

# AuctiOnline

Tecnologie Informatiche per il Web – Aste Online – Gruppo 17

Pizzamiglio Giacomo (10620604)

Prisciantelli Andrea (10618568)

Romagnoni Lorenzo (10661108)

Prof. Piero FRATERNALI



**POLITECNICO**  
MILANO 1863

# Sommario

- Design del Database
- Versione HTML pura
- Versione JavaScript
- Divisione del lavoro

AuctiOnline

# Progettazione del Database

AuctiOnline

# Specifica

Un'applicazione web consente la gestione di aste online. Gli utenti 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.

# Specifica - Analisi

Entità

Viste

Un'applicazione web consente la gestione di aste online. Gli **utenti** 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.

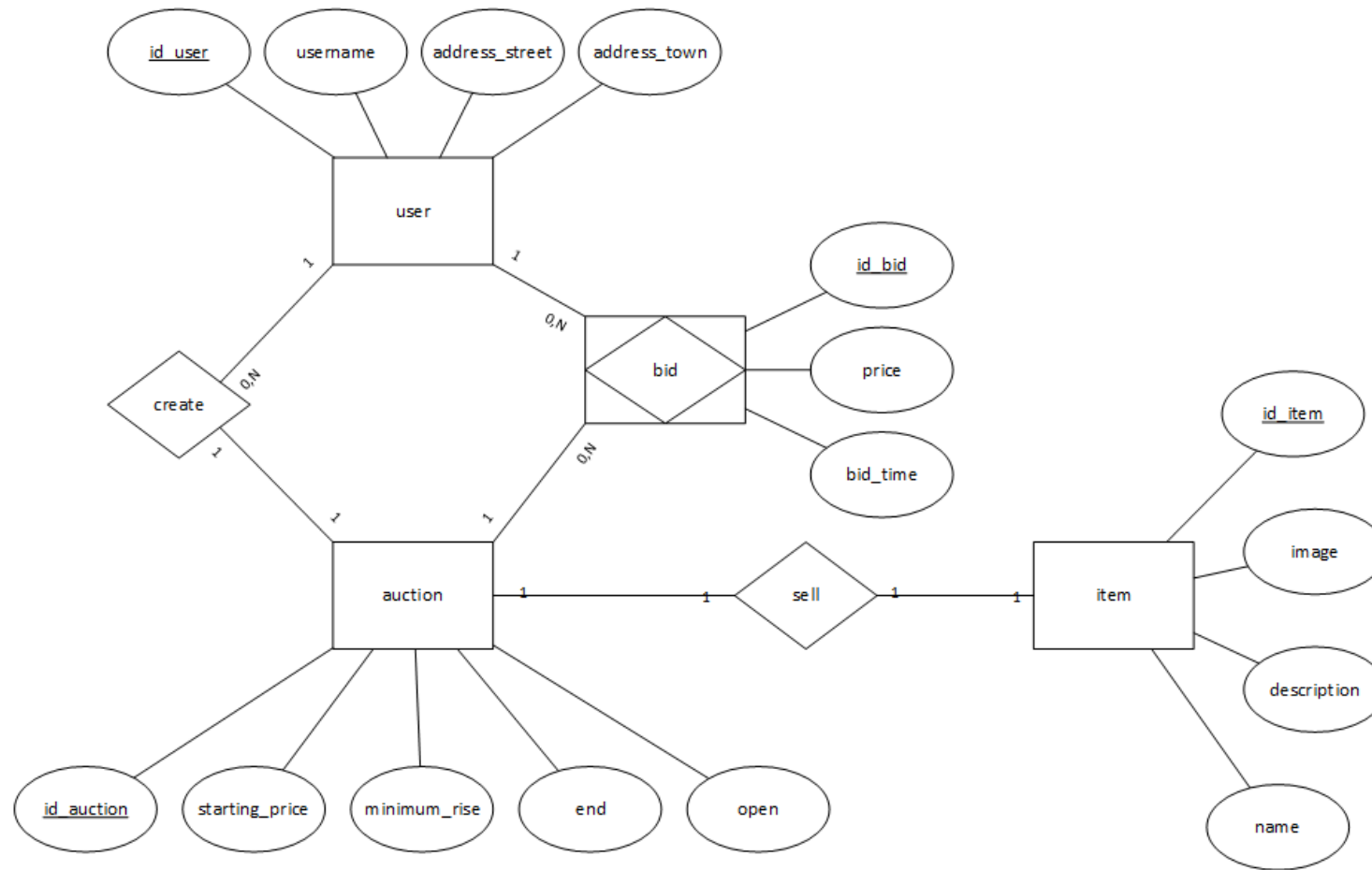
Cliccando su un'asta aperta compare la pagina OFFERTA che mostra i dati dell'articolo, l'elenco delle offerte pervenute in ordine di data+ora decrescente e un campo di input per inserire la propria offerta, che deve essere superiore all'offerta massima corrente. Dopo l'invio dell'offerta la pagina OFFERTA 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.

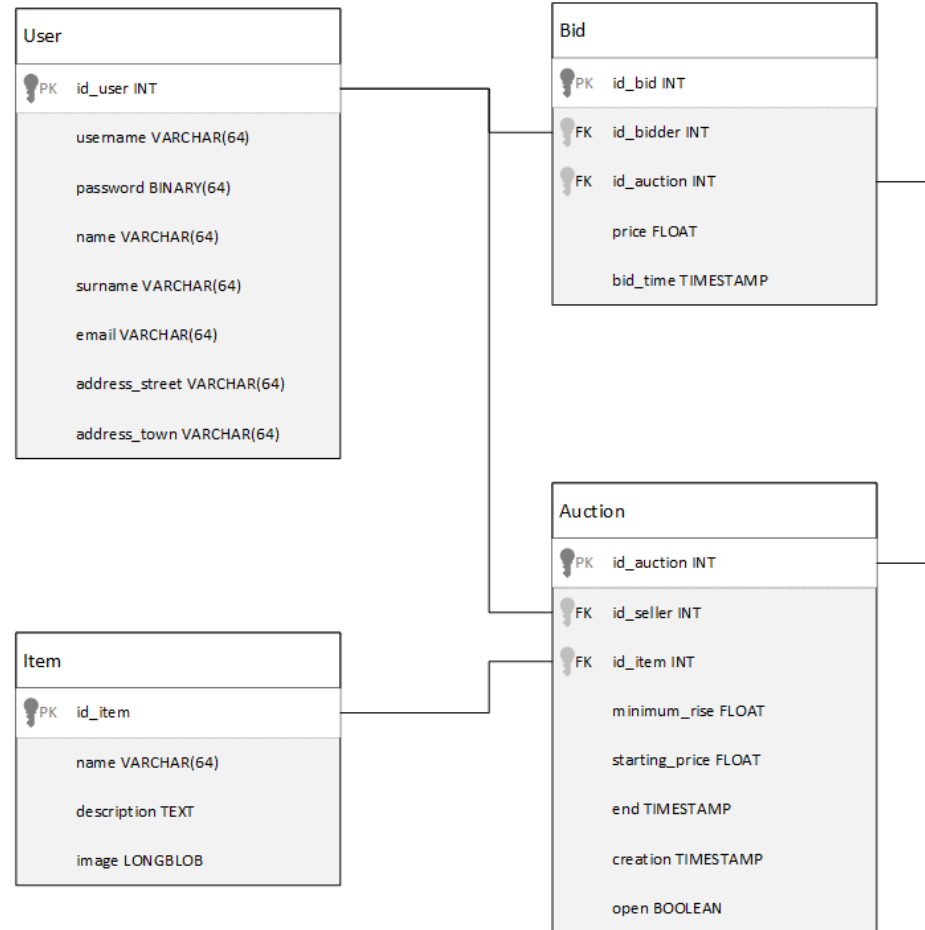
# Specifica – Scelte implementative

- L'entità utente comprende: id (chiave primaria), nome, cognome, email, indirizzo (strada e città), username e password
- Le entità asta e articolo sono trattate separatamente nonostante la relazione tra le due sia di cardinalità 1:1. Non è stato effettuato l'accorpamento delle due entità per garantire una buona scalabilità del progetto nel caso si vogliano aggiungere future funzioni (es. vendita di articoli multipli nel contesto di un'unica asta).
- Per aumentare il livello di sicurezza dell'applicazione si è deciso di immagazzinare le password nella base dati sotto forma di hash (algoritmo sha-256). In questo modo non dovrebbe essere possibile ricavare il loro valore in chiaro a partire dai dati memorizzati nel server.
- Per semplificare e organizzare le interrogazioni alla base di dati sono state definite alcune view per standardizzare le query più ricorrenti. Le view gestiscono anche l'ordinamento dei dati restituiti.

# Progettazione ER



# Schema





# Schema - User

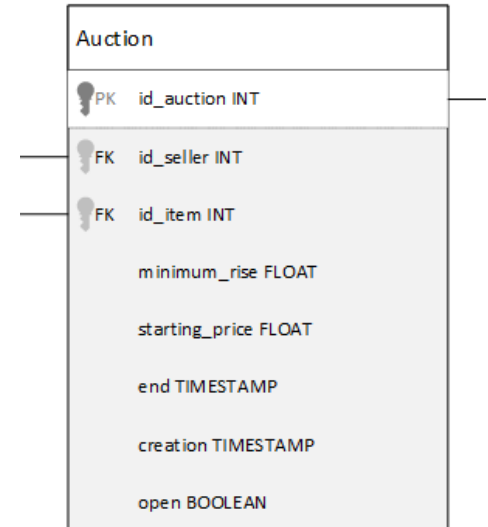
```
1 CREATE TABLE `user` (  
2   `id_user` int NOT NULL AUTO_INCREMENT,  
3   `username` varchar(64) NOT NULL,  
4   `password` binary(64) NOT NULL,  
5   `name` varchar(64) NOT NULL,  
6   `surname` varchar(64) NOT NULL,  
7   `email` varchar(64) NOT NULL,  
8   `address_street` varchar(64) NOT NULL,  
9   `address_town` varchar(64) NOT NULL,  
10  PRIMARY KEY (`id_user`),  
11  UNIQUE KEY `username_UNIQUE` (`username`)  
12 ) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

Siccome l'algoritmo sha-256 restituisce hash lunghi 128 caratteri, si è deciso di memorizzarli in modo più compatto come array di 64 BINARY, equivalenti a 64\*2 CHAR

User	
PK	id_user INT
	username VARCHAR(64)
	password BINARY(64)
	name VARCHAR(64)
	surname VARCHAR(64)
	email VARCHAR(64)
	address_street VARCHAR(64)
	address_town VARCHAR(64)


# Schema - Auction

```
1 CREATE TABLE `auction` (  
2   `id_auction` int NOT NULL AUTO_INCREMENT,  
3   `id_item` int NOT NULL,  
4   `id_seller` int NOT NULL,  
5   `minimum_rise` float unsigned NOT NULL,  
6   `starting_price` float unsigned NOT NULL,  
7   `end` timestamp NOT NULL,  
8   `creation` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,  
9   `open` tinyint NOT NULL DEFAULT '1',  
10  PRIMARY KEY (`id_auction`),  
11  KEY `id_seller_idx` (`id_seller`),  
12  KEY `id_item_idx` (`id_item`),  
13  CONSTRAINT `id_item` FOREIGN KEY (`id_item`) REFERENCES `item` (`id_item`) ON DELETE CASCADE ON UPDATE CASCADE,  
14  CONSTRAINT `id_seller` FOREIGN KEY (`id_seller`) REFERENCES `user` (`id_user`) ON UPDATE CASCADE  
15 ) ENGINE=InnoDB AUTO_INCREMENT=24 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```



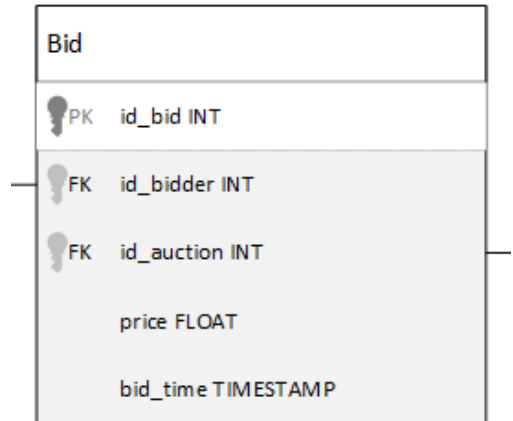
# Schema - Item

```
1 CREATE TABLE `item` (  
2   `id_item` int NOT NULL AUTO_INCREMENT,  
3   `name` varchar(128) NOT NULL,  
4   `description` text NOT NULL,  
5   `image` longblob NOT NULL,  
6   PRIMARY KEY (`id_item`)  
7 ) ENGINE=InnoDB AUTO_INCREMENT=25 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

Item	
 PK	id_item
name VARCHAR(64)	
description TEXT	
image LONGBLOB	

# Schema - Bid

```
1 CREATE TABLE `bid` (  
2   `id_bid` int NOT NULL AUTO_INCREMENT,  
3   `id_auction` int NOT NULL,  
4   `id_bidder` int NOT NULL,  
5   `price` float unsigned NOT NULL,  
6   `bid_time` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,  
7   PRIMARY KEY (`id_bid`),  
8   KEY `id_seller_idx` (`id_bidder`),  
9   KEY `id_auction_idx` (`id_auction`),  
10  CONSTRAINT `id_auction` FOREIGN KEY (`id_auction`) REFERENCES `auction` (`id_auction`) ON DELETE CASCADE ON UPDATE CASCADE,  
11  CONSTRAINT `id_bidder` FOREIGN KEY (`id_bidder`) REFERENCES `user` (`id_user`) ON UPDATE CASCADE  
12 ) ENGINE=InnoDB AUTO_INCREMENT=29 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```



# View – Aste & Articoli

```
2      ALGORITHM = UNDEFINED
3      DEFINER = `root`@`localhost`
4      SQL SECURITY DEFINER
5  VIEW `tiw-project`.`auction_item` AS
6      SELECT
7          `a`.`id_auction` AS `id_auction`,
8          `a`.`id_item` AS `id_item`,
9          `a`.`id_seller` AS `id_seller`,
10         `a`.`minimum_rise` AS `minimum_rise`,
11         `a`.`starting_price` AS `starting_price`,
12         `a`.`end` AS `end`,
13         `a`.`creation` AS `creation`,
14         `a`.`open` AS `open`,
15         `i`.`name` AS `name`,
16         `i`.`description` AS `description`,
17         `i`.`image` AS `image`
18  FROM
19      (`tiw-project`.`auction` `a`
20      JOIN `tiw-project`.`item` `i` ON ((`a`.`id_item` = `i`.`id_item`)))
21  ORDER BY `a`.`end` DESC;
```

Questa view è utilizzata a sua volta dalle altre view e serve a restituire le aste associate all'articolo corrispondente.

# View – Offerte Massime

```
1  CREATE
2      ALGORITHM = UNDEFINED
3      DEFINER = `root`@`localhost`
4      SQL SECURITY DEFINER
5  VIEW `tiw-project`.`max_bids` AS
6      SELECT
7          `b`.`id_bid` AS `id_bid`,
8          `b`.`id_auction` AS `id_auction`,
9          `b`.`id_bidder` AS `id_bidder`,
10         `b`.`price` AS `price`,
11         `b`.`bid_time` AS `bid_time`
12     FROM
13         `tiw-project`.`bid` `b`
14     WHERE
15         (`b`.`price` = (SELECT
16             MAX(`b2`.`price`)
17         FROM
18             `tiw-project`.`bid` `b2`
19         WHERE
20             (`b2`.`id_auction` = `b`.`id_auction`)))
```

Questa view è utilizzata a sua volta dalle altre view e serve a restituire per ogni asta l'offerta più alta correntemente.

# View – Dettagli aste

```
1 • CREATE
2     ALGORITHM = UNDEFINED
3     DEFINER = `root`@`localhost`
4     SQL SECURITY DEFINER
5     VIEW `tiw-project`.`auctions_details` AS
6     SELECT
7         `tiw-project`.`ai`.`id_auction` AS `id_auction`,
8         `tiw-project`.`ai`.`minimum_rise` AS `minimum_rise`,
9         `tiw-project`.`ai`.`starting_price` AS `starting_price`,
10        `tiw-project`.`ai`.`end` AS `end`,
11        `tiw-project`.`ai`.`creation` AS `creation`,
12        `tiw-project`.`ai`.`open` AS `open`,
13        `tiw-project`.`ai`.`id_seller` AS `id_seller`,
14        `tiw-project`.`ai`.`id_item` AS `id_item`,
15        `tiw-project`.`ai`.`name` AS `name`,
16        `tiw-project`.`ai`.`description` AS `description`,
17        `tiw-project`.`ai`.`image` AS `image`,
18        `b`.`id_bid` AS `id_bid`,
19        `b`.`id_bidder` AS `id_bidder`,
20        `b`.`price` AS `price`,
21        `b`.`bid_time` AS `bid_time`,
22        `u`.`username` AS `bidder_name`
23    FROM
24        ((`tiw-project`.`auction_item` `ai`
25         LEFT JOIN `tiw-project`.`bid` `b` ON ((`tiw-project`.`ai`.`id_auction` = `b`.`id_auction`)))
26         LEFT JOIN `tiw-project`.`user` `u` ON ((`u`.`id_user` = `b`.`id_bidder`)))
27    ORDER BY `b`.`bid_time` DESC;
```

Questa view è utilizzata dalle servlet per ottenere tutti i dati relativi alle aste, comprese tutte le offerte.

# View – Dettagli asta (riassunti)

```
1 • CREATE
2     ALGORITHM = UNDEFINED
3     DEFINER = `root`@`localhost`
4     SQL SECURITY DEFINER
5     VIEW `tiw-project`.`auctions_summary` AS
6     SELECT
7         `tiw-project`.`ai`.`id_auction` AS `id_auction`,
8         `tiw-project`.`ai`.`id_item` AS `id_item`,
9         `tiw-project`.`ai`.`id_seller` AS `id_seller`,
10        `tiw-project`.`ai`.`minimum_rise` AS `minimum_rise`,
11        `tiw-project`.`ai`.`starting_price` AS `starting_price`,
12        `tiw-project`.`ai`.`end` AS `end`,
13        `tiw-project`.`ai`.`creation` AS `creation`,
14        `tiw-project`.`ai`.`open` AS `open`,
15        `tiw-project`.`ai`.`name` AS `name`,
16        `tiw-project`.`ai`.`description` AS `description`,
17        `tiw-project`.`ai`.`image` AS `image`,
18        `tiw-project`.`mb`.`id_bid` AS `id_max_bid`,
19        `tiw-project`.`mb`.`id_bidder` AS `id_max_bidder`,
20        `tiw-project`.`mb`.`price` AS `max_price`,
21        `tiw-project`.`mb`.`bid_time` AS `max_bid_time`
22    FROM
23        (`tiw-project`.`auction_item` `ai`
24        LEFT JOIN `tiw-project`.`max_bids` `mb` ON (((`tiw-project`.`ai`.`id_auction` = `tiw-project`.`mb`.`id_auction`)))
25    ORDER BY `tiw-project`.`ai`.`end`;
```

Questa view è utilizzata dalle servlet per ottenere dati riassunti relative alle aste.



# Versione HTML pura

AuctiOnline

# Specifica - Analisi

Un'applicazione web consente la gestione di aste online. Gli utenti 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.

# Specifica – Scelte implementative

- La gestione di eventuali eccezioni propagate alle servlet è gestita da un *ErrorHandler*.
- Le azioni non autorizzate restituiscono un errore, gestito dall'*ErrorHandler*.
- Le richieste (POST/GET) mal formate sono trattate come azioni non autorizzate.
- Il bottone per chiudere un'asta aperta è visibile solo se è passata la sua scadenza. Dopo l'interazione con esso si è reindirizzati alla pagina VENDO.

# Componenti – Beans & Controllers

- **Model objects (Beans)**
  - DBObject (classe astratta per tutti gli oggetti)
  - Auction
  - Bid
  - Item
  - User
- **Controllers (servlets) [diritti di accesso]**
  - PerformLogin [not logged users]
  - PerformSignup [not logged users]
  - PerformLogout [logged users]
  - HomeController [logged users]
  - AuctionSubmissionController [logged users]
  - BidController [logged users]
  - BidSubmissionController [logged users]
  - BuyController [logged users]
  - CloseAuctionController [logged users]
  - DetailsController [logged users]
  - SellController [logged users]

# Componenti – DAO

- **Data Access Objects (DAO)**

- **AuctionDAO**

- filter(query, datetime) : List<Auction>
    - createItem(name, description, image): int
    - createAuction(itemId, sellerId, minimumRise, startingPrice, end, open): int
    - createAuctionItem(name, description, image, sellerId, minumumRise, startingPrice, end): void
    - addBidToAuction(auctionId, bidderId, price) : void
    - getUserWonAuctionsList(userId, timeReference) : List<Auction>
    - getUserAuctionLists(userId, timeReference) : Map<String, List<Auction>>
    - getUserOpenAuctions(userId, timeReference) : List<Auction>
    - getUserClosedAuctions(userId, timeReference) : List<Auction>
    - getAuctionDetails(auctionId, timeReference) : Auction
    - closeAuction(auctionId)

- **UserDAO**

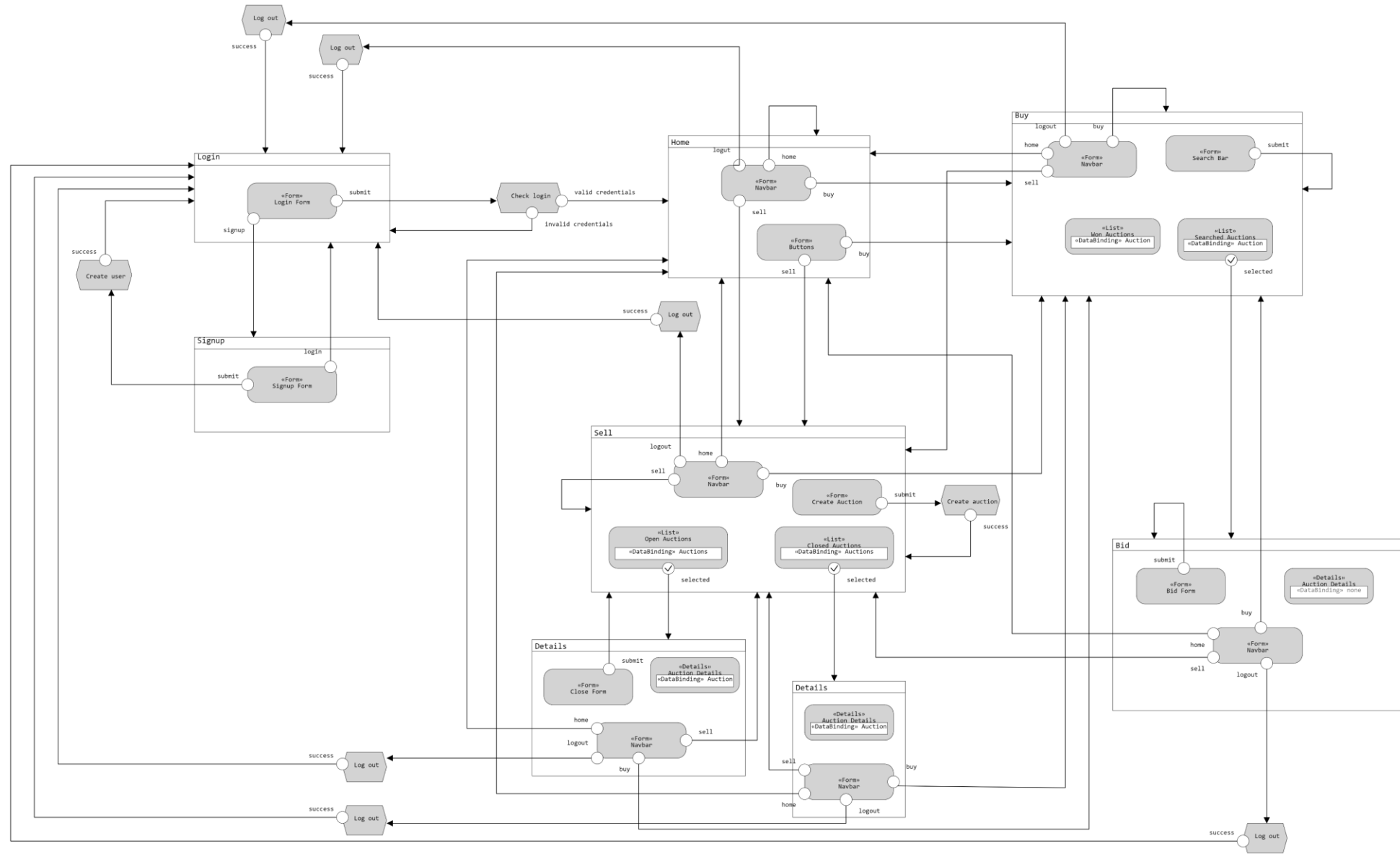
- performUserLogin(username, password) : User
    - createUser(username, password, name, surname, email, addressTown, addressStreet) : void
    - userAlreadyExists(username) : boolean

# Componenti - Templates

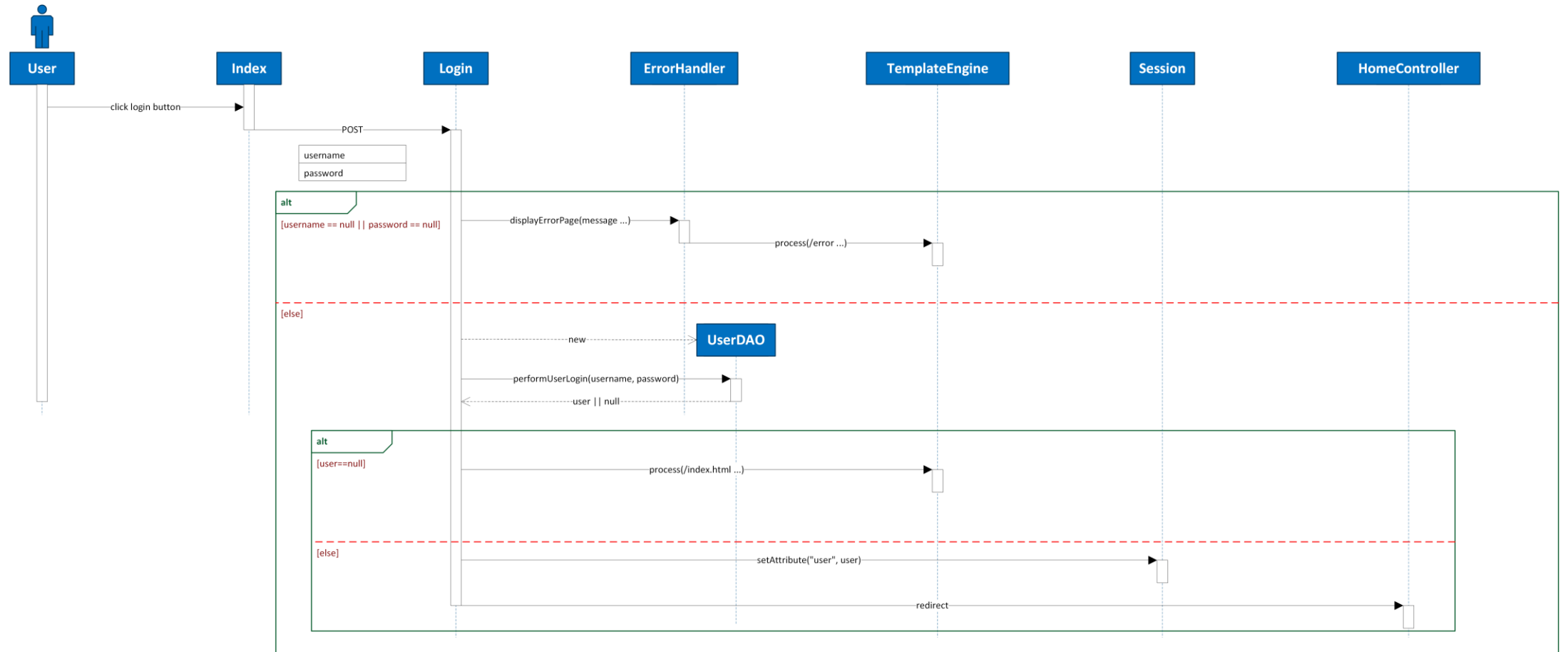
- **Templates (Views)**

- Index.html – login page
- Signup.html – create new user
- Home.html – website home page
- Buy.html – search auctions and view won auctions
- Auction.html – view auction details and make a bid
- Sell.html – view sold auctions and create new auctions
- Details.html – view open/closed auction details
- Error.html – display error page

# IFML – Design applicativo

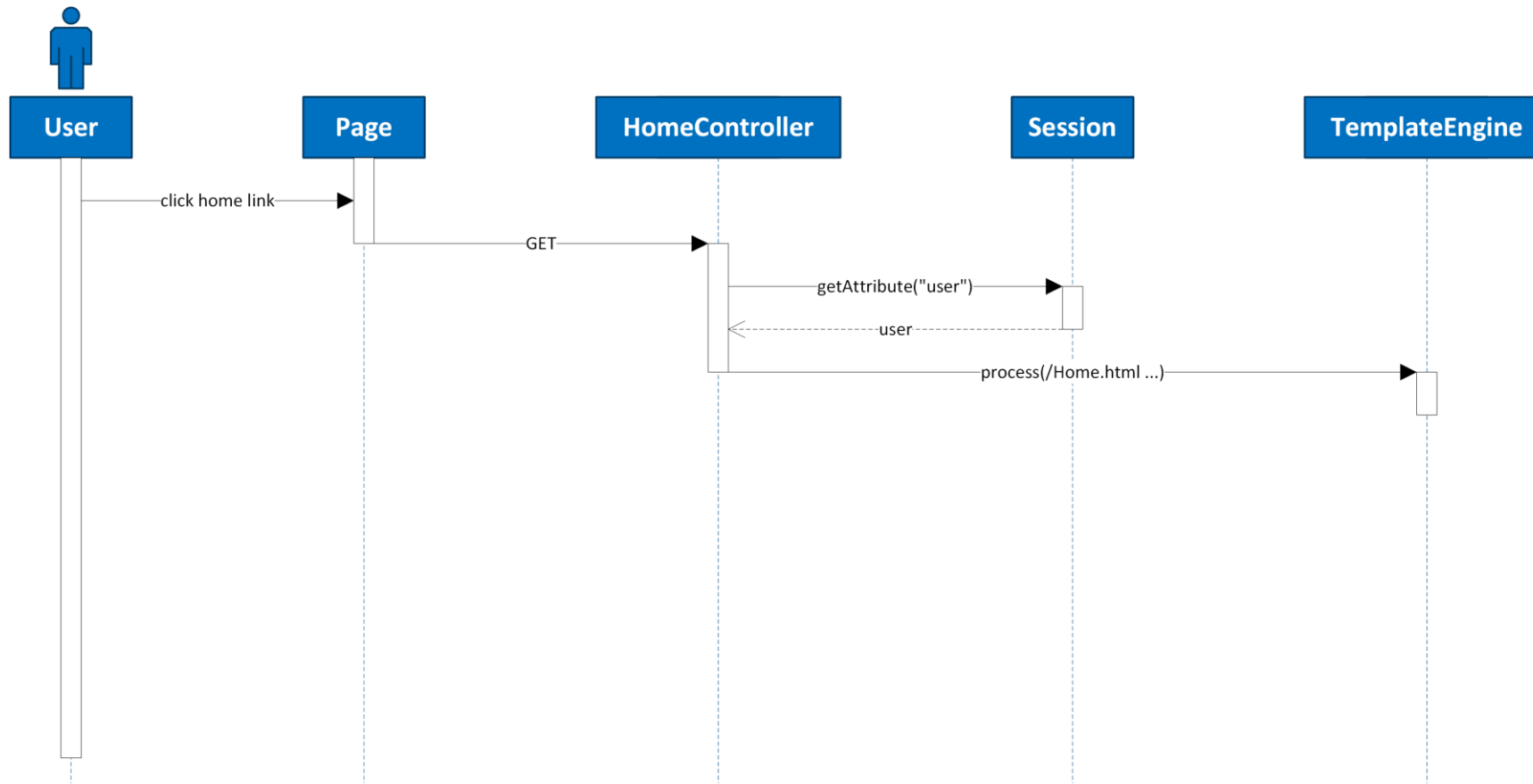


# Sequenza - Login

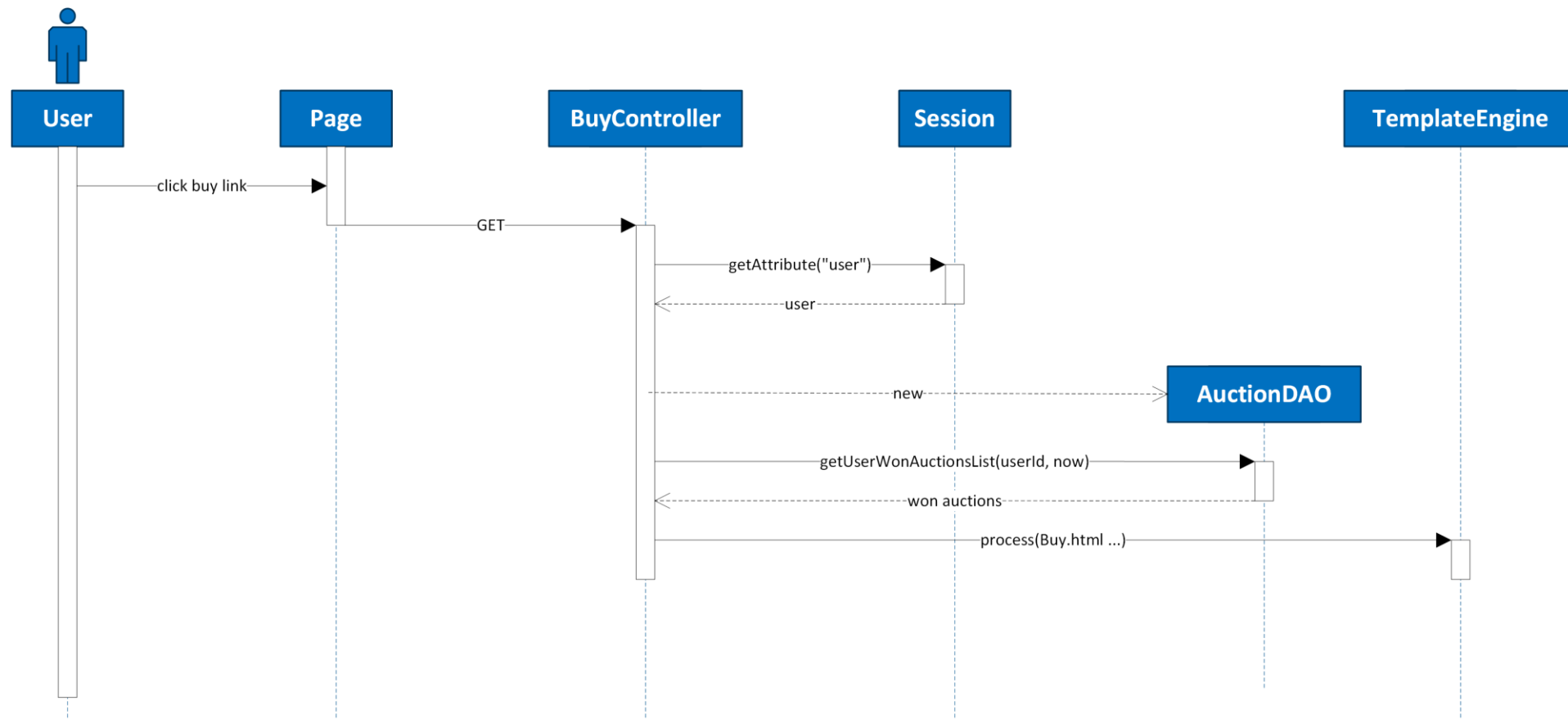




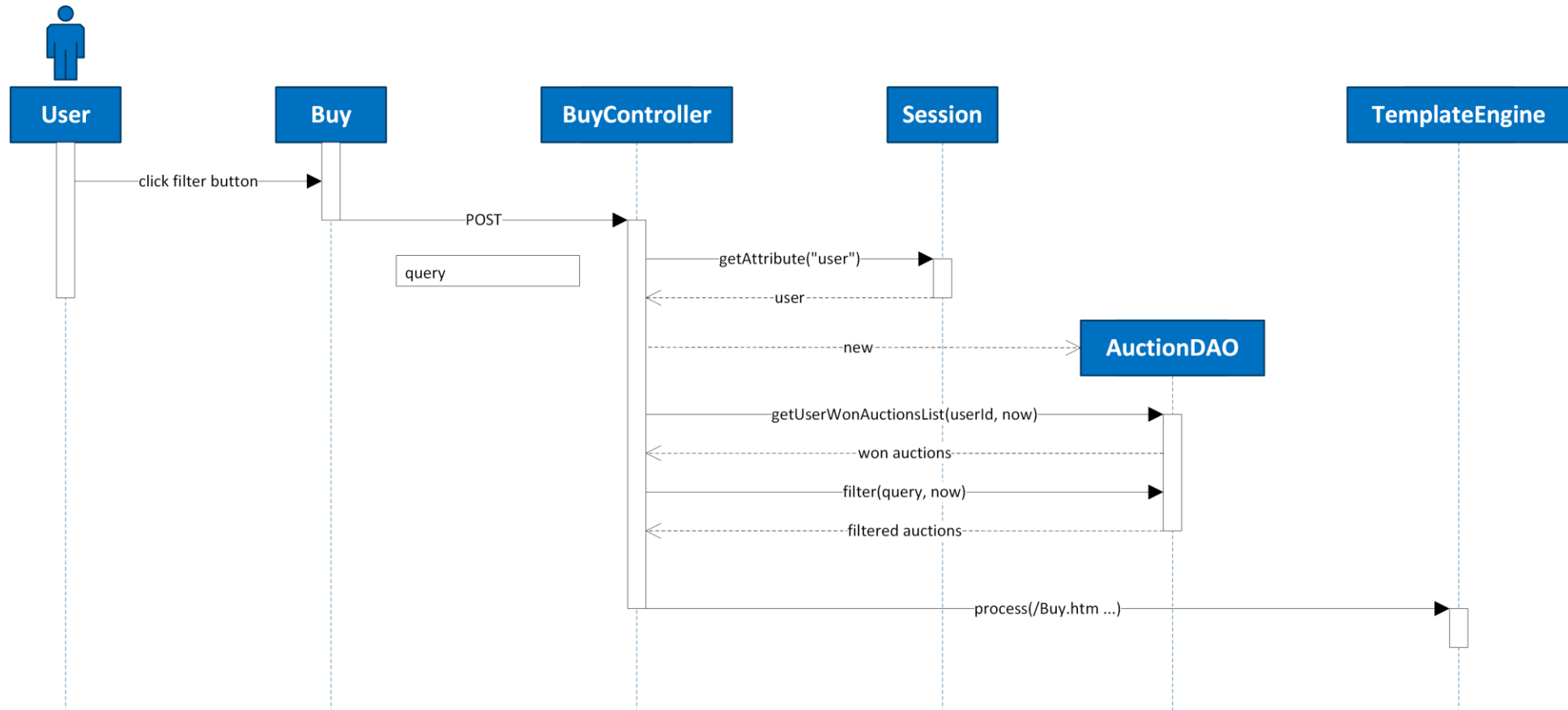
# Sequenza - Home



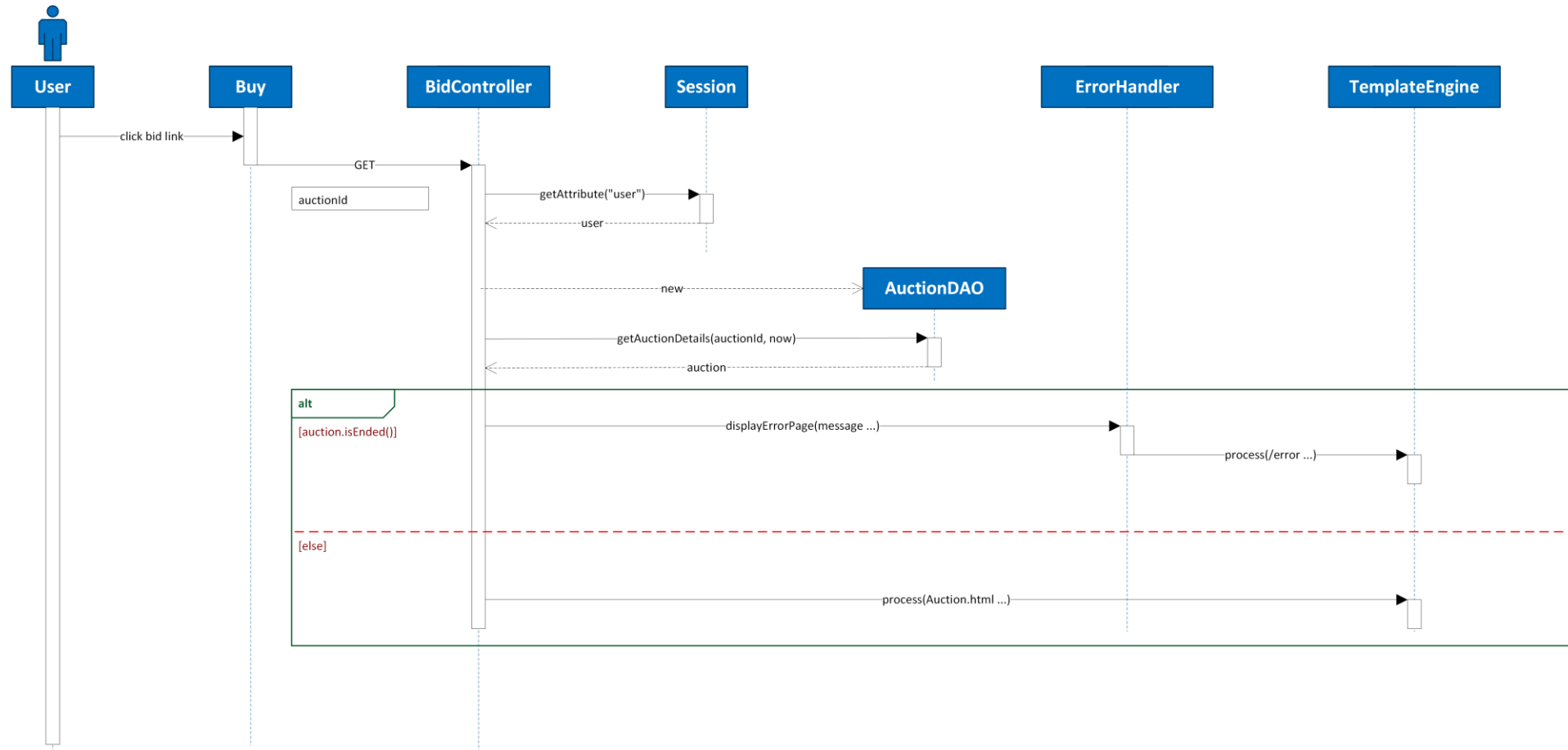
# Sequenza - Buy



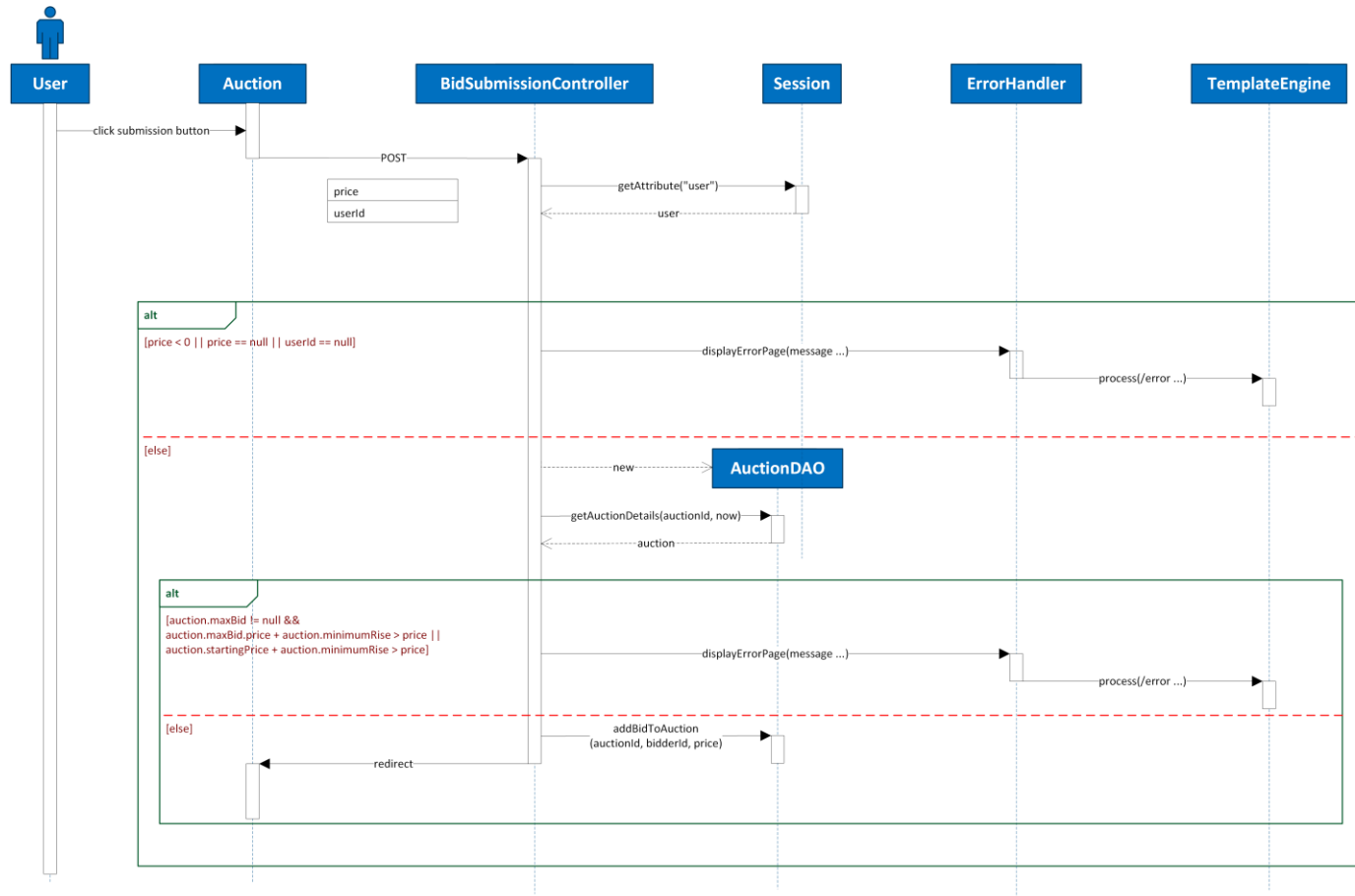
# Sequenza – Buy Filter



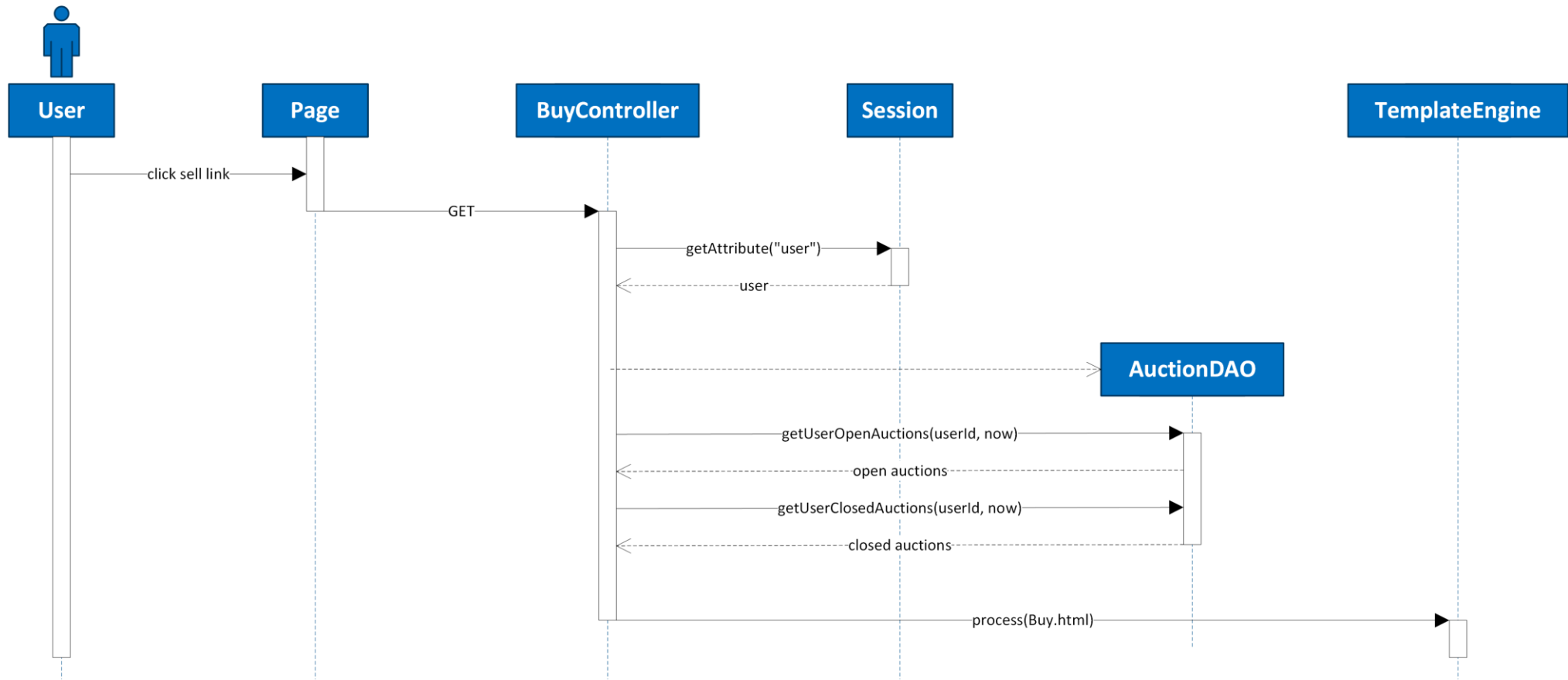
# Sequenza - Bid



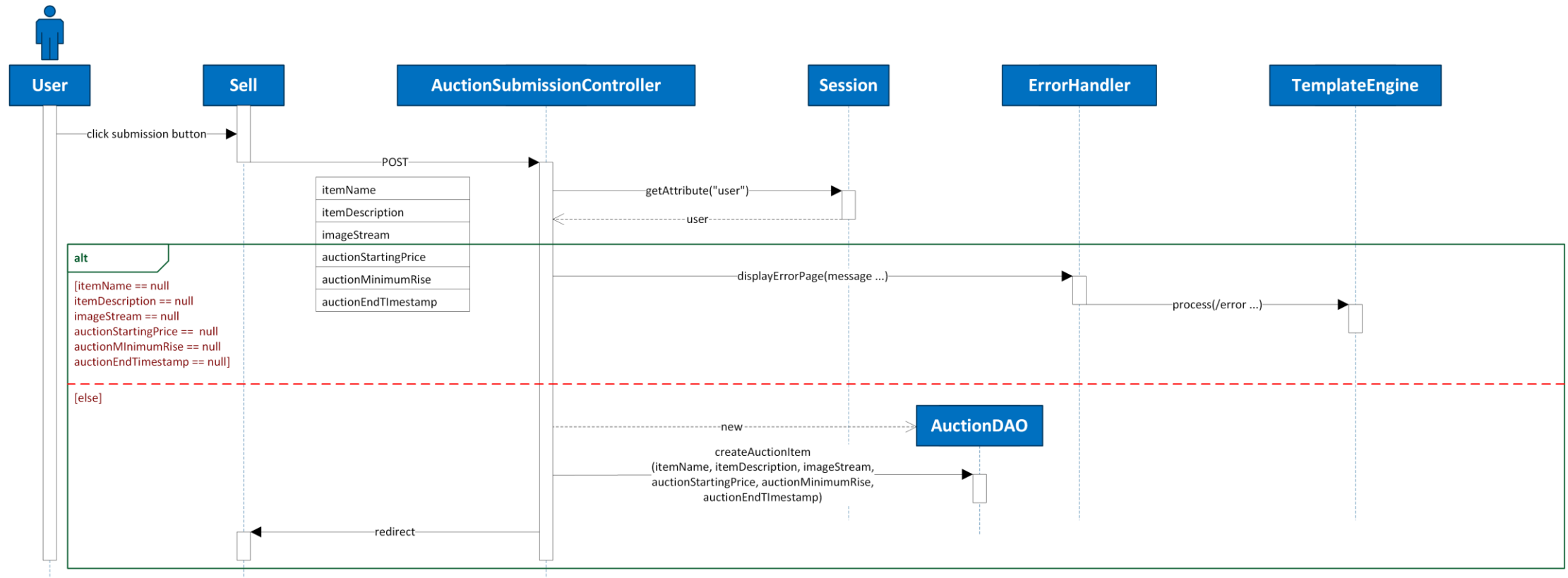
# Sequenza – Bid Submission



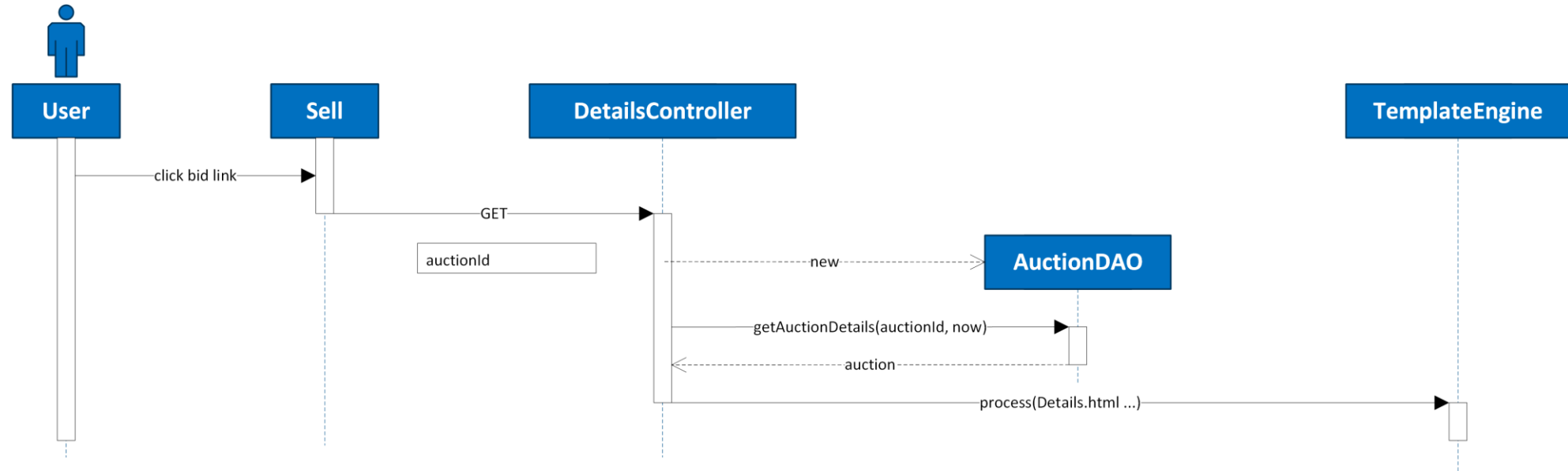
# Sequenza - Sell



# Sequenza – Create Auction

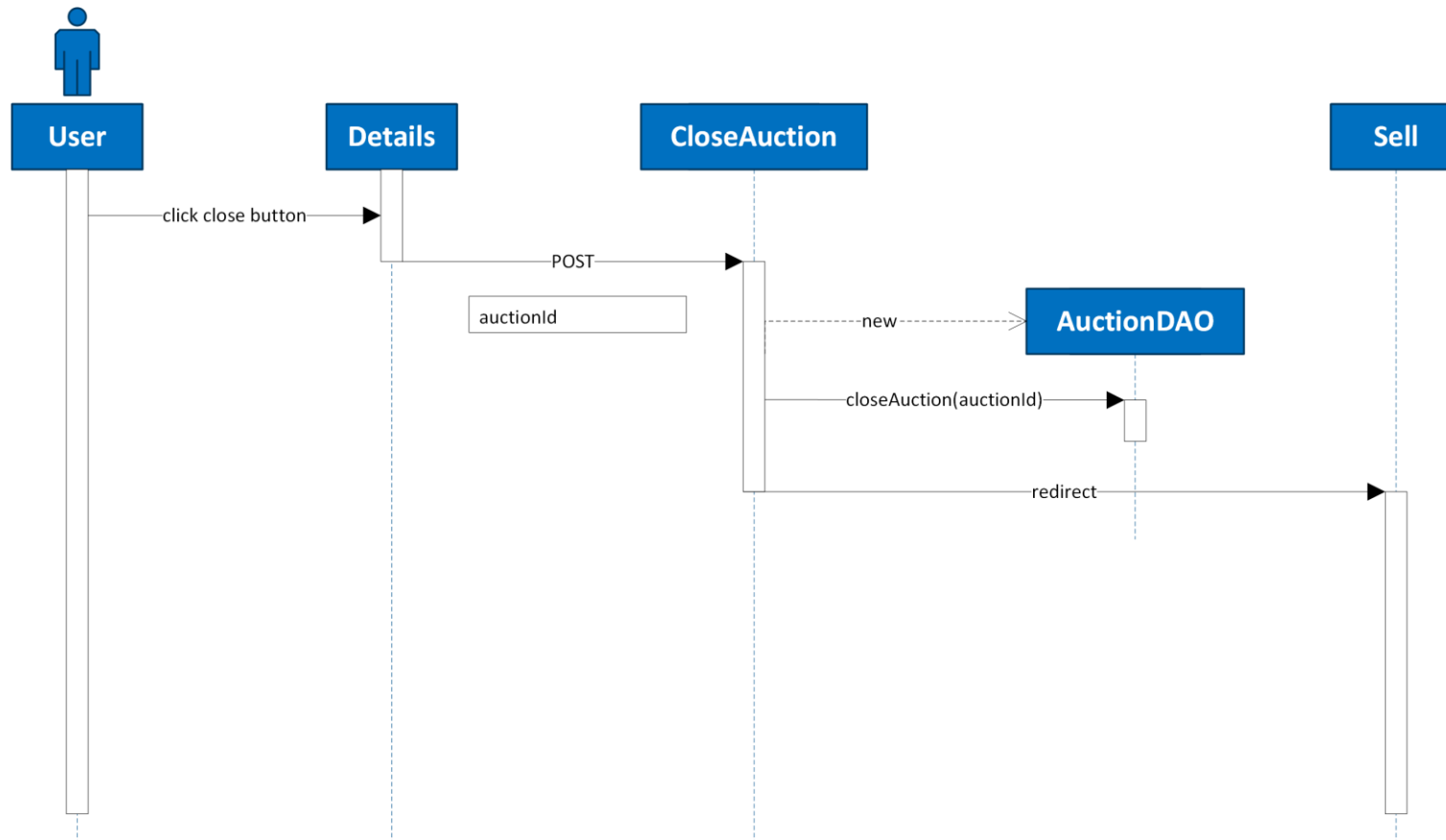


# Sequenza - Details

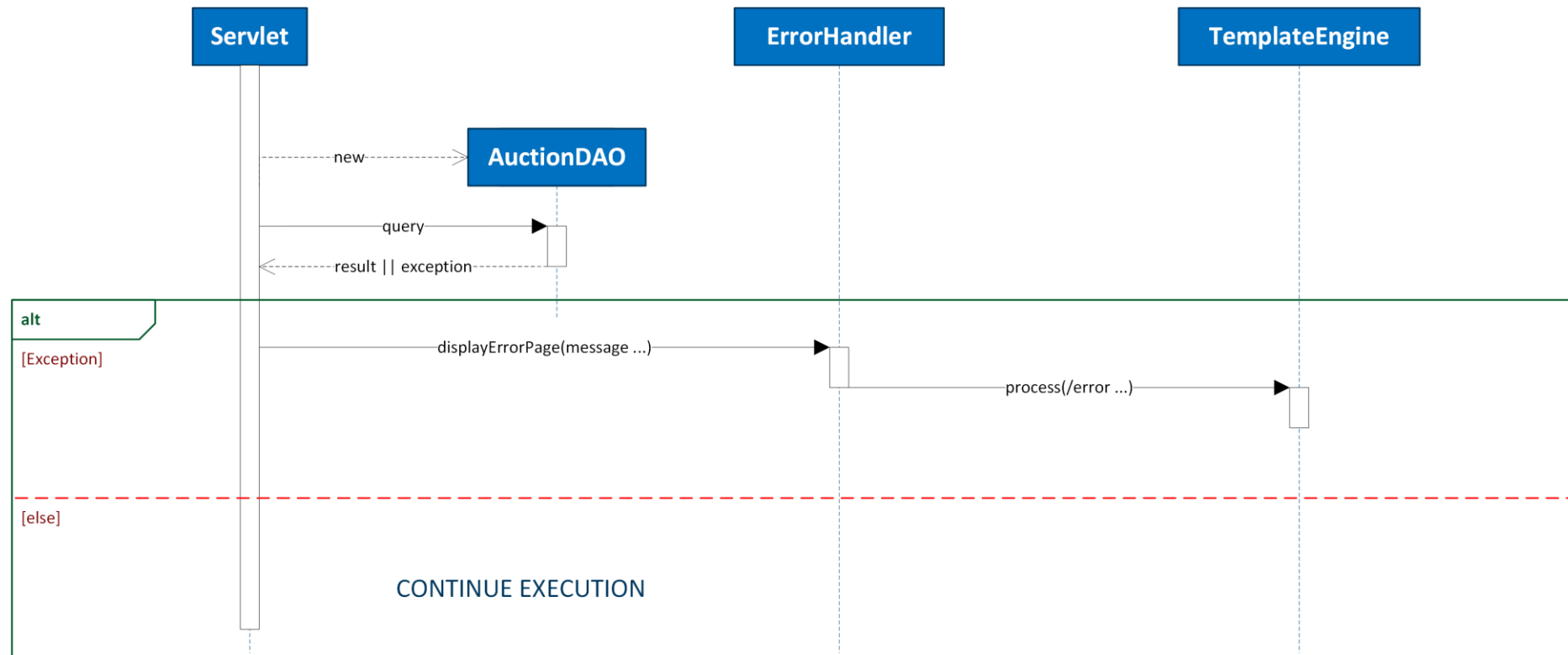




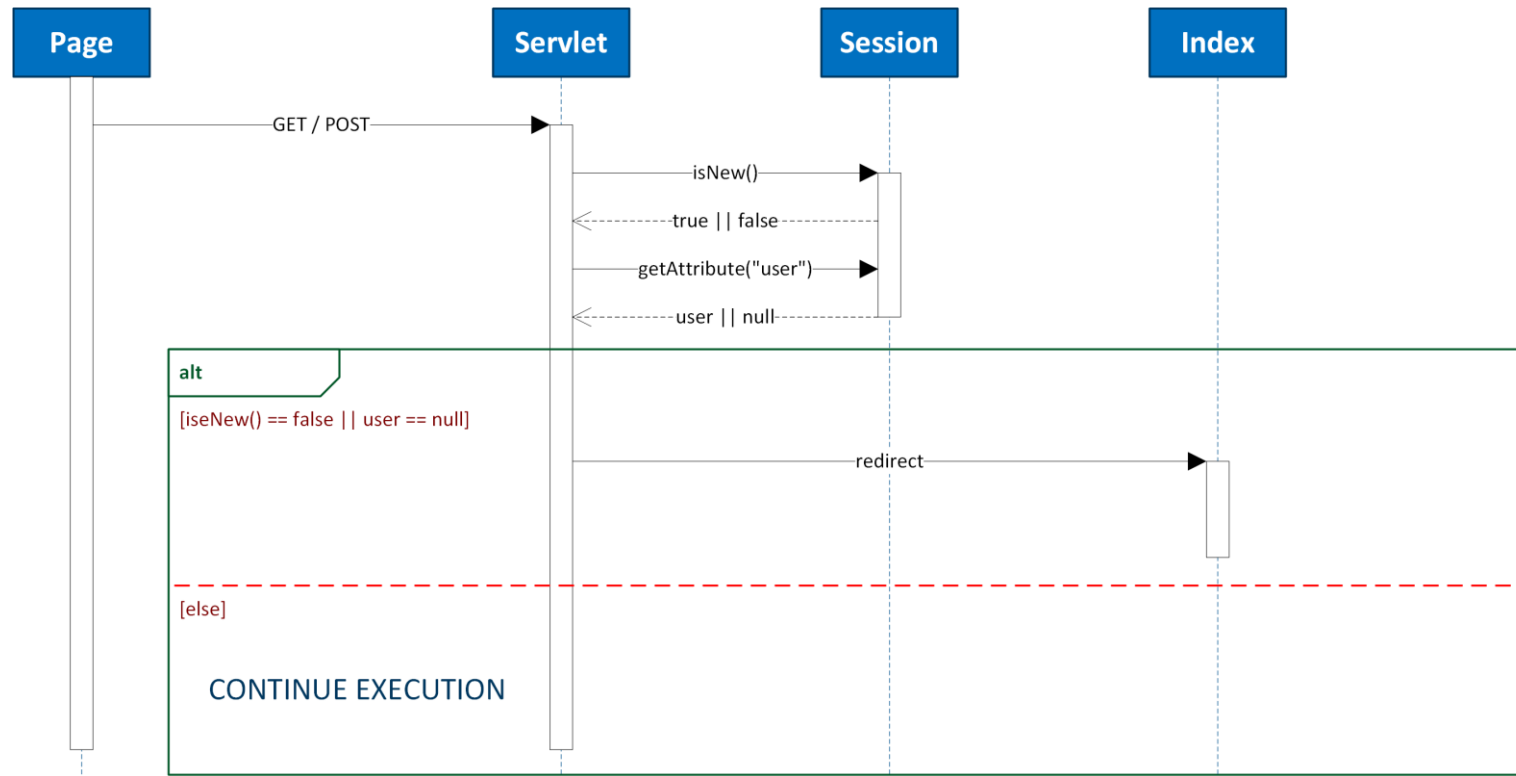
# Sequenza – Close Auction



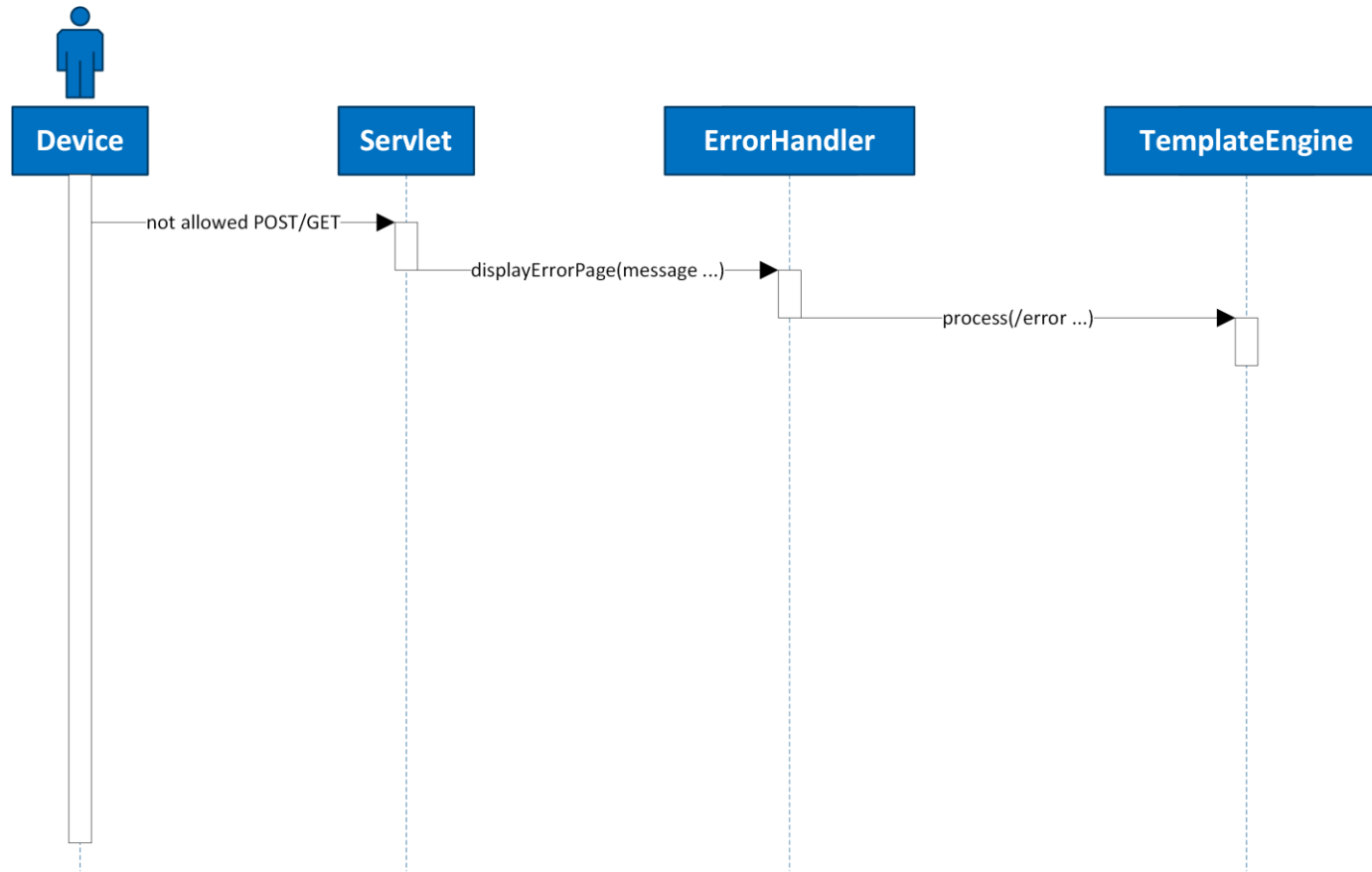
# Sequenza - Exception



# Sequenza - Session



# Sequenza – Request not allowed



# Versione JavaScript

AuctiOnline

# Specifica per la versione JavaScript

- Si realizzi un'applicazione client server web che estende e/o modifica le specifiche precedenti come segue:
  - 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.

# Specifica – Scelte implementative

- Per ogni utente viene salvata una coppia di cookie composta da «*user-history*» e «*user-action*» (dove *user* è l'username dell'utente), contenenti rispettivamente la lista degli ID delle aste visitate in precedenza e l'ultima azione svolta dall'utente. In questo modo è possibile permettere la navigazione nello stesso browser di più utenti.
- Gli errori da parte dell'utente durante l'inserimento di una offerta sono gestiti lato client, controllando in ogni form i parametri inseriti e ricaricando la pagina (senza eseguire alcuna azione lato server) in caso di errore.
- Durante l'inserimento di un'asta eventuali errori (parametri mancanti o non validi) vengono gestiti lato server. In caso di errore l'utente visualizza un messaggio, senza ricaricare la pagina.
- Non è possibile chiudere un'asta di cui non si è il venditore, e nemmeno inserire un'offerta da parte di un altro utente che non sia quello autenticato nella sessione.
- E' sempre possibile fare logout: in questo caso si viene re-indirizzati nella pagina di login.
- Anche in seguito ad un logout i cookies si conservano lato utente (rimanendo nei limiti dei 30 giorni). Questo permette all'utente di poter visualizzare il proprio storico anche in un successivo login.
- E' stato gestito il caso in cui l'utente forzi l'aggiornamento della pagina.

# Componenti – Beans & Controllers

- **Model objects (Beans) – uguali per le versioni «Pure HTML» e «Javascript»**
  - DBObject (classe astratta per tutti gli oggetti)
  - Auction
  - Bid
  - Item
  - User
- **Controllers (servlets) [diritti di accesso]**
  - PerformLogin [not logged users]
  - PerformSignup [not logged users]
  - Logout [logged users]
  - AuctionDetailsController [logged users]
  - AuctionSubmissionController [logged users]
  - BidSubmissionController [logged users]
  - CloseAuctionController [logged users]
  - ClosedAuctionsController [logged users]
  - FilteredAuctionsController [logged users]
  - HistoryAuctionsController [logged users]
  - MakeOfferController [logged users]
  - OpenAuctionsController [logged users]
  - WinnerController [logged users]
  - WonAuctionsController [logged users]



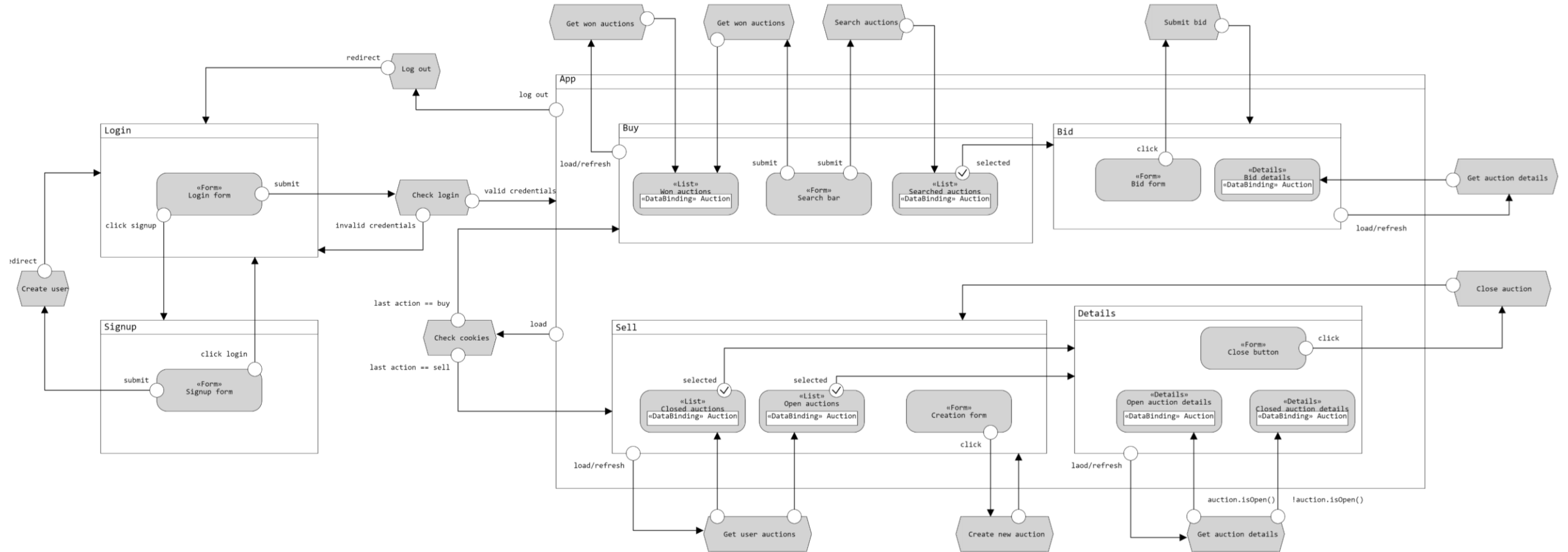
# Componenti – DAO

- **Data Access Objects (DAO)**
  - **AuctionDAO**
    - filterArticleByName(query, datetime) : List<Auction>
    - createItem(name, description, image) : int
    - createAuction(itemId, sellerId, minimumRise, startingPrice, end, open) : int
    - createAuctionItem(name, description, image, sellerId, minimumRise, startingRise, end) : void
    - addBidToAuction(auctionId, bidderId, price) : void
    - getWonAuctionsList(userId, timeReference) : List<Auction>
    - getUserAuctionsLists(userId, timeReference) : Map<String, List<Auction>>
    - getUserOpenAuctions(userId, timeReference) : List<Auction>
    - getUserCloseAuctions(userId, timeReference) : List<Auction>
    - getAuctionDetails(auctionId, timeReference) : Auction
    - closeAuction(auctionid) : void
    - getUserAuctionsByList(auctionsList, datetime) : List<Auction> - used for fetching history
  - **UserDAO**
    - performUserLogin(username, password) : User
    - createUser(username, password, name, surname, email, addressTown, addressStreet) : void
    - userAlreadyExists(username) : boolean
    - getUser(userId) : User

# Componenti - Templates

- **Templates (Views)**
  - Index.html – login page
  - Signup.html – create new user
  - App.html – website main page

# IFML – Design applicativo



# Events & Actions

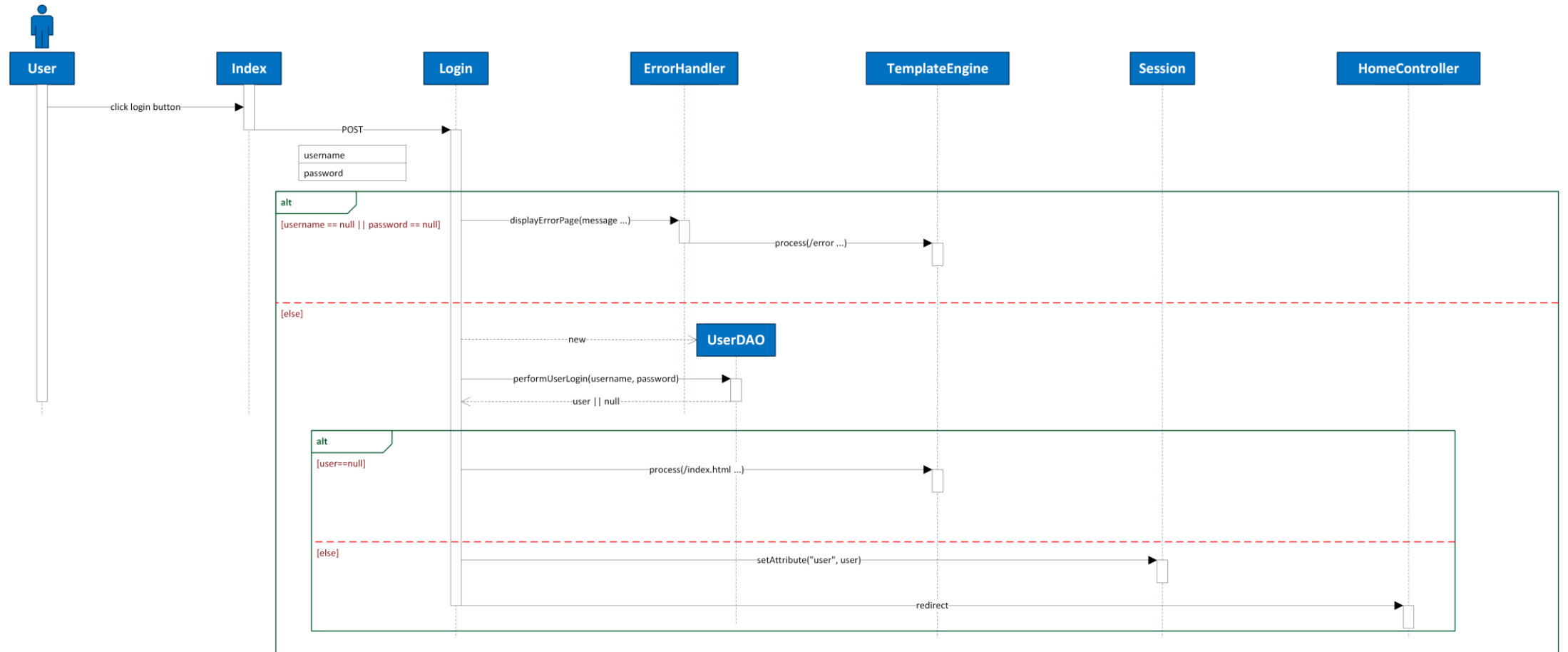
Client		Server	
EVENTO	AZIONE	EVENTO	AZIONE
Index.html > login form > submit	Controllo dei dati	POST (Username, password)	Controllo credenziali
Signup.html > signup form > submit	Controllo dei dati	POST (Informazioni sull'utente)	Creazione nuovo utente
Home.html > load > visualizzazione della pagina ACQUISTO > visualizzazione aste vinte	Aggiorna la view e visualizza le aste vinte dall'utente	GET (Informazioni sulle aste vinte)	Acquisizione dei dati dal DB e conversione in Json
Home.html > load > visualizzazione della pagina ACQUISTO > visualizzazione della cronologia dell'utente	Aggiorna la view e visualizza le aste cercate in precedenza dall'utente e ancora aperte	GET (Informazioni sulle aste della cronologia dell'utente)	Acquisizione dei dati dal DB e conversione in Json
Home.html > load > visualizzazione della pagina VENDO > visualizzazione aste aperte	Aggiorna la view e visualizza la lista delle aste create dall'utente ancora aperte	GET (Informazioni sulle aste aperte create dall'utente)	Acquisizione dei dati dal DB e conversione in Json
Home.html > load > visualizzazione della pagina VENDO > visualizzazione aste chiuse	Aggiorna la view e visualizza la lista delle aste chiuse create dall'utente	GET (Informazioni sulle aste chiuse dall'utente)	Acquisizione dei dati dal DB e conversione in Json
Home.html > filter form > submit	Aggiorna la view visualizzando le aste che soddisfano la query di ricerca	POST (Query di ricerca)	Acquisizione dei dati dal DB e conversione in Json delle aste che corrispondono alla query di ricerca
Home.html > create auction form > submit	Controllo dei dati	POST (Informazioni sull'asta)	Creazione dell'asta
Home.html > create bid form > submit	Controllo dei dati	POST (Dettagli dell'offerta e dell'asta)	Controllo e creazione dell'offerta per la specifica asta
Home.html > close auction > submit	Invio AJAX	POST (ID dell'asta da chiudere)	Controllo e chiusura dell'asta
Home.html > filtered auctions > auction selection	Aggiorna la view e visualizza i dettagli dell'asta selezionata (comprese le offerte)	GET (Informazioni sull'asta e le offerte)	Acquisizione dei dati dal DB e conversione in Json
Home.html > auctions from history > auction selection	Aggiorna la view visualizzando le aste precedentemente viste dall'utente e ancora aperte	GET (Elenco delle aste visualizzate in precedenza e ancora aperte)	Acquisizione dei dati dal DB e conversione in Json
Home.html > user open auctions list > auction selection	Aggiorna la view e visualizza i dettagli per l'asta creata dall'utente e l'elenco delle offerte	GET (Informazioni sull'asta e le offerte)	Acquisizione dei dati dal DB e conversione in Json
Home.html > user closed auctions list > auction selection	Aggiorna la view e visualizza i dettagli dell'asta e l'eventuale vincitore	GET (Informazioni sull'asta e sul vincitore)	Acquisizione dei dati dal DB e conversione in Json

# Controllers & Event Handlers

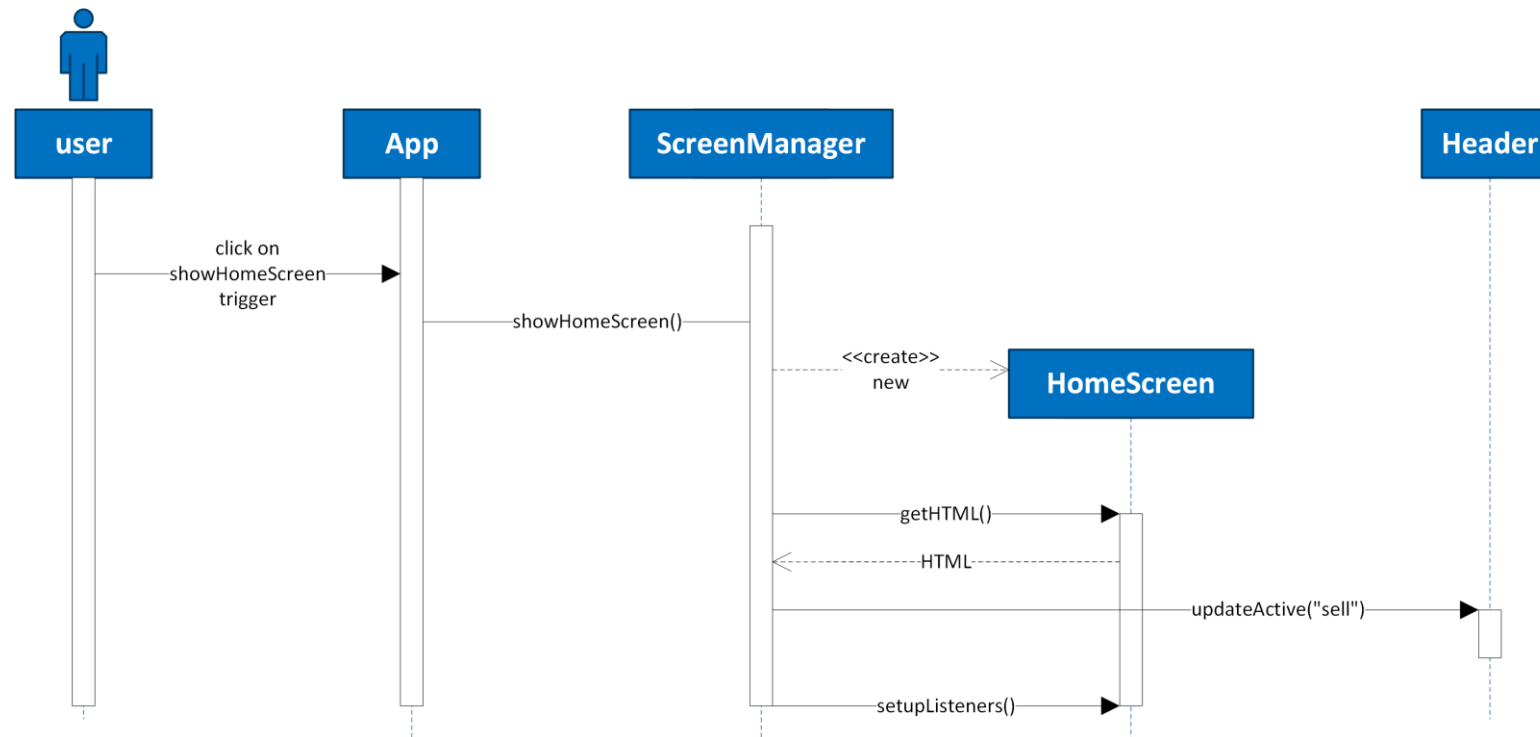
Nota: ogni qualvolta sia necessario comunicare con il server vengono chiamate le funzioni `HttpRequest` o `postRequest` contenute in «utils» per inviare le richieste asincrone

Client		Server	
EVENTO	CONTROLLORE	EVENTO	CONTROLLORE
Index.html > login form > submit	(la chiamata al server non viene effettuata attraverso Javascript)	POST (Username, password)	PerformLogin (servlet)
Signup.html > signup form > submit	(la chiamata al server non viene effettuata attraverso Javascript)	POST (Informazioni sull'utente)	PerformSignup (servlet)
Home.html > load > visualizzazione della pagina ACQUISTO > visualizzazione aste vinte	Function BuyScreen	GET (Informazioni sulle aste vinte)	WonAuctionsController (servlet)
Home.html > load > visualizzazione della pagina ACQUISTO > visualizzazione della cronologia dell'utente	Function BuyScreen	GET (Informazioni sulle aste della cronologia dell'utente)	HistoryAuctionsController (servlet)
Home.html > load > visualizzazione della pagina VENDO > visualizzazione aste aperte	Function SellScreen	GET (Informazioni sulle aste aperte create dall'utente)	OpenAuctionsController (servlet)
Home.html > load > visualizzazione della pagina VENDO > visualizzazione aste chiuse	Function SellScreen	GET (Informazioni sulle aste chiuse dall'utente)	ClosedAuctionsController (servlet)
Home.html > filter form > submit	Function BuyScreen	POST (Query di ricerca)	FilteredAuctionsController (servlet)
Home.html > create auction form > submit	Function SellScreen	POST (Informazioni sull'asta)	AuctionSubmissionController (servlet)
Home.html > create bid form > submit	Function MakeOfferItem	POST (Dettagli dell'offerta e dell'asta)	BidSubmissionController (servlet)
Home.html > close auction > submit	Function DetailsScreen	POST (ID dell'asta da chiudere)	CloseAuctionController (servlet)
Home.html > filtered auctions > auction selection	Function ScreenManager -> MakeOfferScreen	GET (Informazioni sull'asta e le offerte)	MakeOfferController (servlet)
Home.html > auctions from history > auction selection	Function ScreenManager -> MakeOfferScreen	GET (Elenco delle aste visualizzate in precedenza e ancora aperte)	MakeOfferController (servlet)
Home.html > user open auctions list > auction selection	Function ScreenManager -> DetailsScreen	GET (Informazioni sull'asta e le offerte)	AuctionDetailsController (servlet)
Home.html > user closed auctions list > auction selection	Function ScreenManager -> DetailsScreen	GET (Informazioni sull'asta e sul vincitore)	AuctionDetailsController (servlet) WinnerController (servlet)

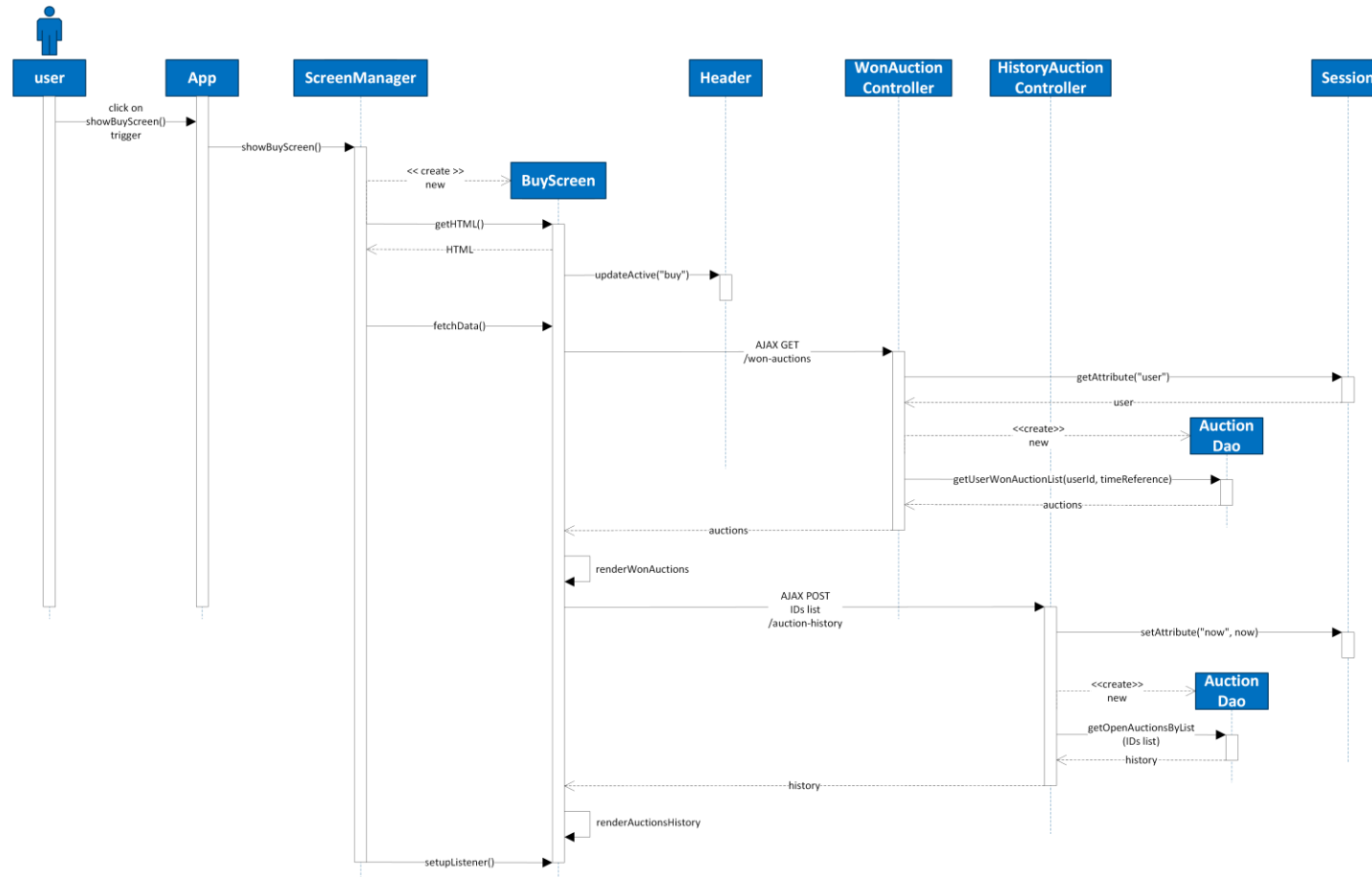
# Sequenza - Login



# Sequenza - Home

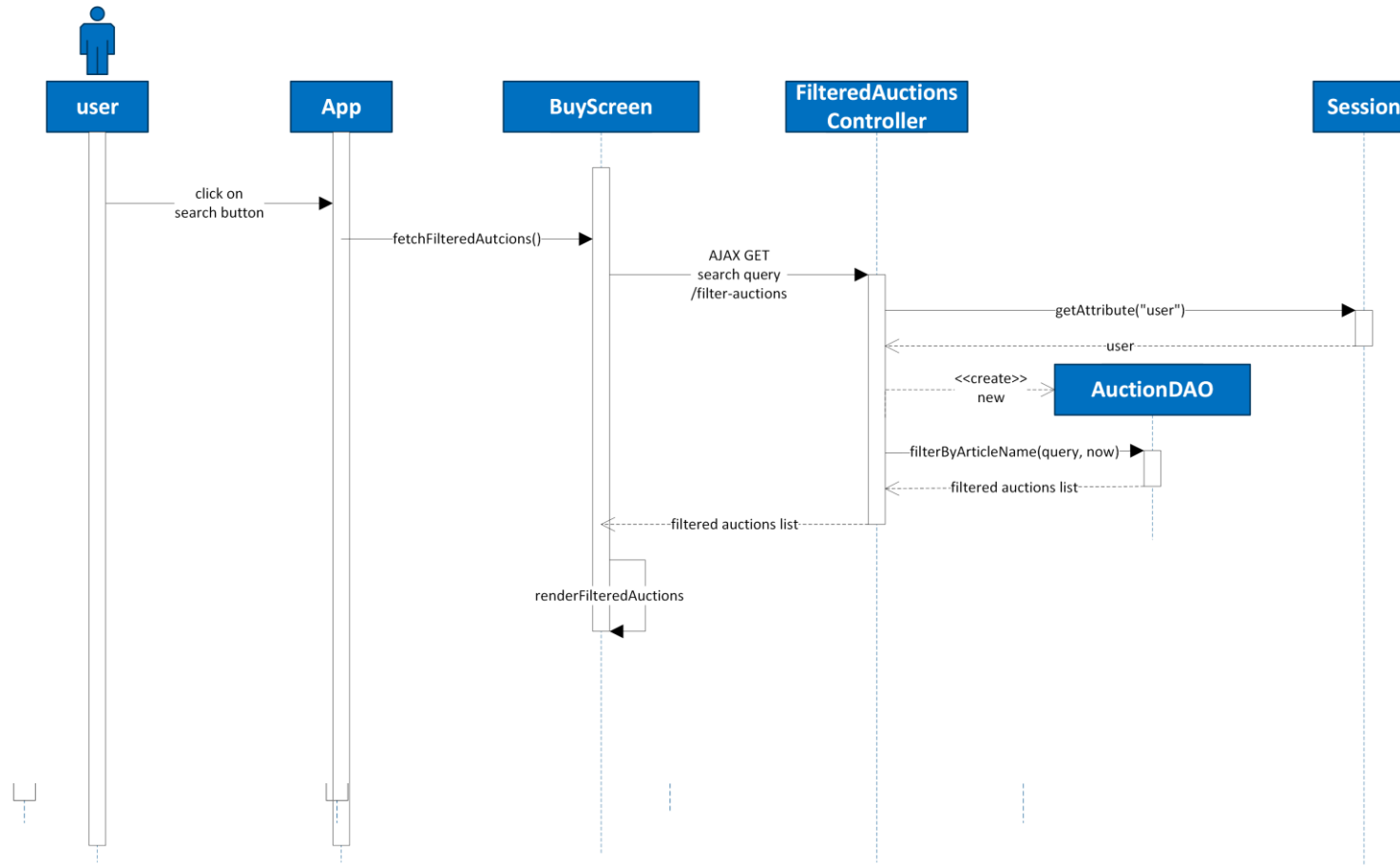


# Sequenza - Buy

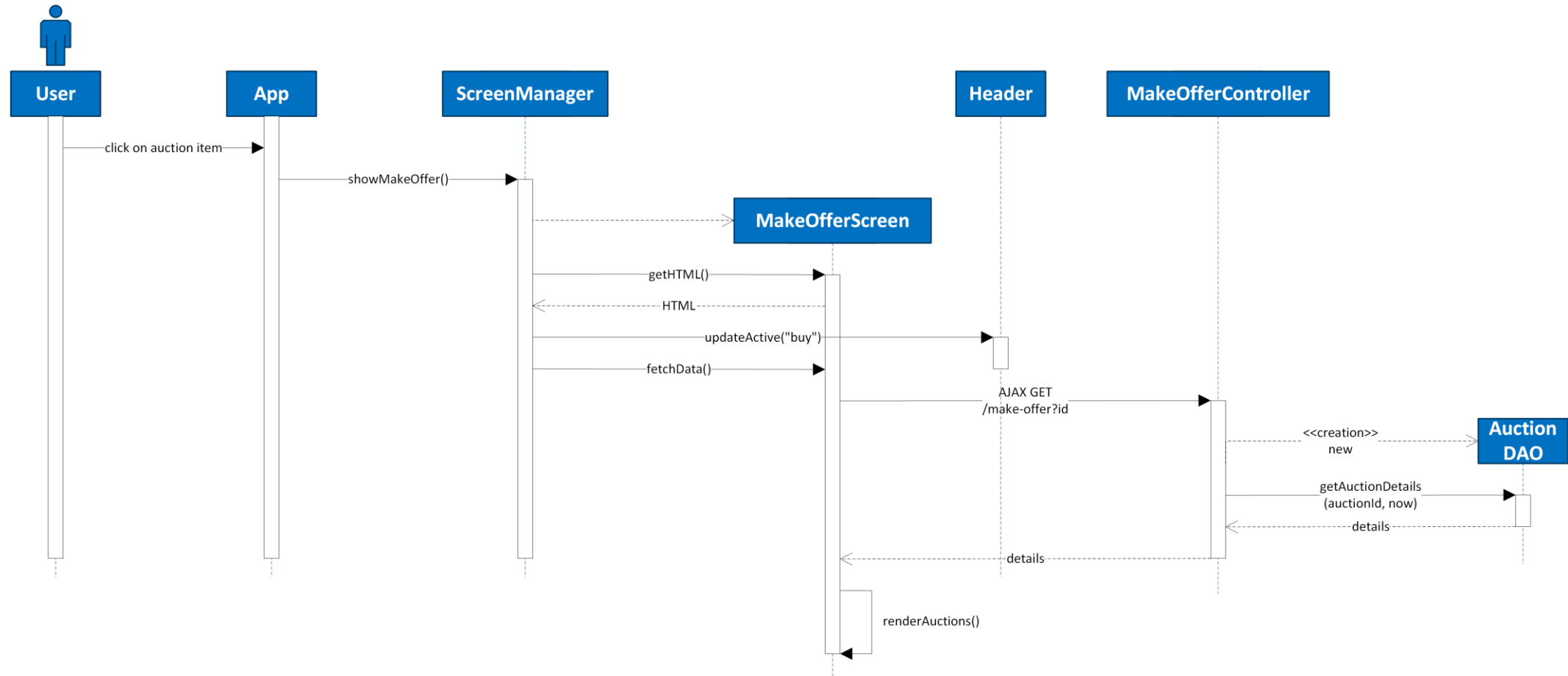




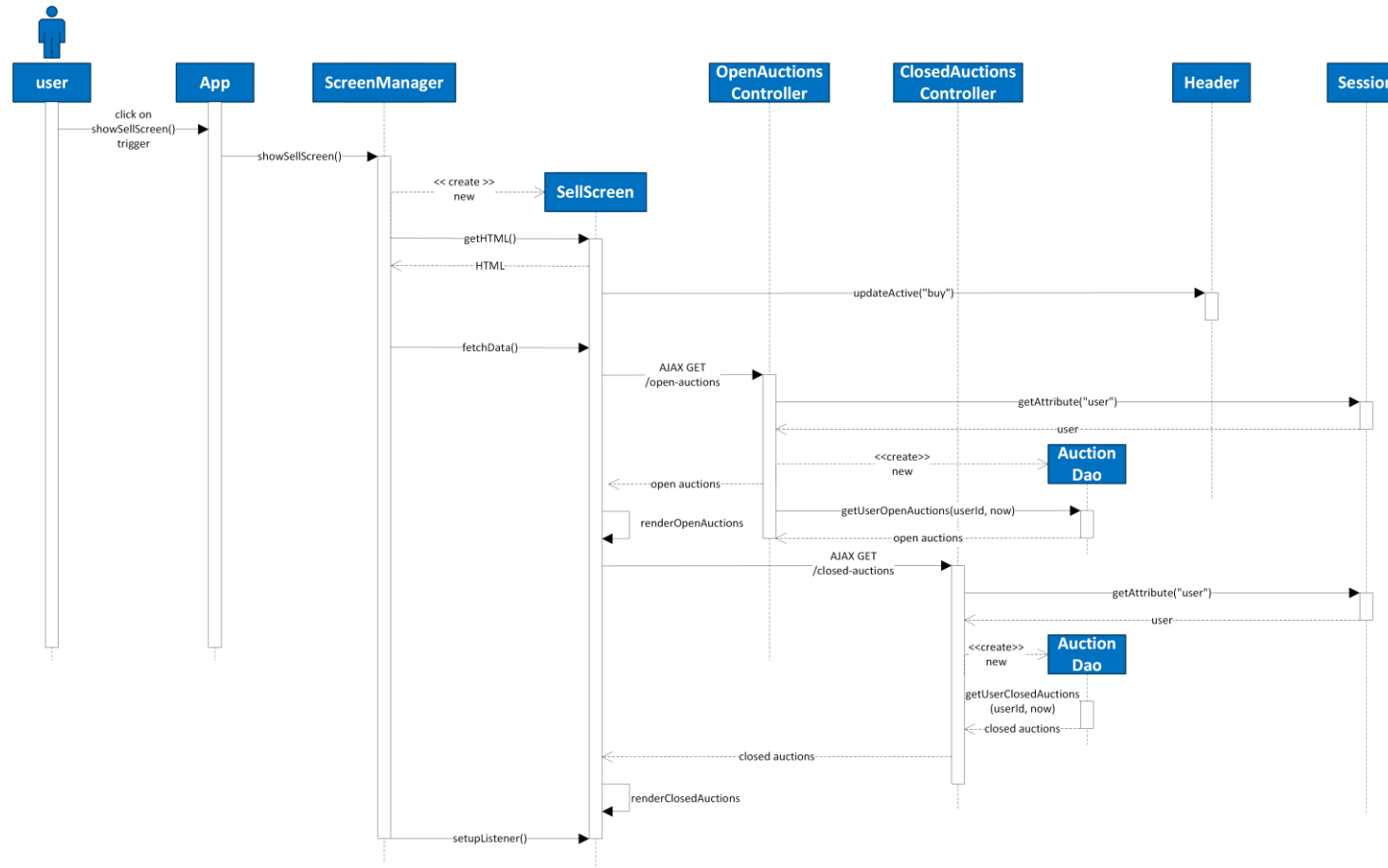
# Sequenza – Buy Filter



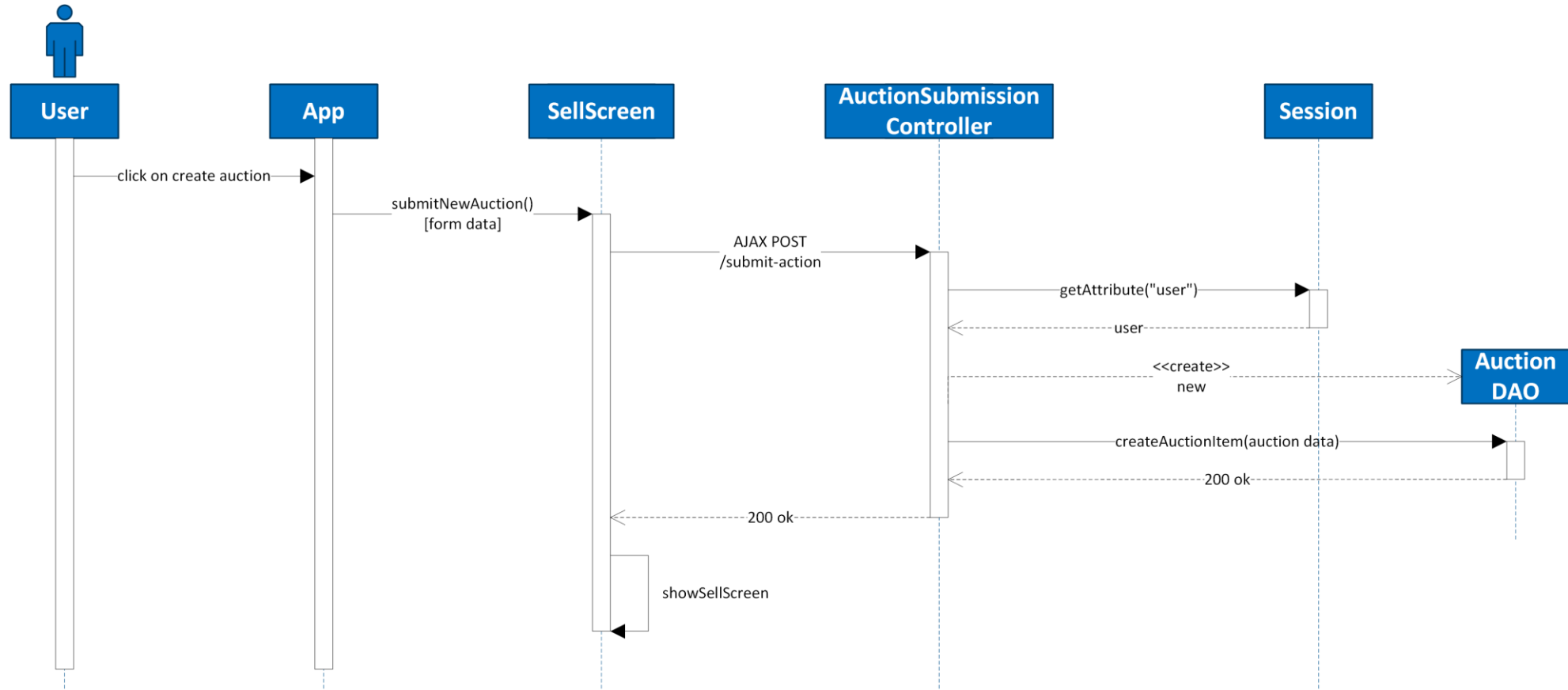
# Sequenza – Bid



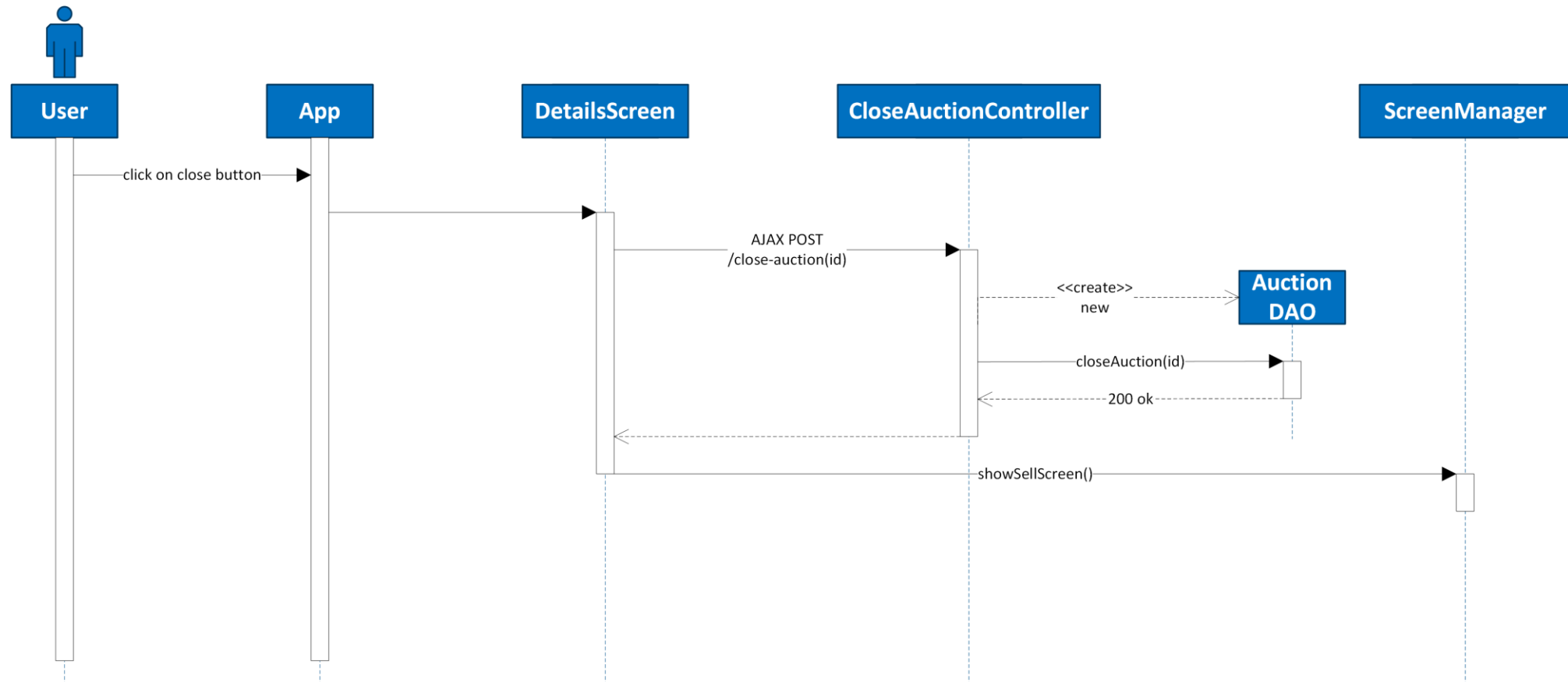
# Sequenza - Sell



# Sequenza – Create auction



# Sequenza – Close auction



# Divisione del lavoro

AuctiOnline

# Divisione del lavoro

- Pizzamiglio Giacomo
  - Sviluppo database
  - Sviluppo DAO
  - Sviluppo pagine VENDO e DETTAGLIO (html)
  - Sviluppo pagine VENDO e DETTAGLIO (JS)
  - Revisione pagine VENDO e DETTAGLIO
  - Redazione relazione e creazione diagrammi
- Prisciantelli Andrea
  - Sviluppo DAO
  - Sviluppo meccanismo di autenticazione
  - Sviluppo pagine LOGIN e SIGNUP
  - Sviluppo pagina HOME (html)
  - Sviluppo funzionalità aggiuntive JS
  - Revisione pagine ACQUISTO, OFFERTA
  - Redazione relazione, creazione delle tabelle e indicazione dei componenti
- Romagnoni Lorenzo
  - Sviluppo ErrorHandler
  - Sviluppo iniziale pagine ACQUISTO e OFFERTA (html)
  - Sviluppo iniziale pagine HOME, ACQUISTO, OFFERTA (JS)
  - Creazione diagrammi
  - Revisione finale parte «HTML pura»