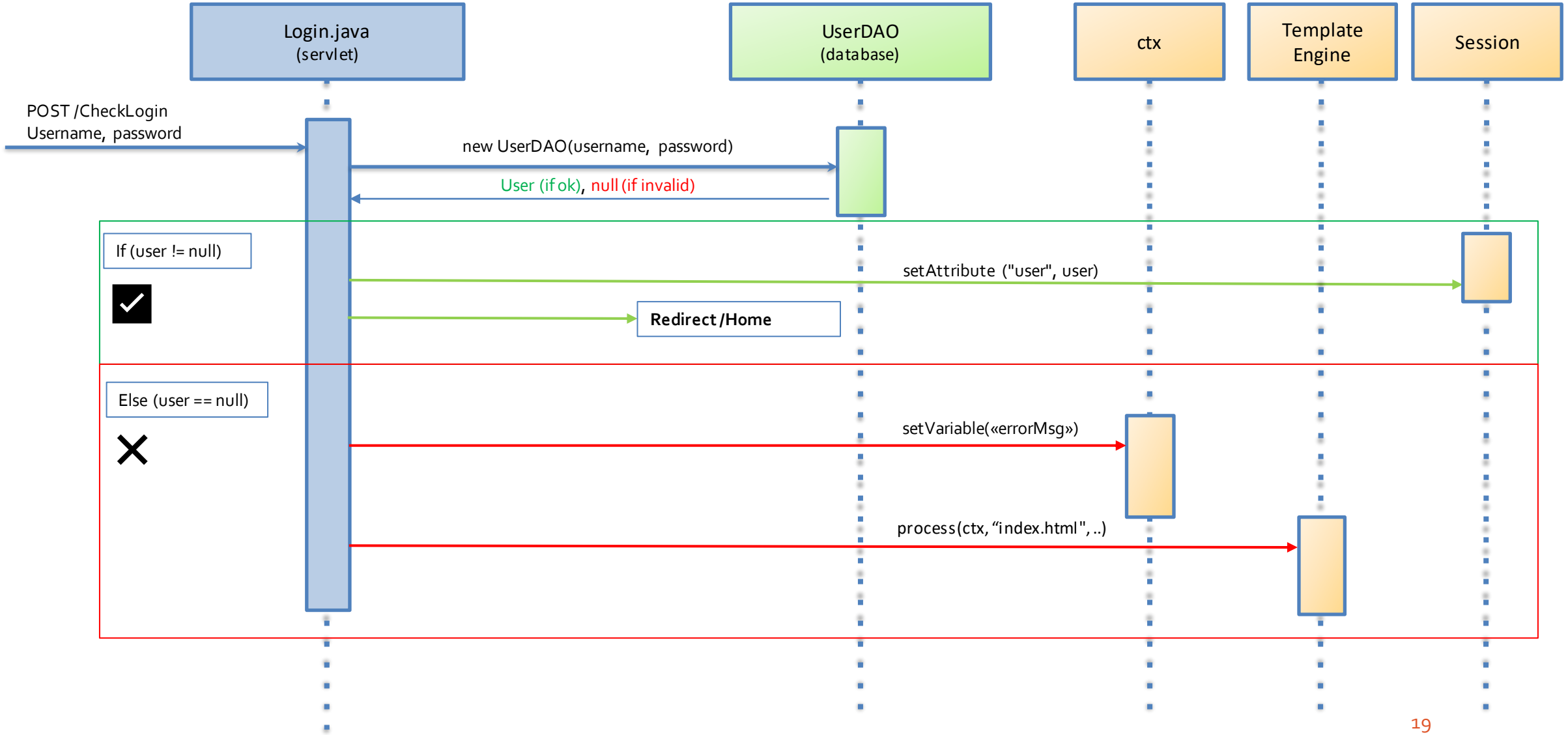
Controller/Event handler JS

Client side		Server side	
Evento	**Controllore**	**Evento**	**Controllore**
Index → Login form → Submit	Function makeCall	POST username password	CheckLogin (servlet)
Home page→ load	Function PageOrchestrator	GET (no parameters)	GetUserData (servlet)
Home page→ Open Buy view	Function SearchAuction.show Function WonAndLatestAuction.show	GET (last visited auction ids)	GetWonAuctions (servlet)
Home page→ Open Sell view	Function AuctionListSell.show	GET (no parameters)	SellServlet (servlet)
Buy view → Search for auction	Function AuctionList.show	GET keyword	GetSearchedAuction (servlet)
Buy view → Auction list → Click auction details	Function AuctionDetails.show	GET idauction	GoToBidPage (servlet)
Buy Auction details → Place bid form → submit	Function makeCall	POST idauction, bidprice	CreateBid (servlet)
Sell view → Create new auction → submit	Function makeCall	POST (auction data)	SellHelperServlet (servlet)
Sell view → Click auction details	Function AuctionDetailsSell.show	GET idauction	AuctionDetailsServlet (servlet)
Sell Auction details → Close auction button	Function makeCall	POST idauction	AuctionDetailsServletHelper (servlet)
Logout		GET	Logout (servlet)

HTML Pure Sequence Diagrams

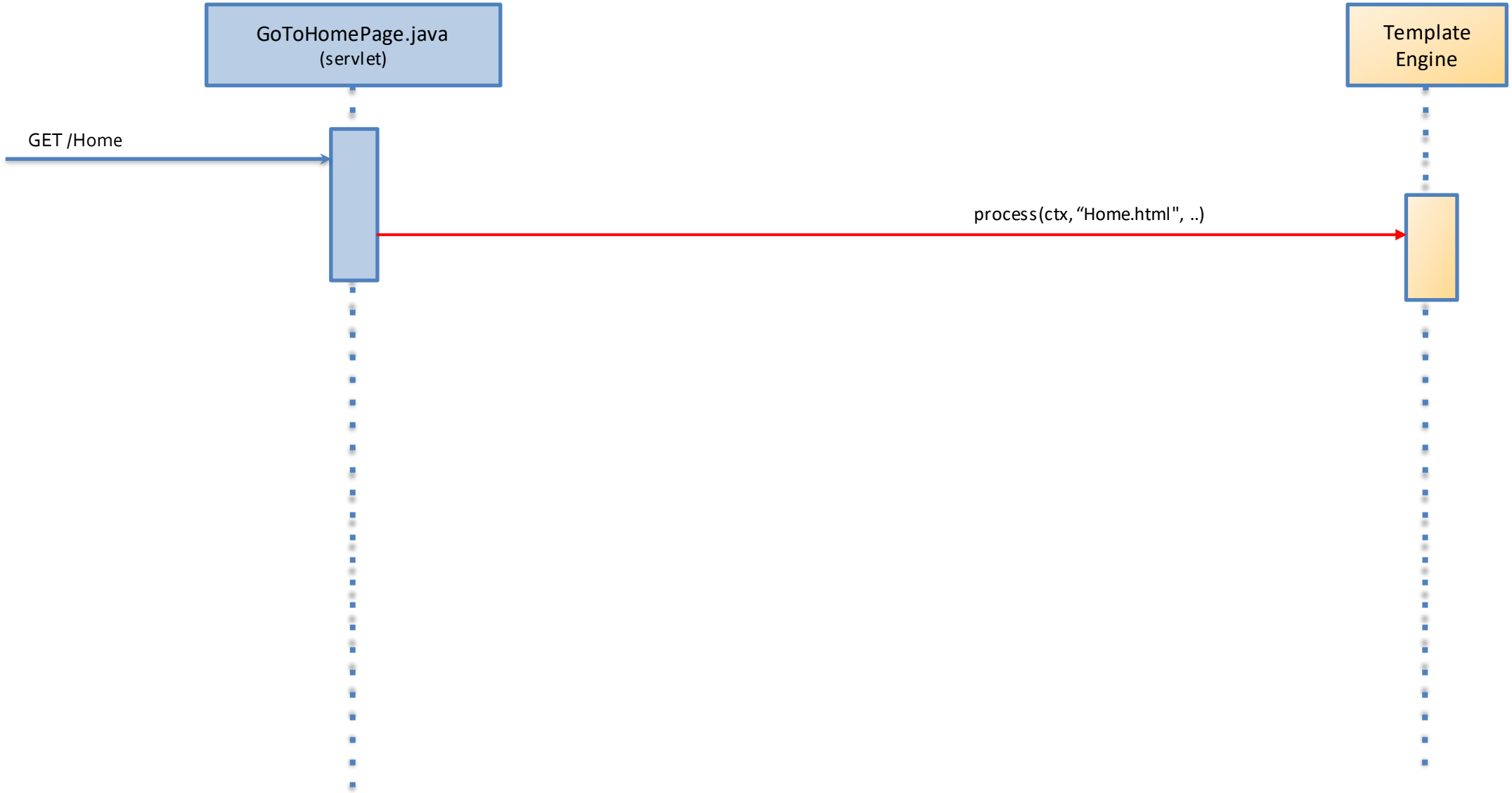
Sequence diagram (1/11)

Event: check login



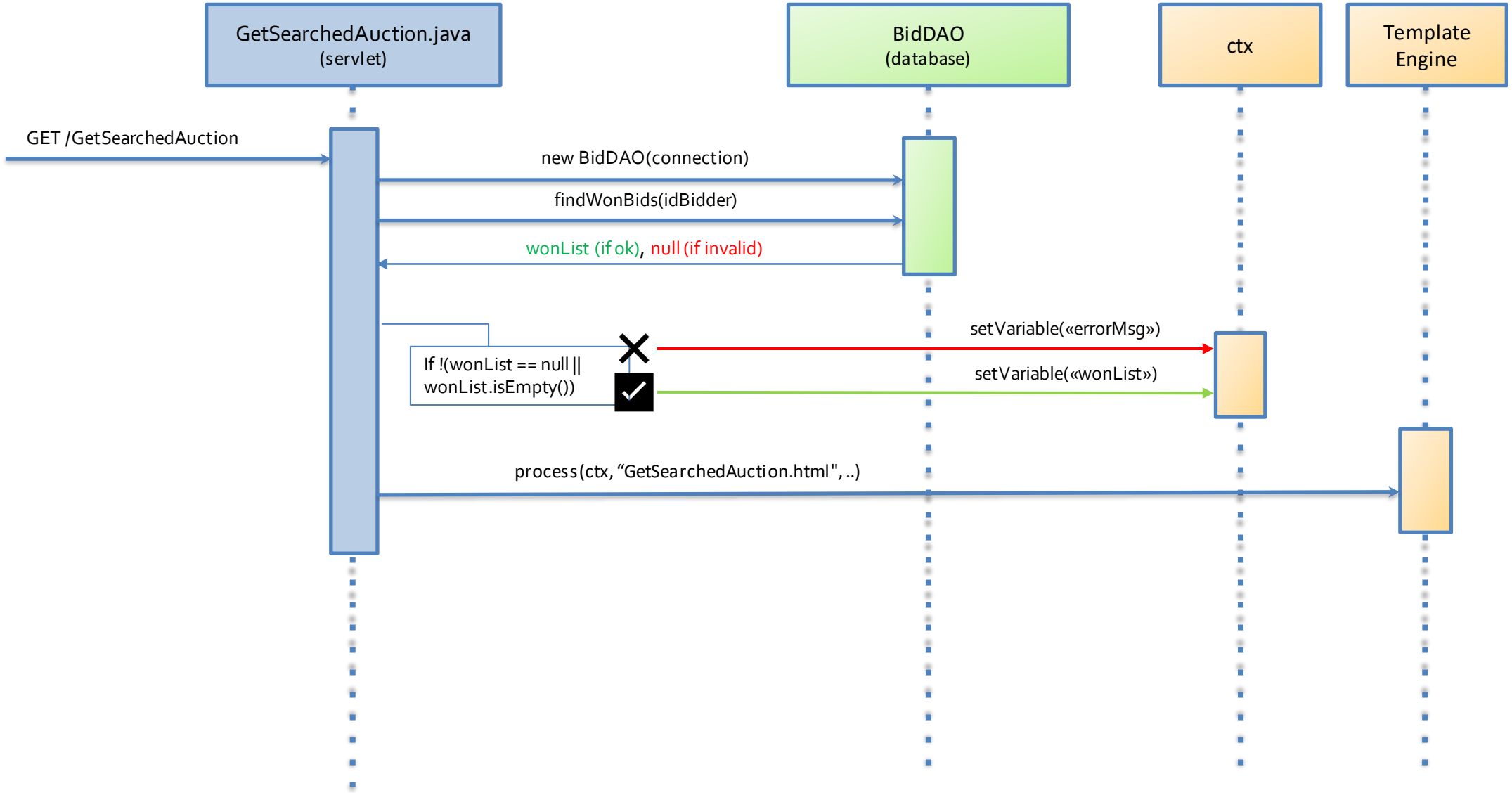
Sequence diagram (2/11)

Event: go to home page



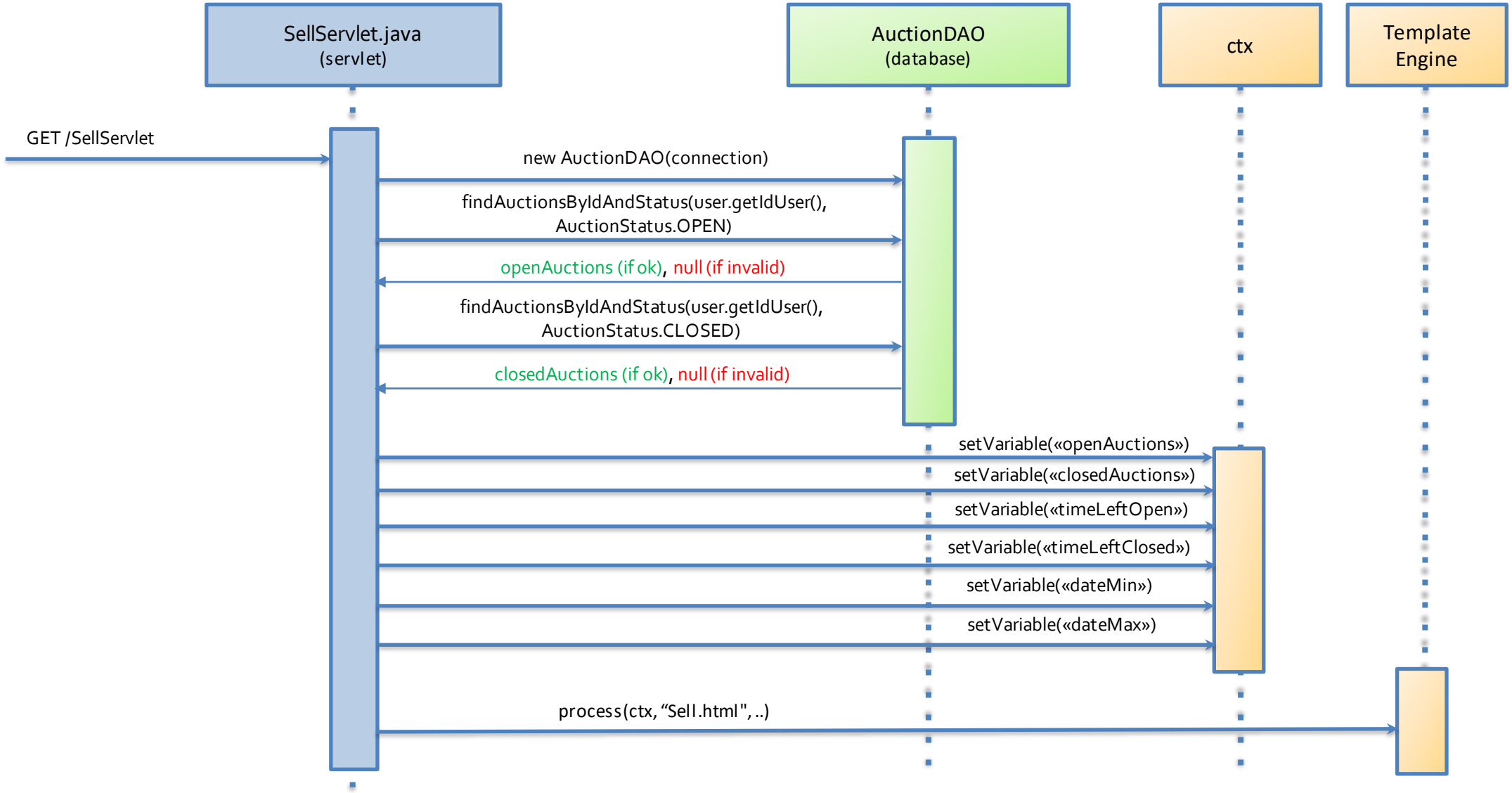
Sequence diagram (3/11)

Event: click on buy



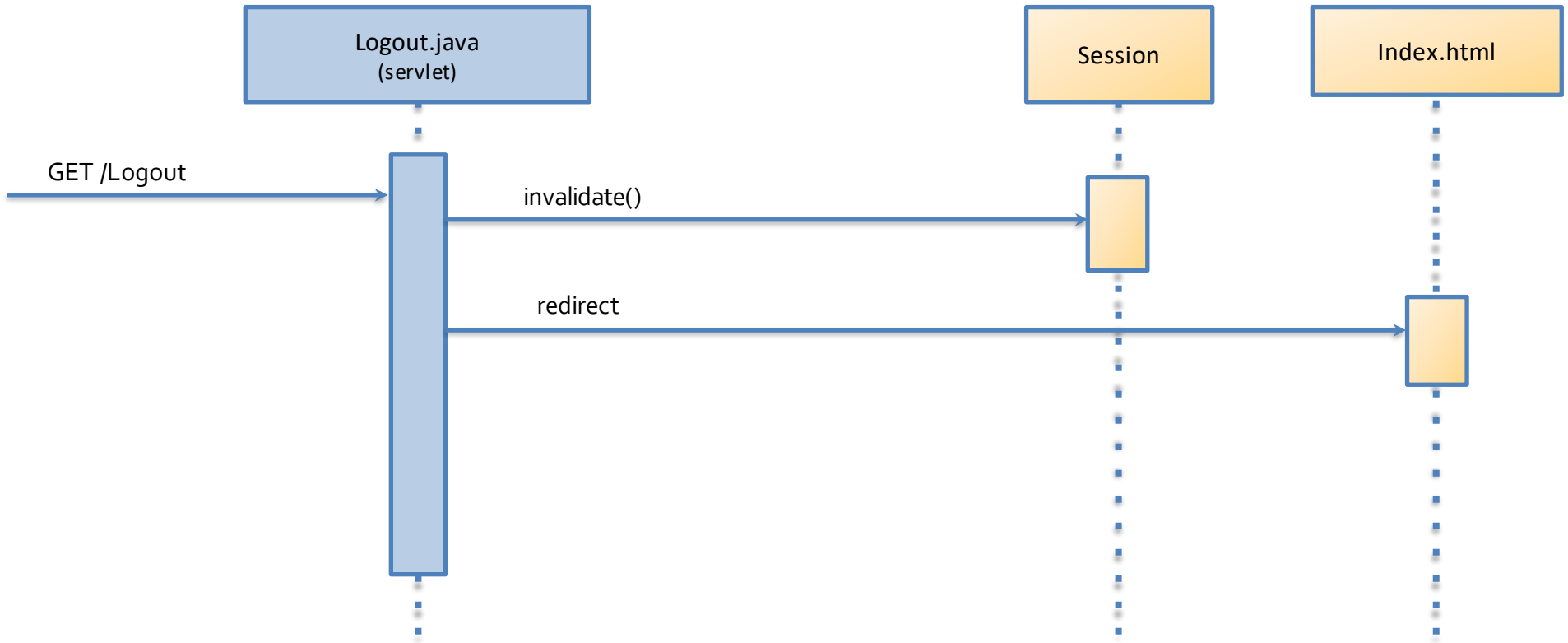
Sequence diagram (4/11)

Event: click on sell



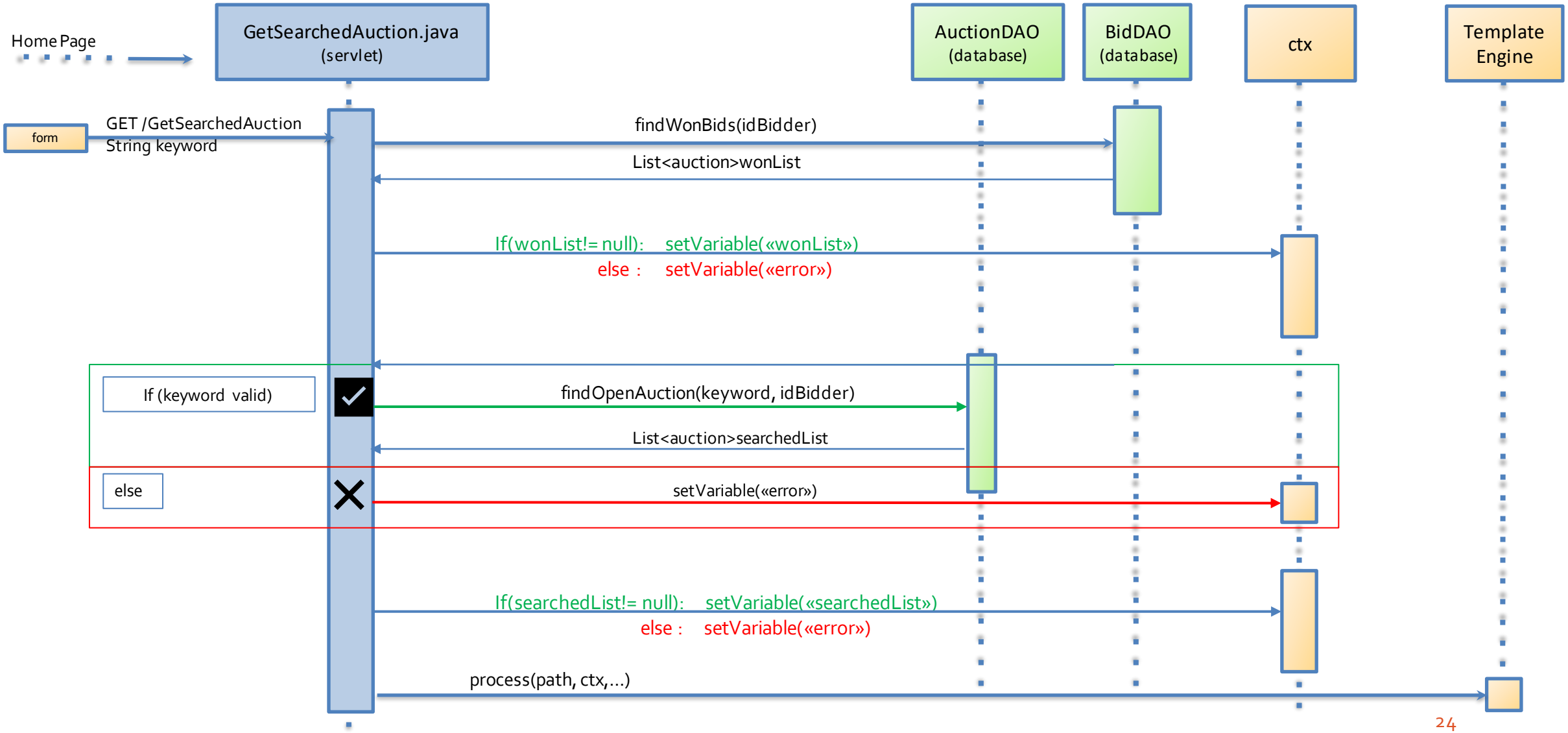
Sequence diagram (5/11)

Event: logout



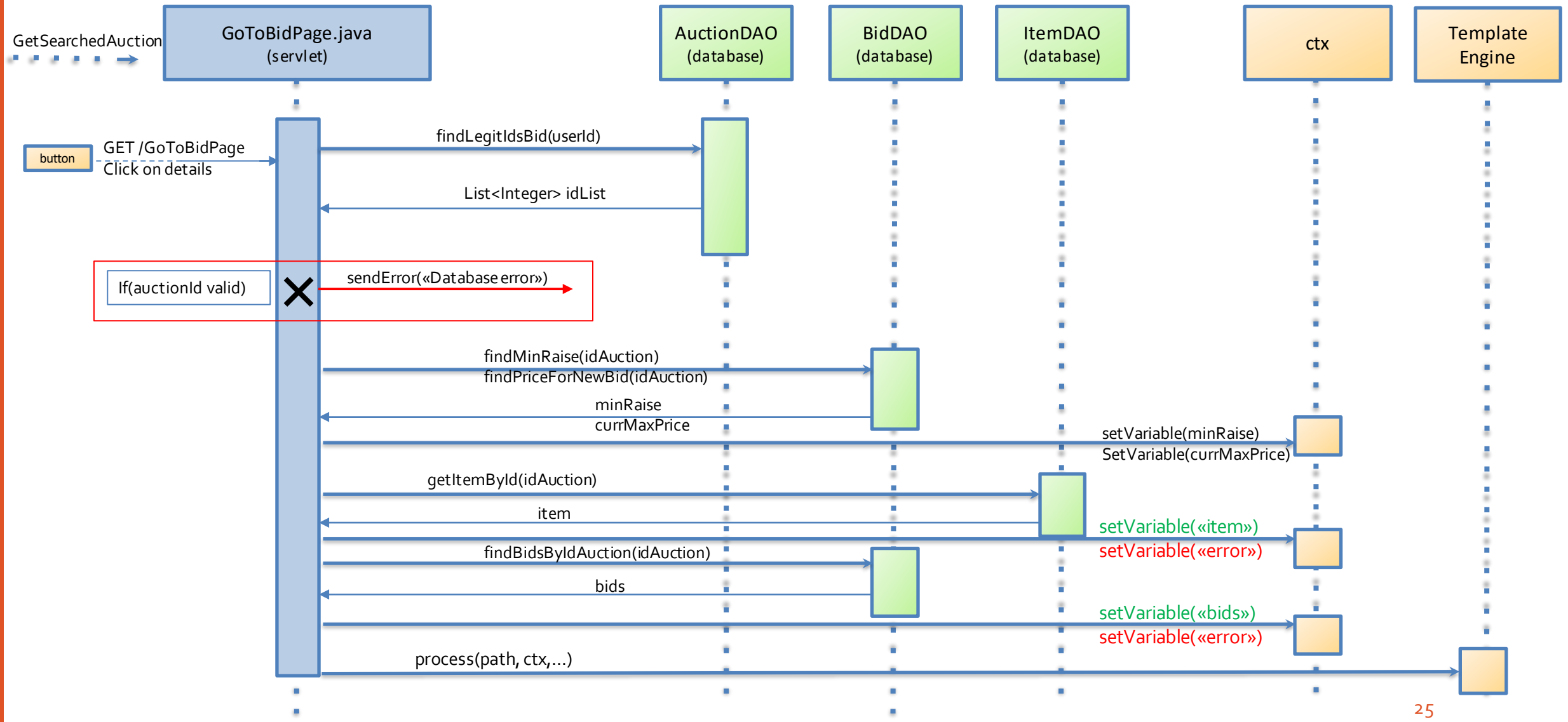
Sequence diagram (6/11)

Event: search auction & get won auctions



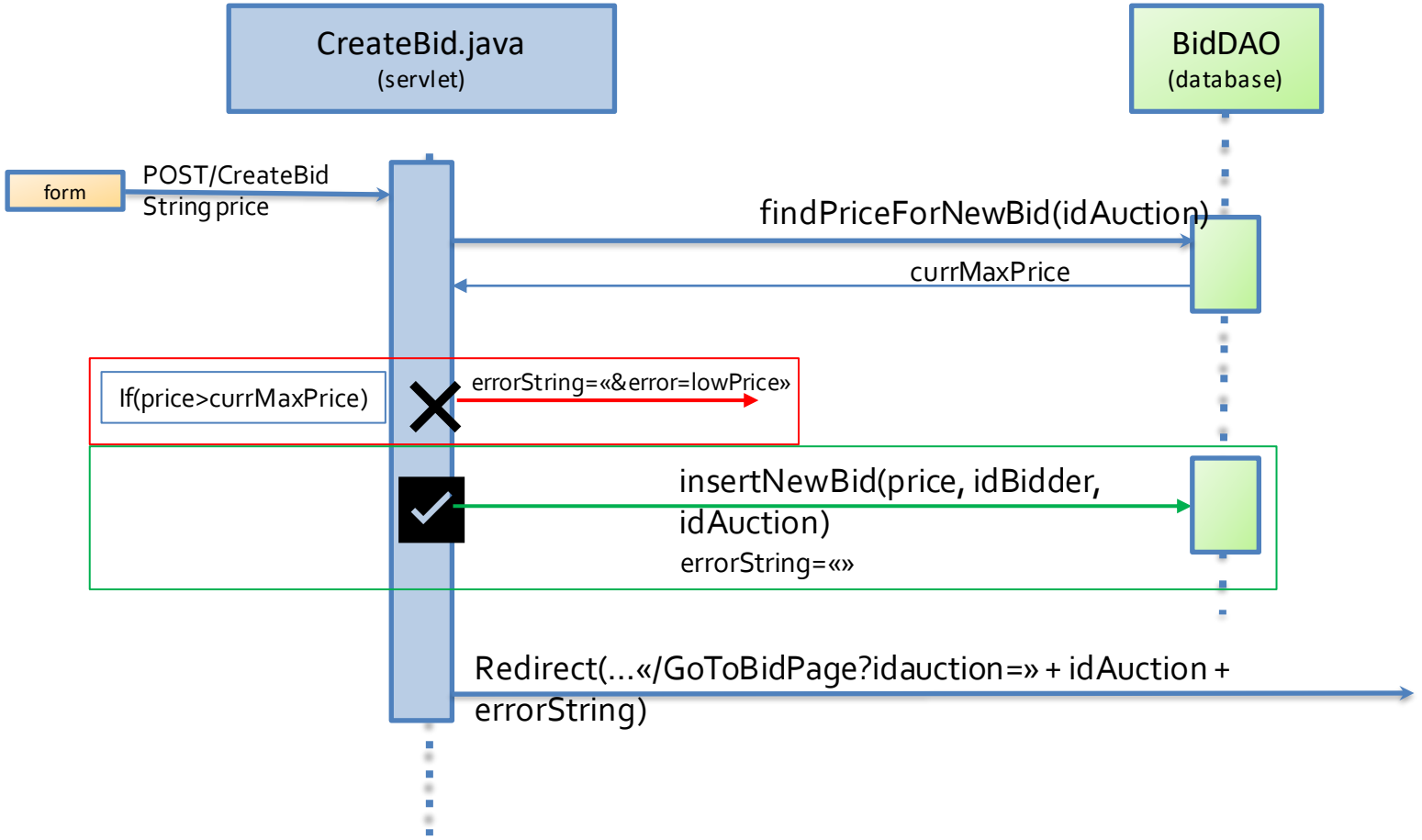
Sequence diagram (7/11)

Event: click on auction details



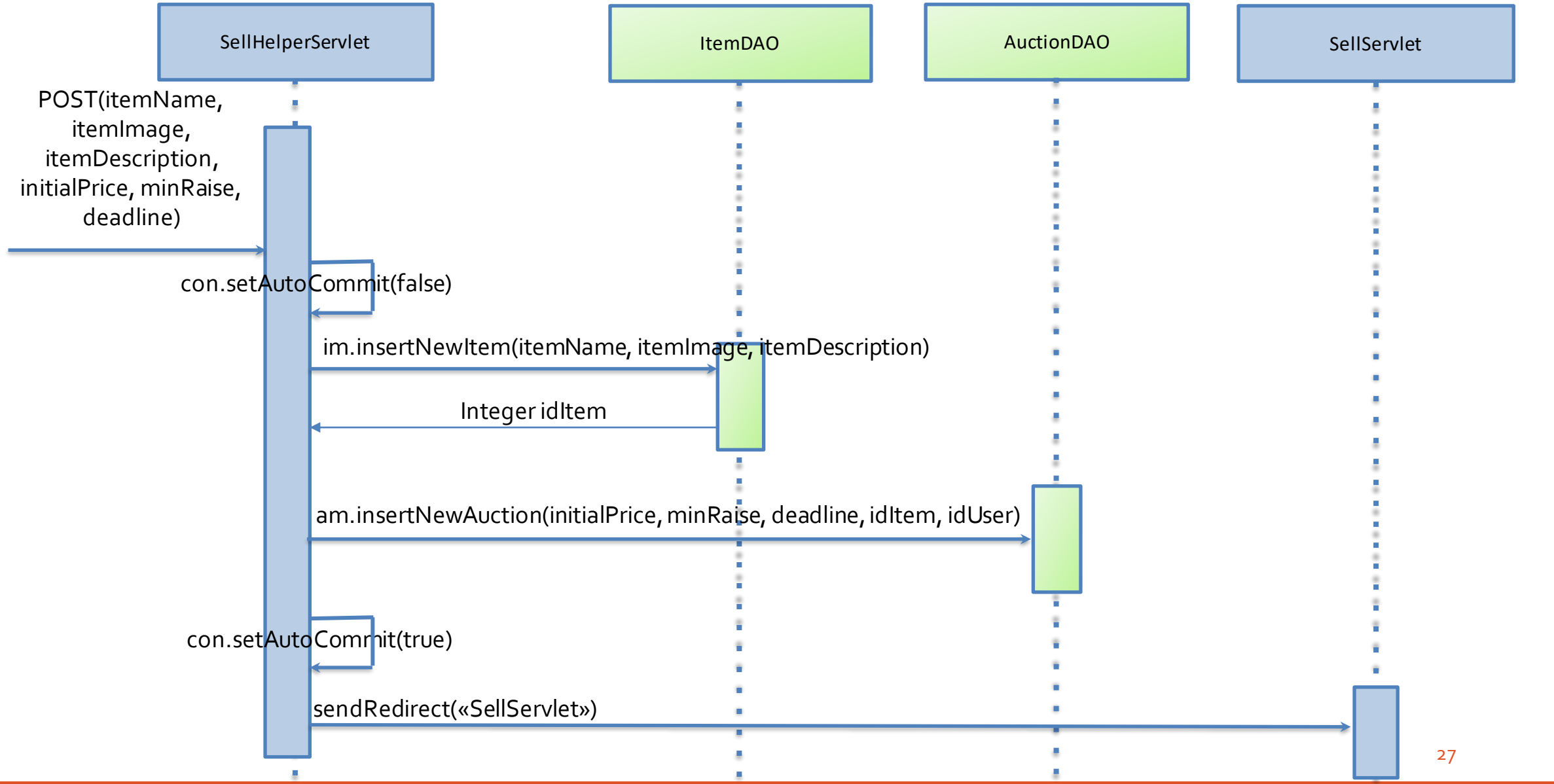
Sequence diagram (8/11)

Event: submit a new bid



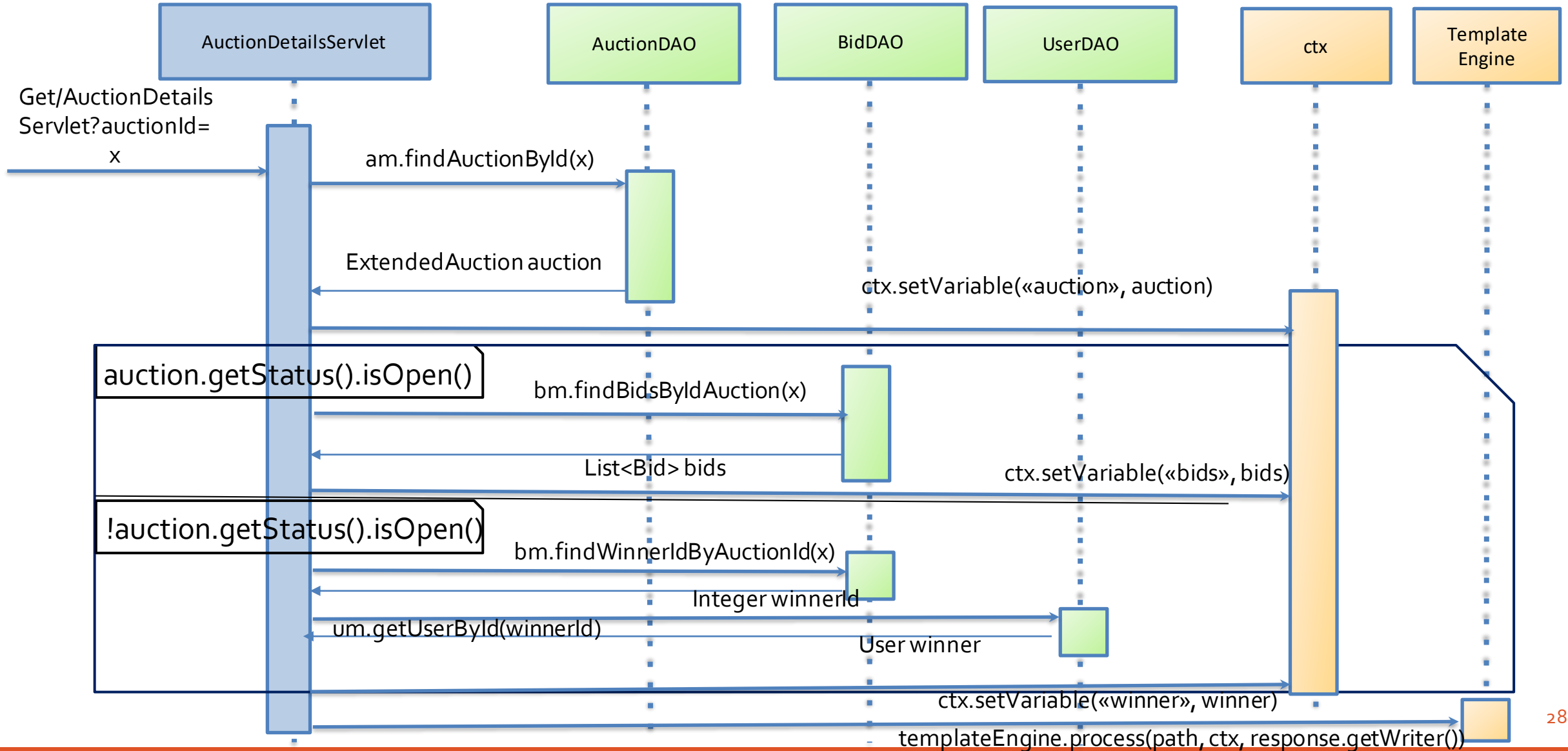
Sequence diagram (9/11)

Event: insert new auction



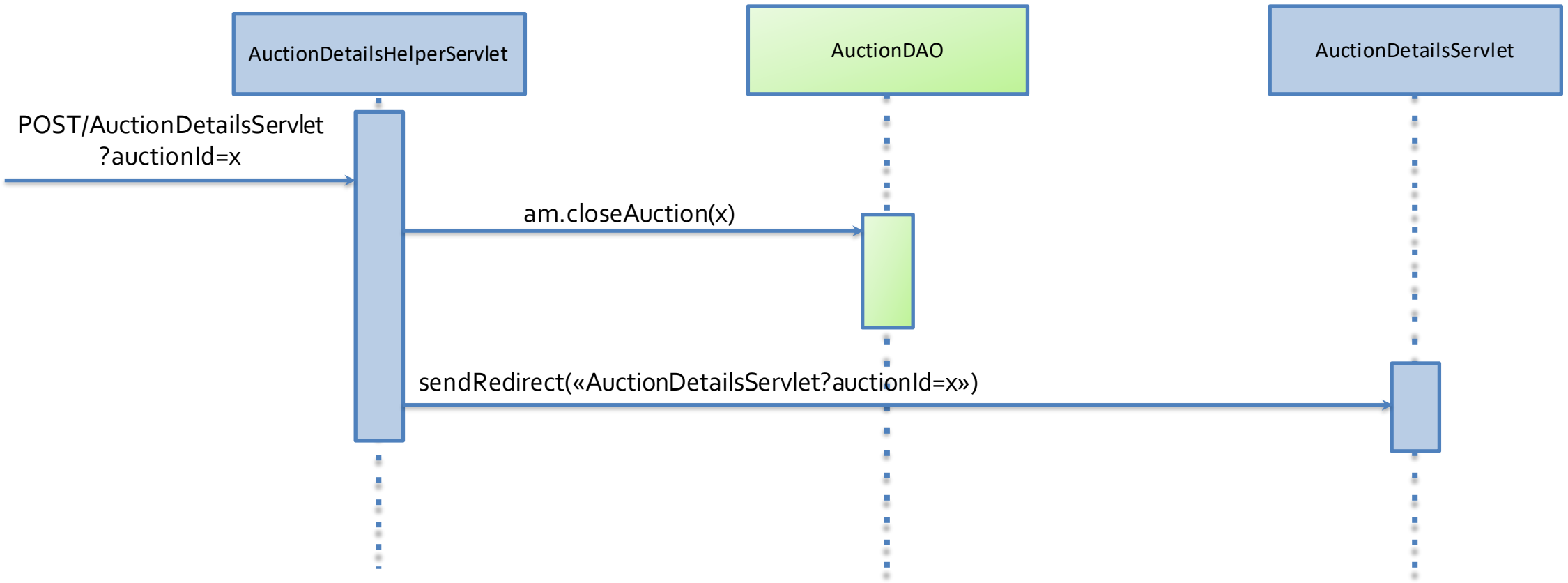
Sequence diagram (10/11)

Event: GET auction details



Sequence diagram (11/11)

Event: close auction



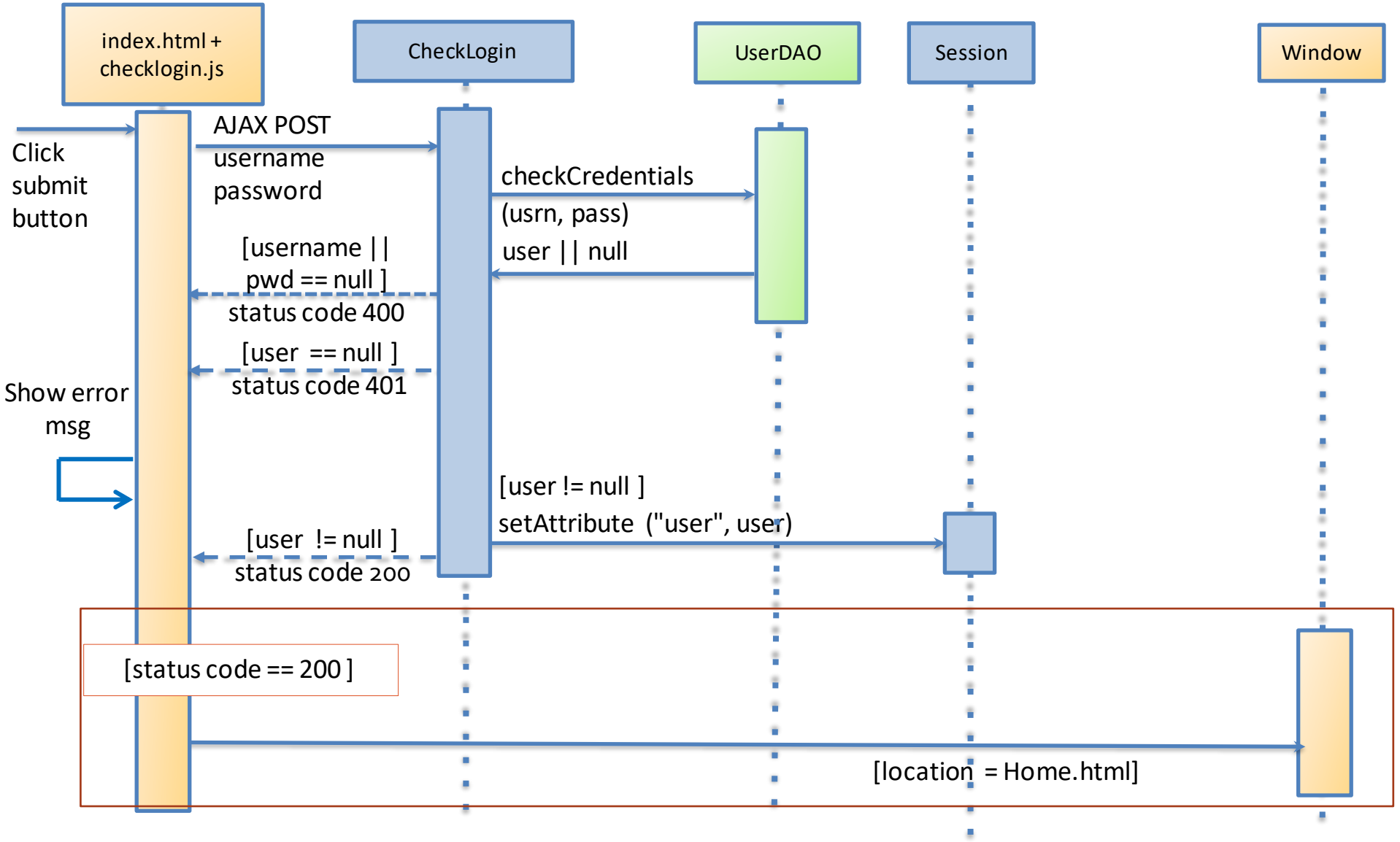
JavaScript Sequence Diagrams

Sequence diagram (1/11)

Event: login (JS)

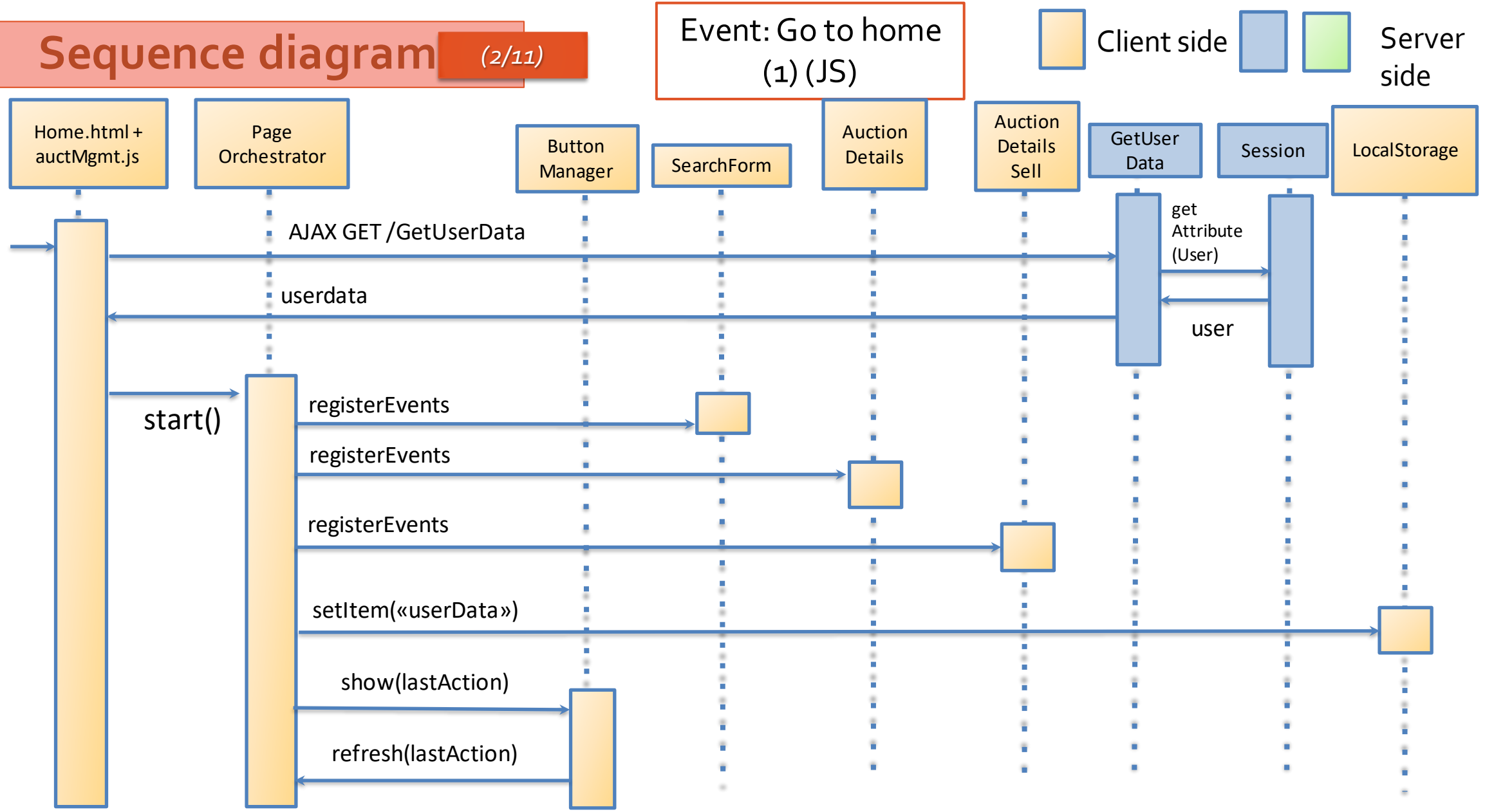
Client side

Server side



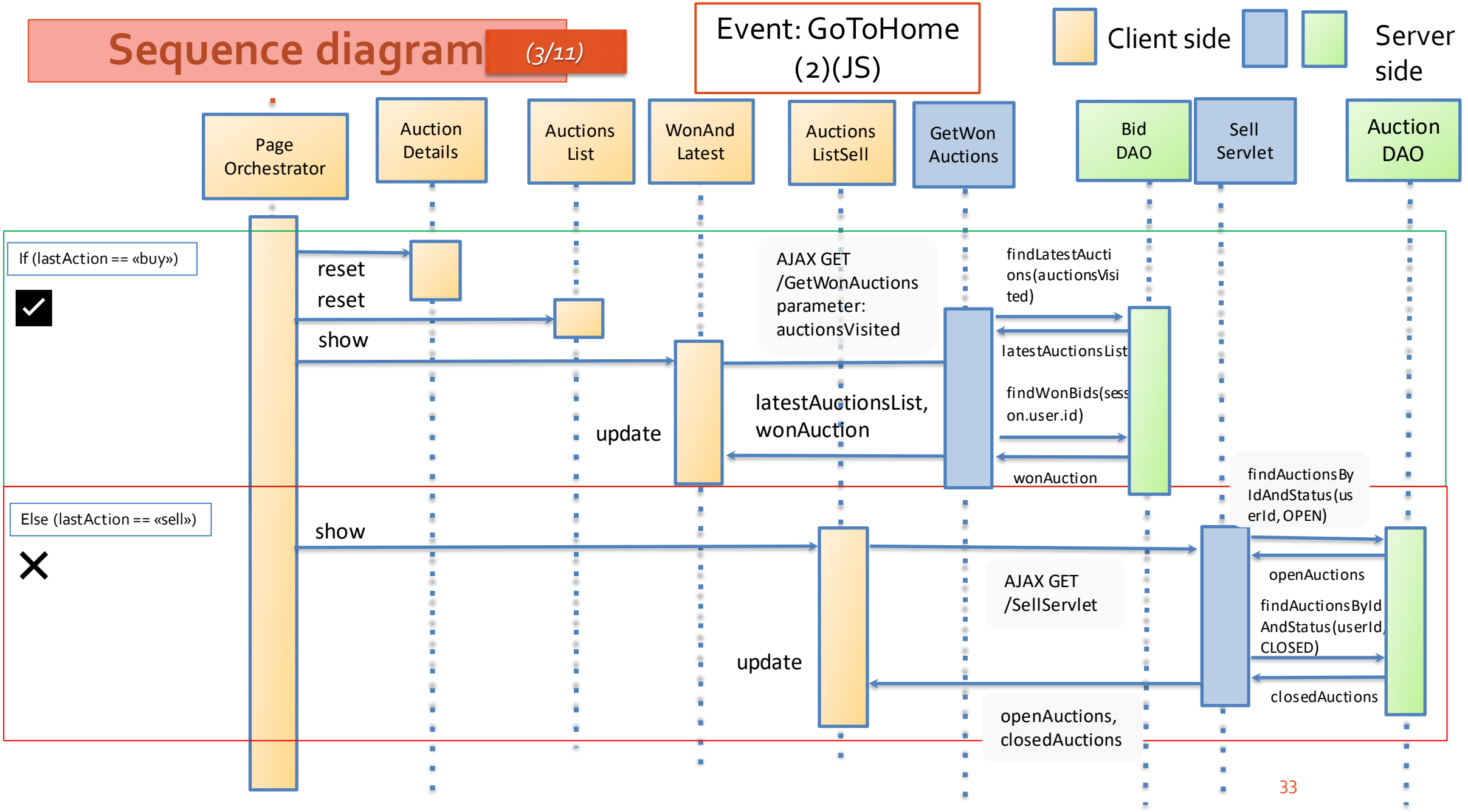
Sequence diagram (2/11)

Event: Go to home (1) (JS)



Sequence diagram (3/11)

Event: GoToHome (2)(JS)

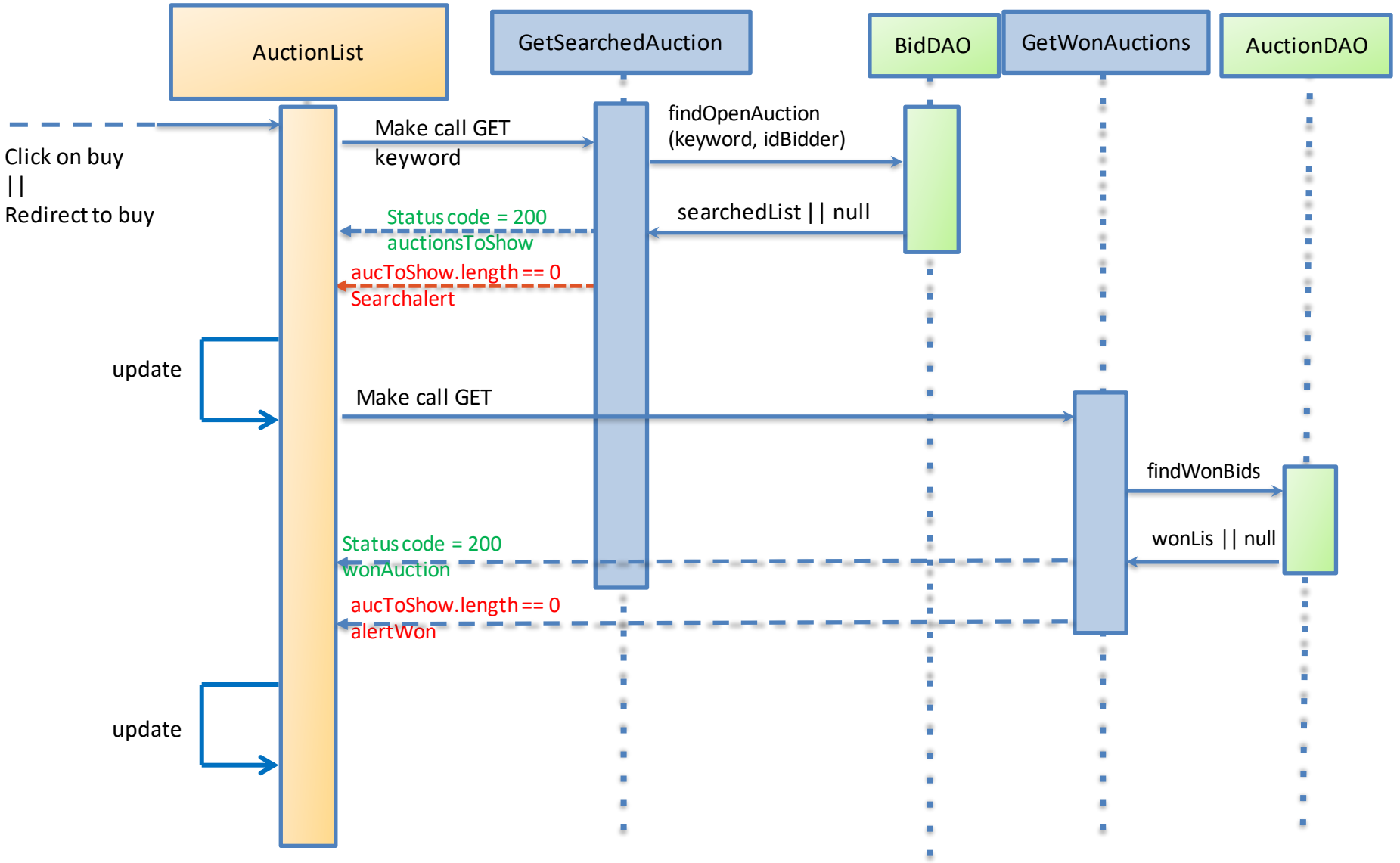


Sequence diagram (4/11)

Event: search auction & get won auctions (JS)

Client side

Server side

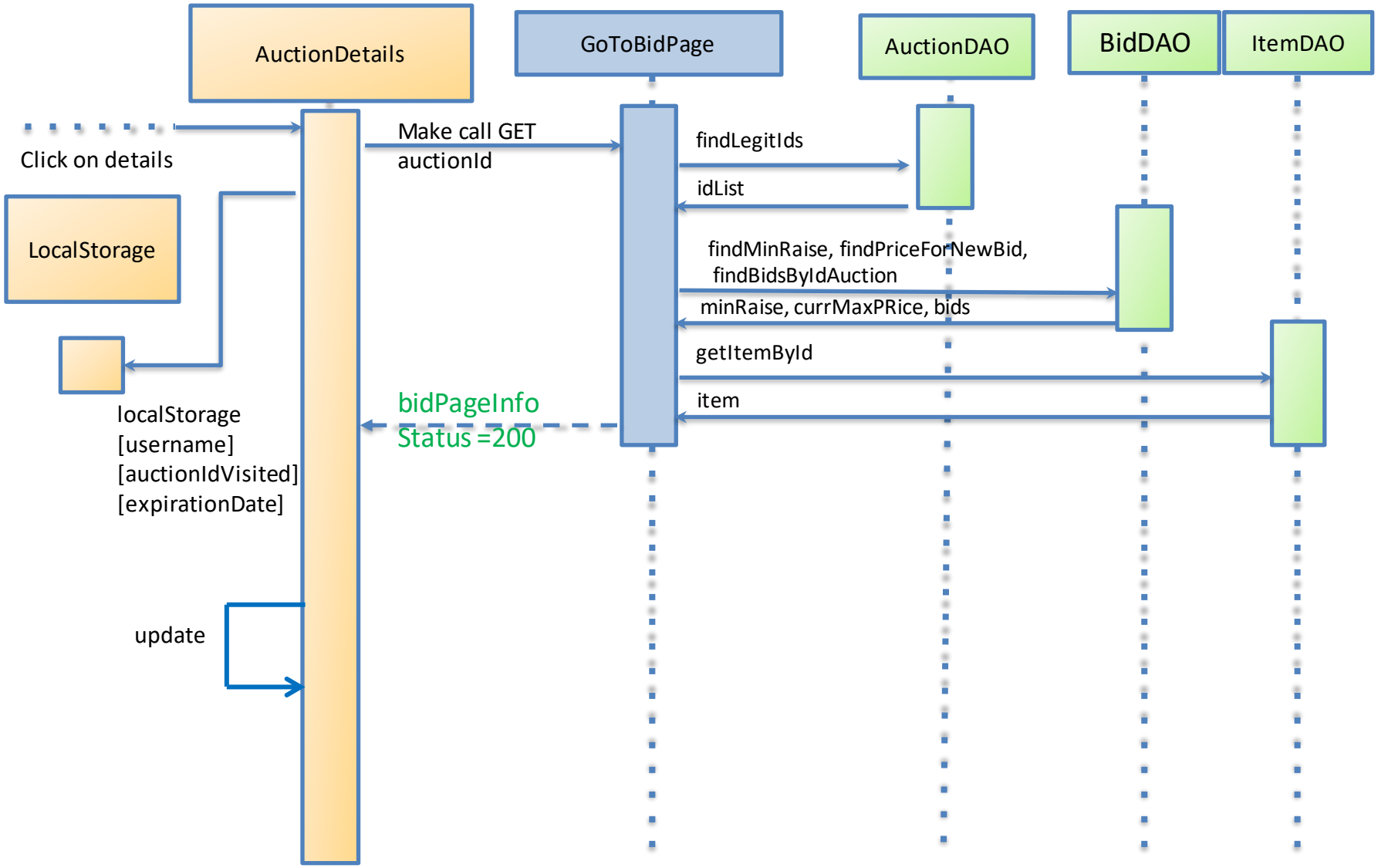


Sequence diagram (5/11)

Event: click on auction details (JS)

Client side

Server side

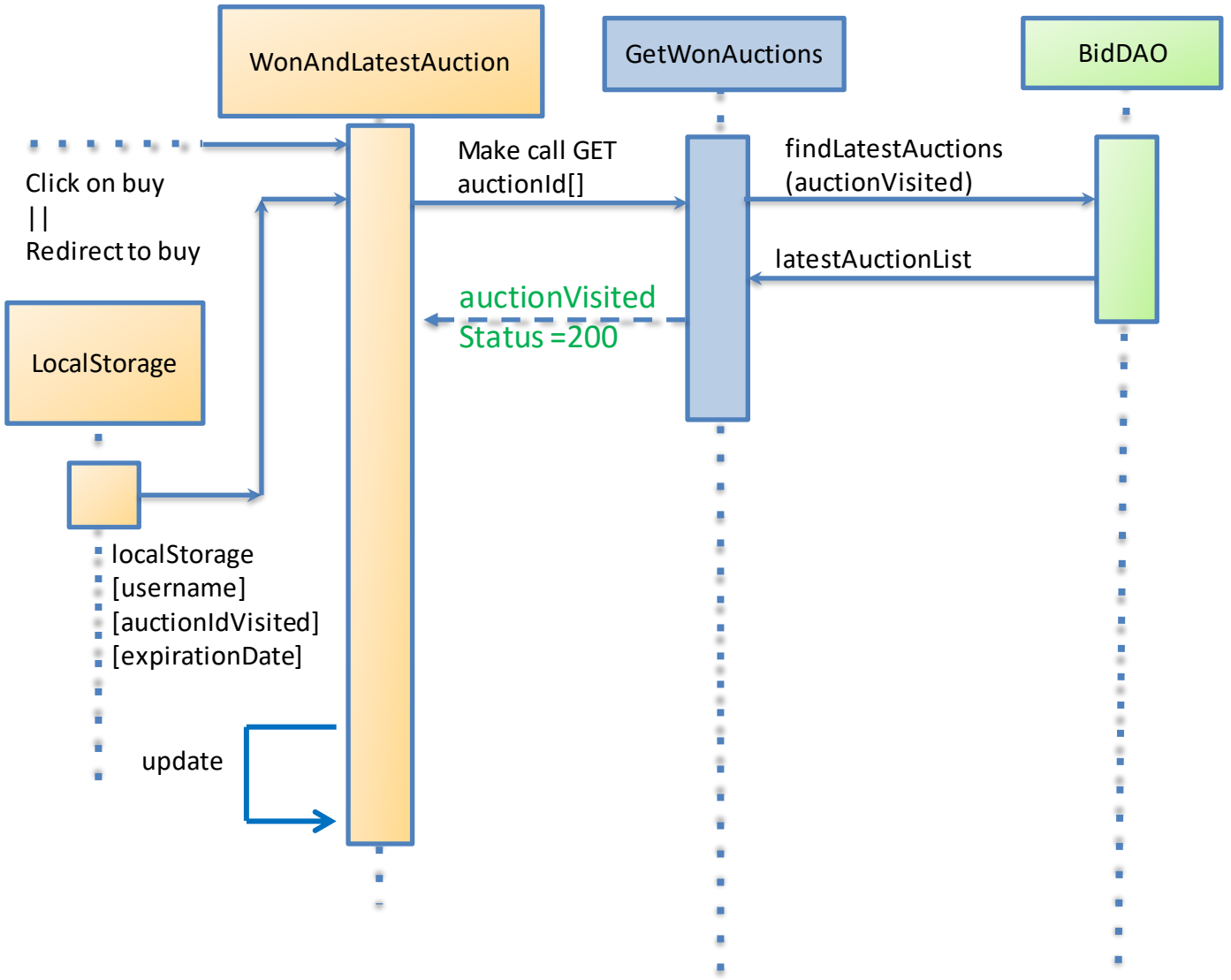


Sequence diagram (6/11)

Event: get latest visited auctions (JS)

Client side

Server side

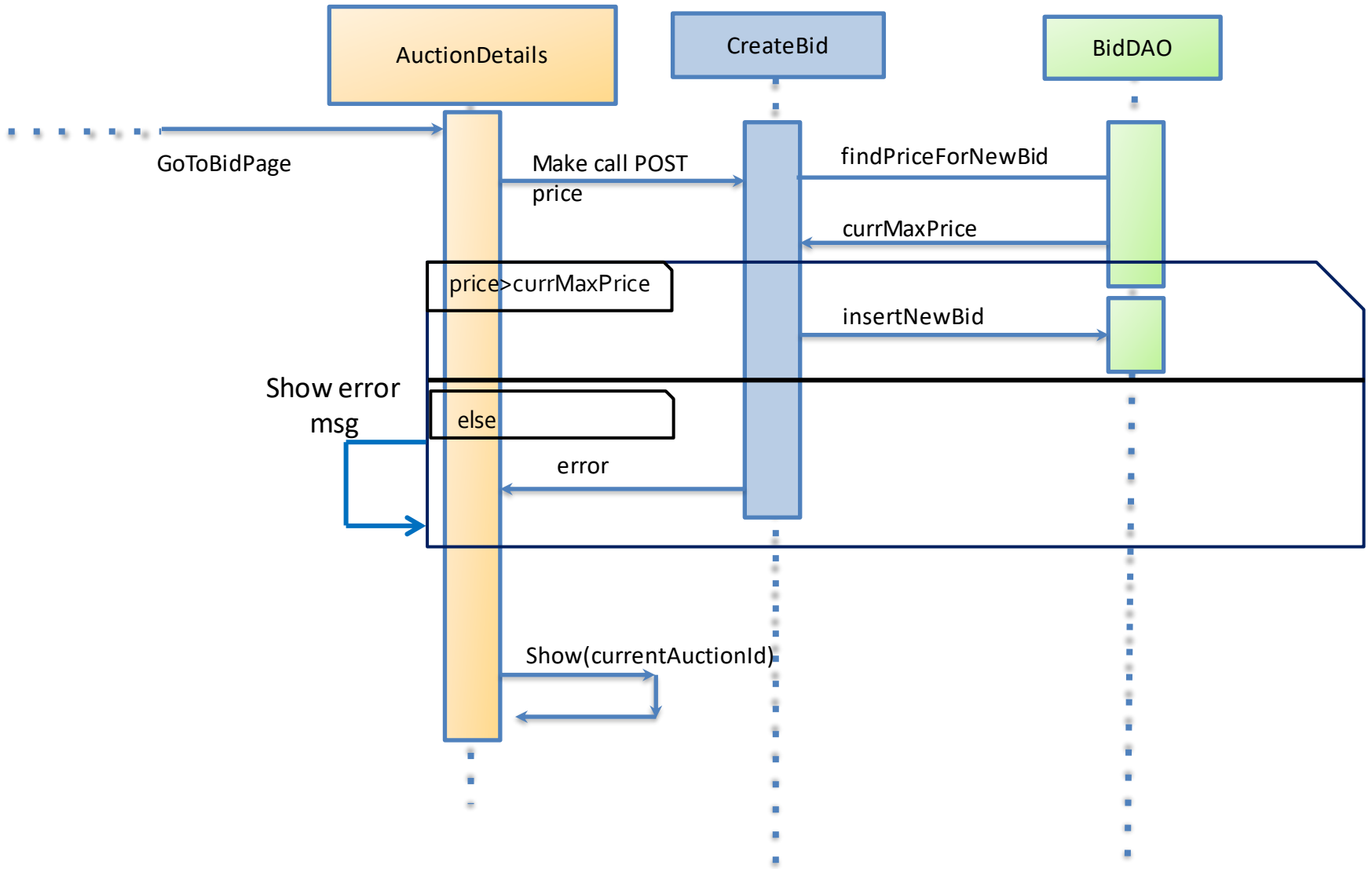


Sequence diagram (7/11)

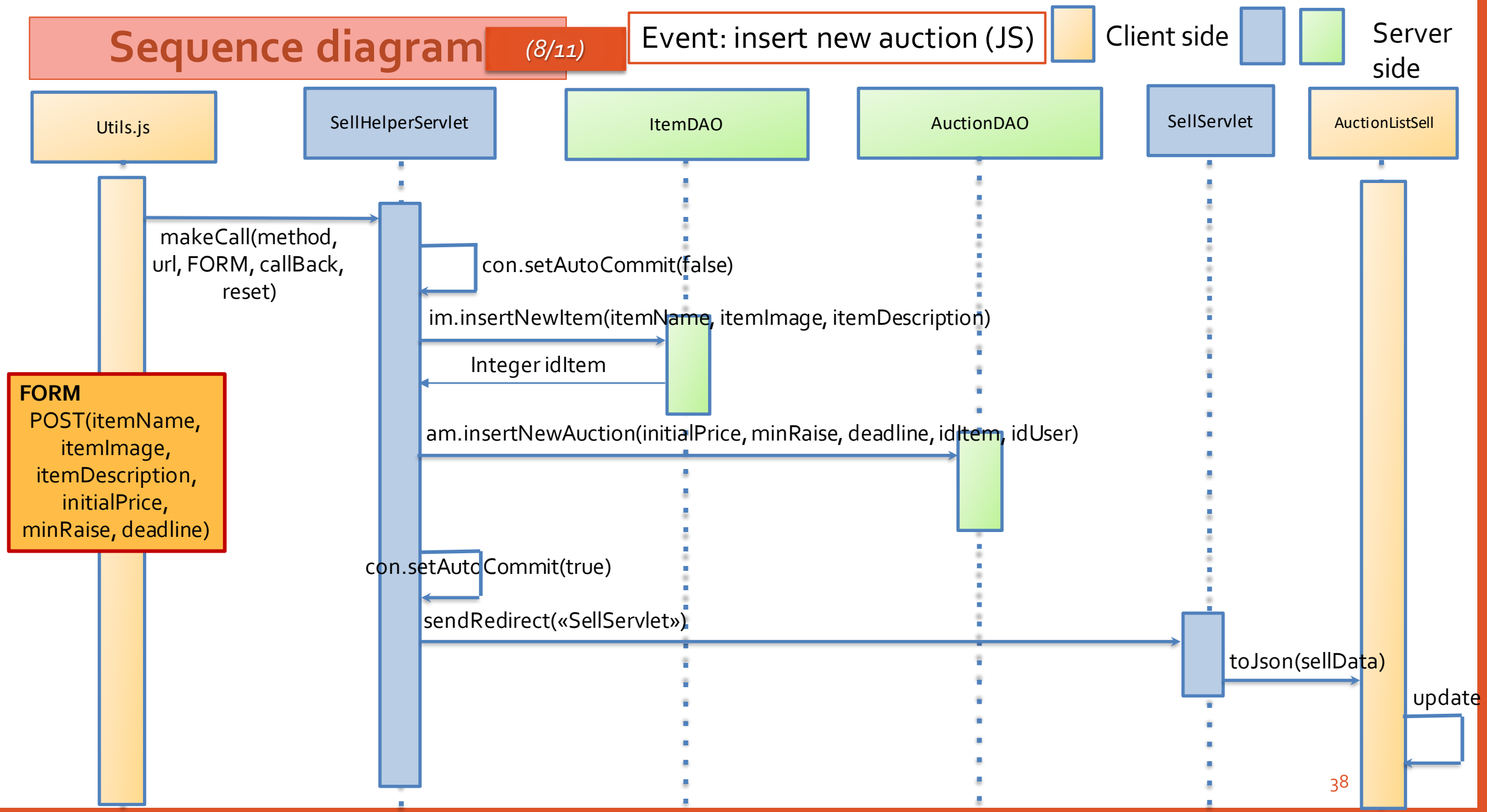
Event: create a new bid (JS)

Client side

Server side



Sequence diagram (8/11)

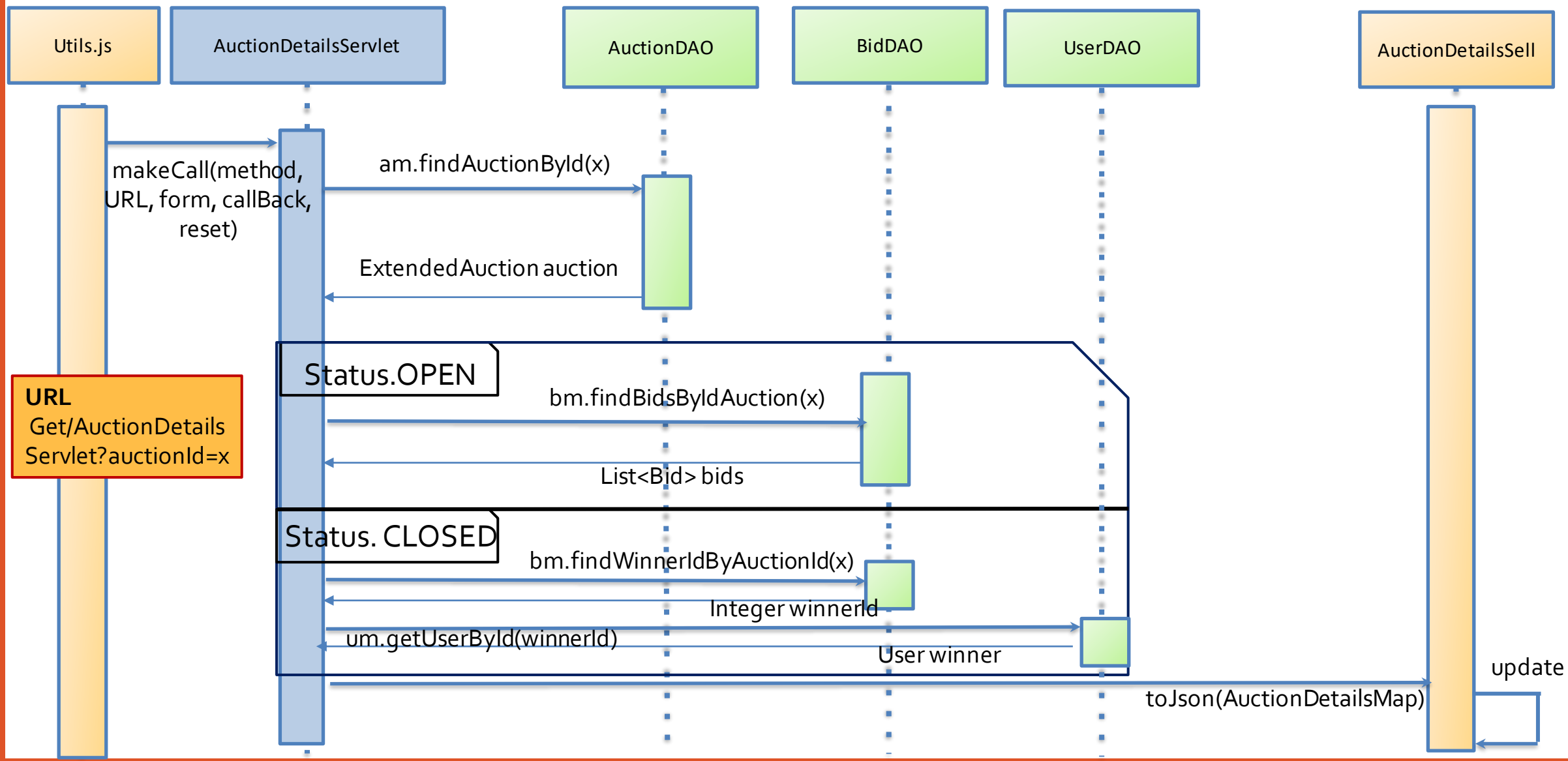


Sequence diagram (9/11)

Event: GET auction details (JS)

Client side

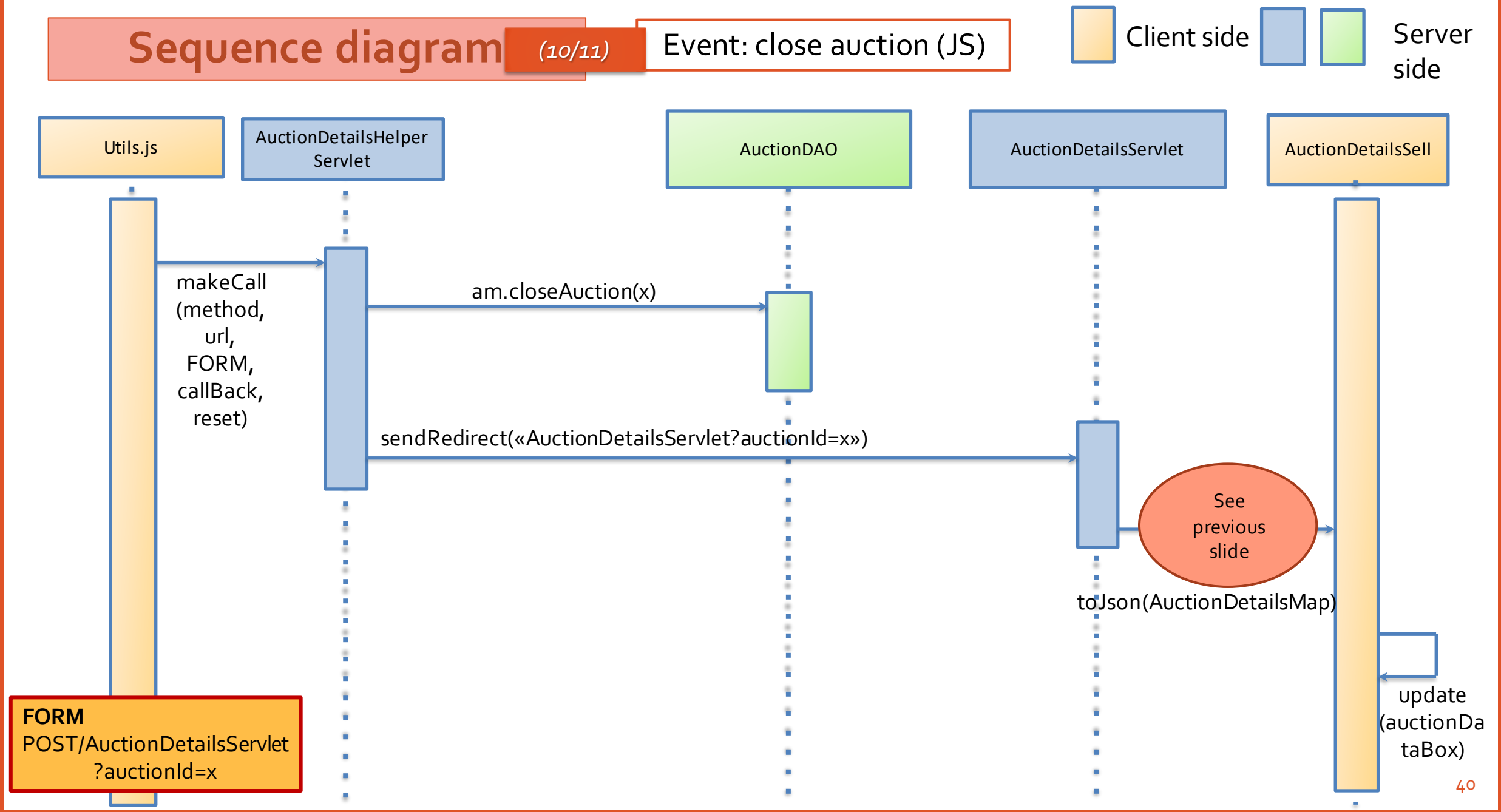
Server side



URL
Get/AuctionDetails Servlet?auctionId=x

Sequence diagram (10/11)

Event: close auction (JS)

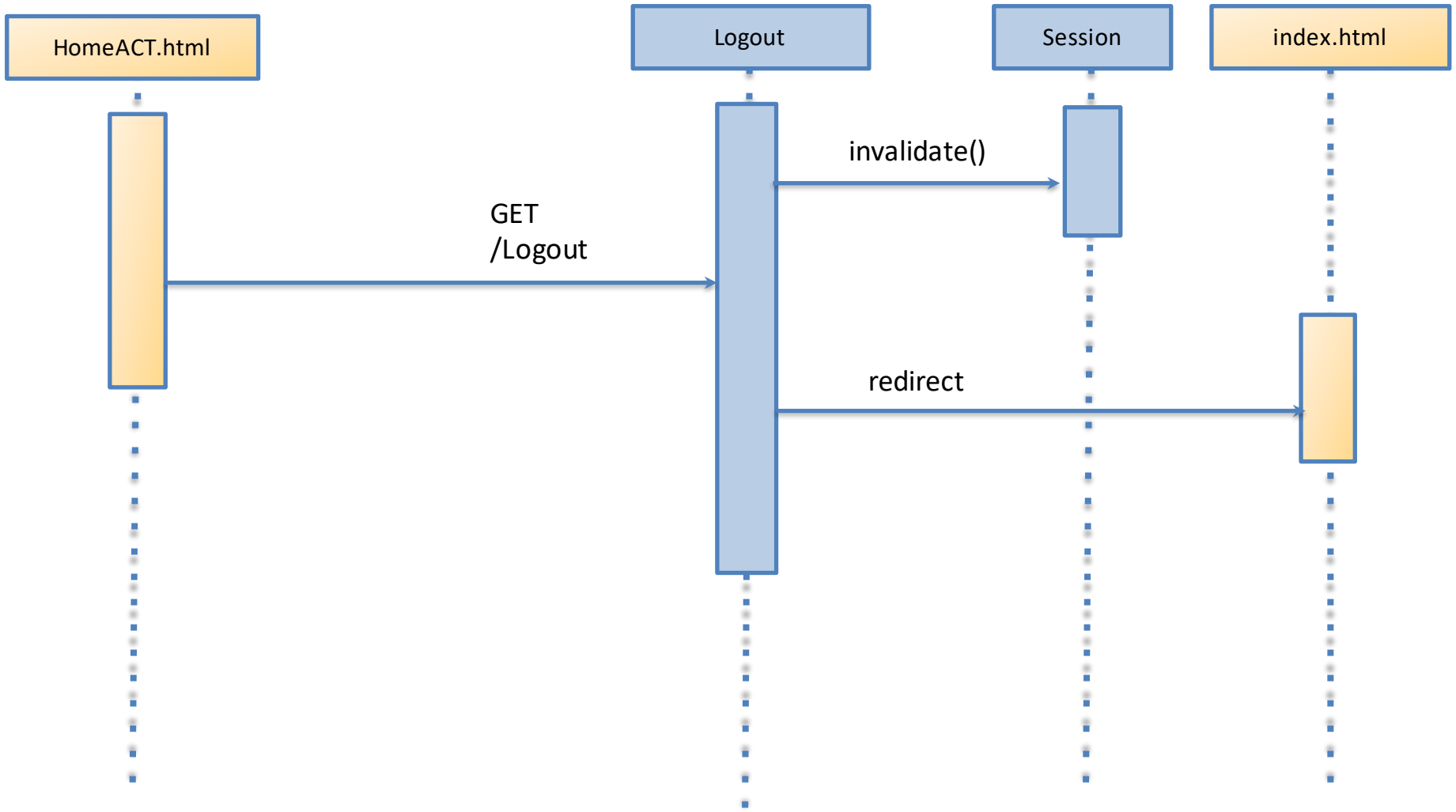


Sequence diagram (11/11)

Event: Logout (JS)

Client side

Server side



MARCO D'ANTINI

ALFREDO LANDI

LUIGI PICCOLI

```
response.getWriter().println("<p>Thank you</p>");
```

Image credits

- Page 1: auction browser by Vectorstall from the Noun Project