POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Progetto di Tecnologie Informatiche per il Web

Documentazione
TIW

Author: **Carlo arnone**

Student ID: 10797703
Professor: Prof. Piero Fraternali
Academic Year: 2022-23

Contents

1 | Requisiti

1.1. Specifica Generale

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 due form, una per creare un nuovo articolo e una per creare una nuova asta per vendere gli articoli dell'utente. ll primo form inserisce nuovi articoli nel database e il secondo mostra l'elenco degli articoli disponibili nel database e dà la possibilità di selezionarne più di uno. Un articolo ha codice, nome, descrizione, immagine e prezzo. Un'asta comprende uno o più articoli messi in vendita, il prezzo iniziale dell'insieme di articoli, il rialzo minimo di ogni offerta (espresso come un numero intero di euro) e una scadenza (data e ora, es. 19-04-2021 alle 24:00). Il prezzo iniziale dell'asta è ottenuto come somma del prezzo degli articoli compresi nell'offerta. Lo stesso articolo non può essere incluso in aste diverse. Una volta venduto, un articolo non deve essere più disponibile per l'inserimento in ulteriori aste. La lista delle aste è ordinata per data+ora crescente. L'elenco riporta: codice e nome degli articoli compresi nell'asta, offerta massima, tempo mancante (numero di giorni e ore) tra il momento (data ora) 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 tutti 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 dall'utente e non ci si occupi della chiusura automatica di aste dopo la scadenza). Se l'asta è chiusa, la pagina riporta tutti i dati dell'asta, il nome dell'aggiudicatario, il prezzo finale e l'indirizzo (fisso) 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 è aggiornata e mostra un elenco di aste aperte (la cui scadenza è posteriore alla data e ora dell'invio) per cui la parola chiave compare nel nome o nella descrizione di almeno uno degli articoli dell'asta. La lista è ordinata in modo decrescente in base al tempo (numero di giorni e ore) mancante alla chiusura. Cliccando su un'asta aperta compare la pagina OFFERTA che mostra i dati degli articoli, 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 di un importo pari almeno al rialzo minimo. 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 degli articoli e il prezzo finale

1.2. Elaborazione Specifica Generale

- **Login**: al momento del login dovrà essere salvato il timestamp (il tutto viene fatto lato DB).
- **Vincoli temporali**: le scadenze delle nuove aste non può essere antecedente alla data odierna.
- **Vincoli Numerici**: il prezzo proposto di un' Offerta non può essere negativo, ed il primo offerente deve offrire almeno il prezzo previsto dalla base d'asta più il minimo rialzo, come devono essere positivi e maggiori di zero i prezzo di un articolo e il rialzo minimo all'interno di un'asta.
- **Vincoli Aste**: Un utente che ha creato un'asta non può sottomettere offerte alla stessa, un'asta deve avere almeno un articolo.

1.3. Estensione JSP

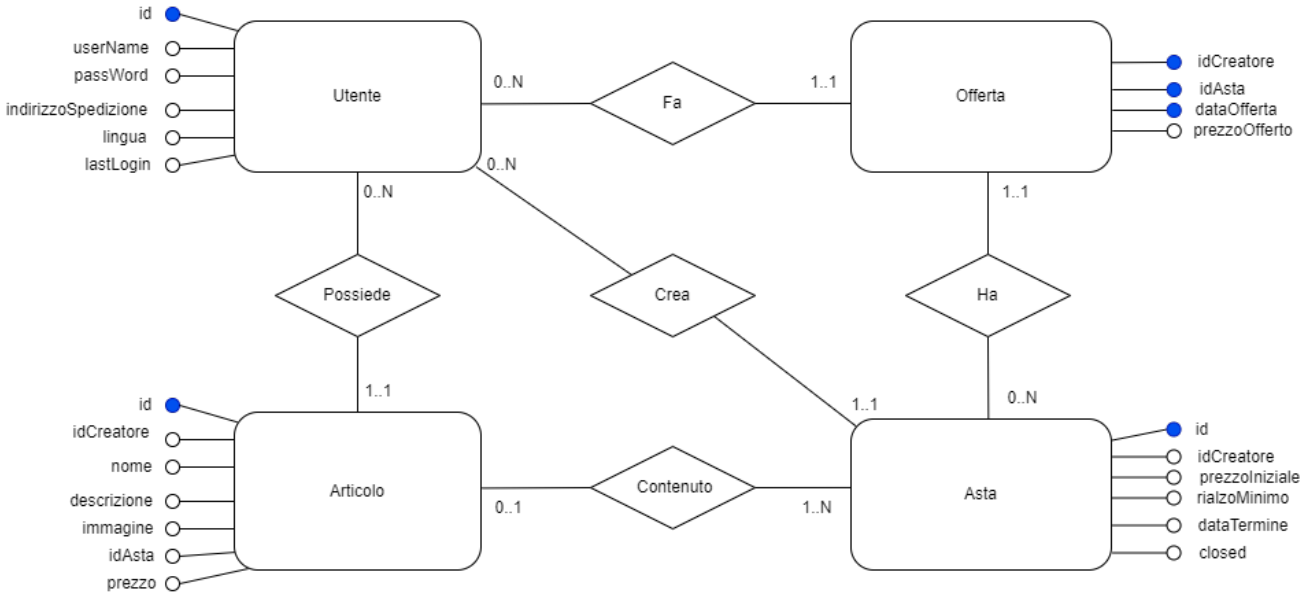
Dopo il login, l'intera applicazione è realizzata con un'unica pagina.
Se l'utente accede per la prima volta l'applicazione mostra il contenuto della pagina ACQUISTO. Se l'utente ha già usato l'applicazione, questa mostra il contenuto della pagina VENDO se l'ultima azione dell'utente è stata la creazione 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.

1.4. Elaborazione Specifiche JSP

- **Interazioni Valide**: Tutte le interazioni dell'utente tranne l'apertura dei dettagli di un'asta sono interazioni valide che sovrascrivono l'ultima azione compiuta dall'utente.
- **Salvataggio info Lato Client**: Le informazioni vengono salvate nei cookies, con formato "cookie-name"_"id-user".

2 | Progettazione base di dati

2.1. Schema ER



2.2. DDL

2.2.1. Aste

```
CREATE TABLE `aste` (
  `id` int NOT NULL AUTO_INCREMENT,
  `idCreatore` int NOT NULL,
  `prezzoIniziale` int NOT NULL,
  `rialzoMinimo` int NOT NULL,
  `dataTermine` datetime NOT NULL,
  `closed` bit(1) NOT NULL,
  PRIMARY KEY (`id`)
);
```

2.2.2. AsteAperte

```
CREATE VIEW `asteaperte` AS (SELECT
 1 AS `id`,
 1 AS `idCreatore`,
 1 AS `prezzoMassimoRaggiunto`,
 1 AS `rialzoMinimo`,
 1 AS `dataTermine`,
 1 AS `oreRimanenti`);
```

2.2.3. AsteChiuse

```
CREATE VIEW `astechiuse` AS (SELECT
 1 AS `id`,
 1 AS `idCreatore`,
 1 AS `idVincitore`,
 1 AS `prezzoFinale`,
 1 AS `dataTermine`,
 1 AS `indirizzoSpedizione`);
```

2.2.4. Articoli

```
CREATE TABLE `articoli` (
  `id` int NOT NULL AUTO_INCREMENT,
  `idCreatore` int NOT NULL,
  `nome` varchar(50) NOT NULL,
  `descrizione` varchar(255) DEFAULT NULL,
  `immagine` varchar(255) NOT NULL,
  `idAsta` int DEFAULT NULL,
  `prezzo` int NOT NULL,
  PRIMARY KEY (`id`)
)
```

2.2.5. Utenti

```
CREATE TABLE `utenti` (
  `id` int NOT NULL AUTO_INCREMENT,
  `userName` varchar(50) NOT NULL,
  `passWord` varchar(100) NOT NULL,
  `indirizzoSpedizione` varchar(255) DEFAULT NULL,
  `lingua` varchar(15) DEFAULT 'it_IT',
  `lastLogin` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`)
)
```

2.2.6. Offerte

```
CREATE TABLE `offerte` (
  `idCreatore` int NOT NULL,
  `idAsta` int NOT NULL,
  `prezzoOfferto` int NOT NULL,
  `dataOfferta` datetime NOT NULL,
  PRIMARY KEY (`idCreatore`,`idAsta`,`dataOfferta`)
)
```

2.3. BenchMark Create View Query

Ho deciso di progettare il DB in modo tale che potessi fare distinzione tra asta chiuse ed aste aperte tramite delle view, avendo quindi avuto molteplici idee su come strutturare la query SQL per ottenere le relative view, ho eseguito un benchmark su le seguenti due query.

Statement 1:

```
CREATE VIEW asteAperte(idAsta, prezzoMassimoRaggiunto, rialzoMinimo,
    dataScadenza, oreRimanenti) AS (
    SELECT A.id, 0, A.rialzoMinimo, A.dataTermine, TIMESTAMPDIFF(HOUR,
        A.dataTermine, U.lastLogin )
    FROM aste A join utenti U on idCreatore = U.id
    WHERE closed = false AND A.id NOT IN (SELECT distinct idAsta
                                          FROM offerte)

    UNION

    SELECT A.id, O.prezzoOfferto, A.rialzoMinimo, A.dataTermine,
        TIMESTAMPDIFF(HOUR, A.dataTermine, U.lastLogin )
    FROM aste A join offerte O on A.id = O.idAsta
              left join utenti U on U.id = O.idCreatore
    WHERE closed = false AND O.prezzoOfferto = (SELECT MAX(prezzoOfferto)
                                          FROM offerte O1
                                          WHERE A.id = O1.idAsta)
);
```

Statement 2:

```
CREATE VIEW asteAperte(idAsta, prezzoMassimoRaggiunto, rialzoMinimo,
    dataScadenza, oreRimanenti) AS (

    SELECT A.id, O.prezzoOfferto, A.rialzoMinimo, A.dataTermine,
        TIMESTAMPDIFF(HOUR , U.lastLogin, A.dataTermine)
    FROM aste A left join offerte O on A.id = O.idAsta
              left join utenti U on U.id = O.idCreatore
    WHERE A.closed = false AND (O.prezzoOfferto = (SELECT
        MAX(prezzoOfferto)
                                          FROM offerte O1
                                          WHERE A.id = O1.idAsta)
                               OR O.prezzoOfferto IS NULL
                           )
);
```

	text
1	Run 0, Statement 1 : 81217
2	Run 0, Statement 2 : 165406
3	Run 1, Statement 1 : 143026
4	Run 1, Statement 2 : 171447
5	Run 2, Statement 1 : 148718
6	Run 2, Statement 2 : 185665
7	Run 3, Statement 1 : 135717
8	Run 3, Statement 2 : 176425
9	Run 4, Statement 1 : 153406
10	Run 4, Statement 2 : 184924

Figure 2.1: **Composizione BenchMark**: 5 cluster di 10000 esecuzioni, su un DB popolato con 300000 Offerte, 100000 aste e 2000 utenti.

3 | Caratteristiche Comuni

3.1. Progettazione e Design

3.2. Rappresentazione IFML

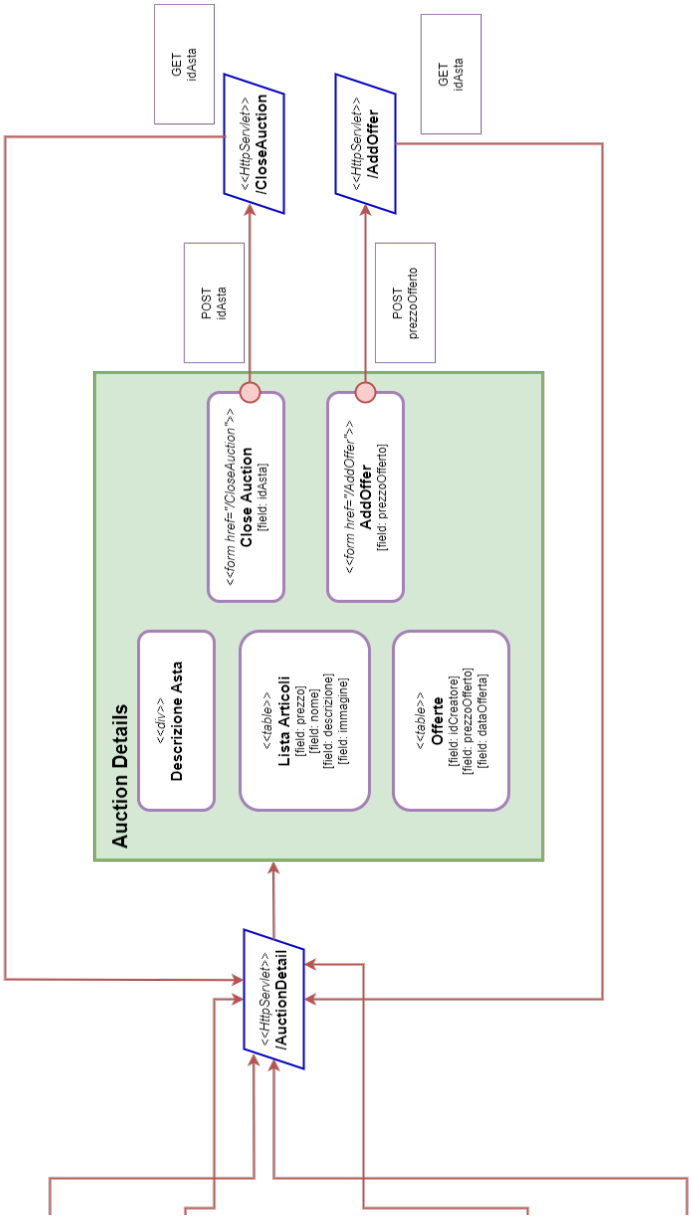
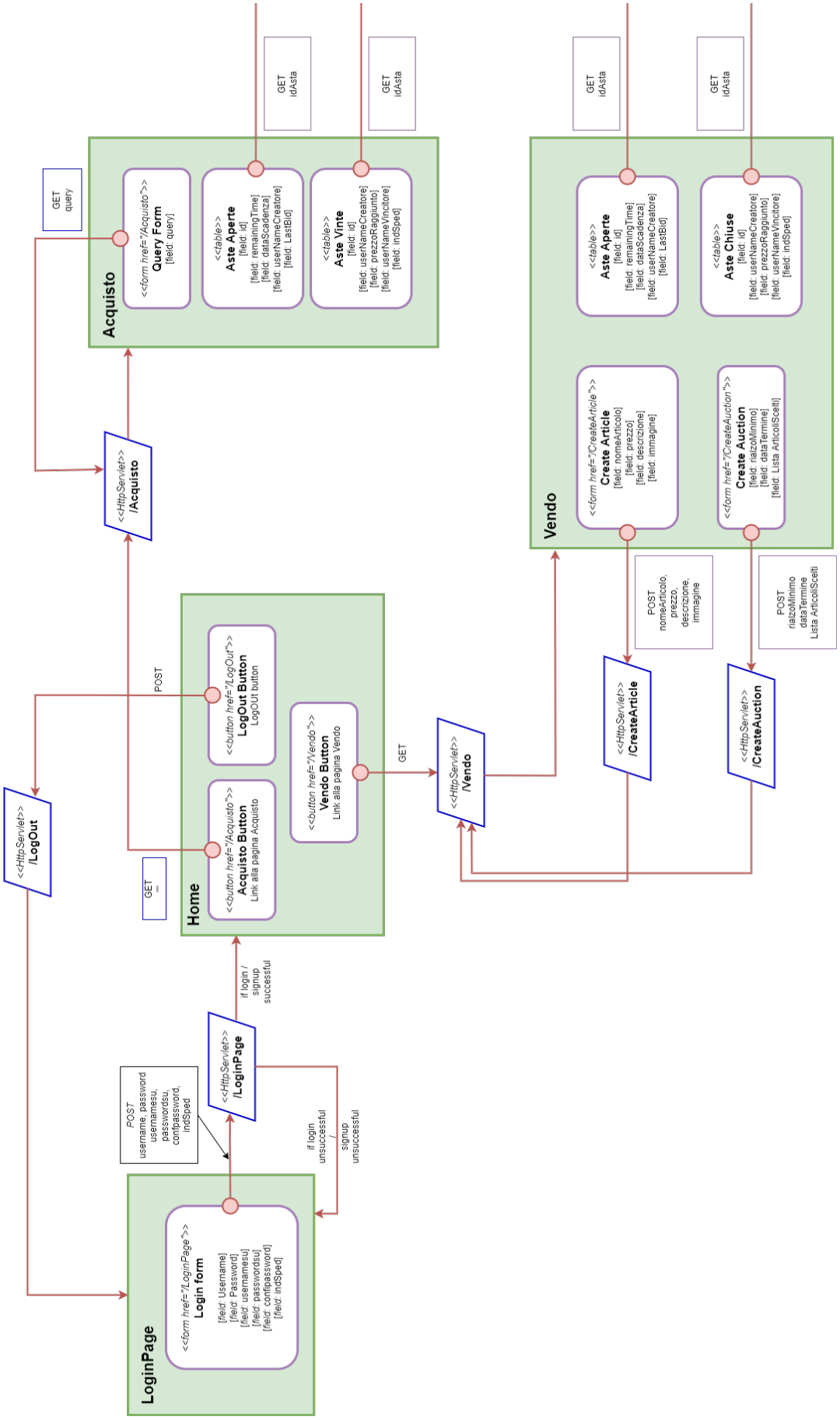


Figure 3.1: Image (Page 2)


LoginPage
<<form href="/LoginPage">>
Login form
[field: Username]
[field: Password]
[field: usernamesu]
[field: passwordsu]
[field: confpassword]
[field: indSped]
POST
username, password
usernamesu,
passwordsu,
confpassword,
indSped
<<HttpServlet>>
/LoginPage
if login unsuccessful / signup unsuccessful
if login / signup successful
Home
<<button href="/Acquisto">>
Acquisto Button
Link alla pagina Acquisto
<<button href="/LogOut">>
LogOut Button
LogOut button
<<button href="/Vendo">>
Vendo Button
Link alla pagina Vendo
GET
...
POST
<<HttpServlet>>
/LogOut
<<HttpServlet>>
/Acquisto
GET
<<HttpServlet>>
/Vendo
Acquisto
GET
query
<<form href="/Acquisto">>
Query Form
[field: query]
<<table>>
Aste Aperte
[field: id]
[field: remainingTime]
[field: dataScadenza]
[field: userNameCreatore]
[field: LastBid]
<<table>>
Aste Vinte
[field: id]
[field: userNameCreatore]
[field: prezzoRaggiunto]
[field: userNameVincitore]
[field: indSped]
GET
idAsta
GET
idAsta
Vendo
<<form href="/CreateArticle">>
Create Article
[field: nomeArticolo]
[field: prezzo]
[field: descrizione]
[field: immagine]
<<form href="/CreateAuction">>
Create Auction
[field: rialzoMinimo]
[field: dataTermine]
[field: Lista ArticoliScelti]
<<table>>
Aste Aperte
[field: id]
[field: remainingTime]
[field: dataScadenza]
[field: userNameCreatore]
[field: LastBid]
<<table>>
Aste Chiuse
[field: id]
[field: userNameCreatore]
[field: prezzoRaggiunto]
[field: userNameVincitore]
[field: indSped]
GET
idAsta
GET
idAsta
POST
nomeArticolo,
prezzo,
descrizione,
immagine
POST
rialzoMinimo
dataTermine
Lista ArticoliScelti
<<HttpServlet>>
/CreateArticle
<<HttpServlet>>
/CreateAuction

IFML comune, le uniche differenze sono come le pagine vengono mostrate nel caso HTML con thymeleaf e il templateEngine, nell'altro tramite la funzione showPageX() conenuta in utils.js

3.2.1. SequenceDiagram

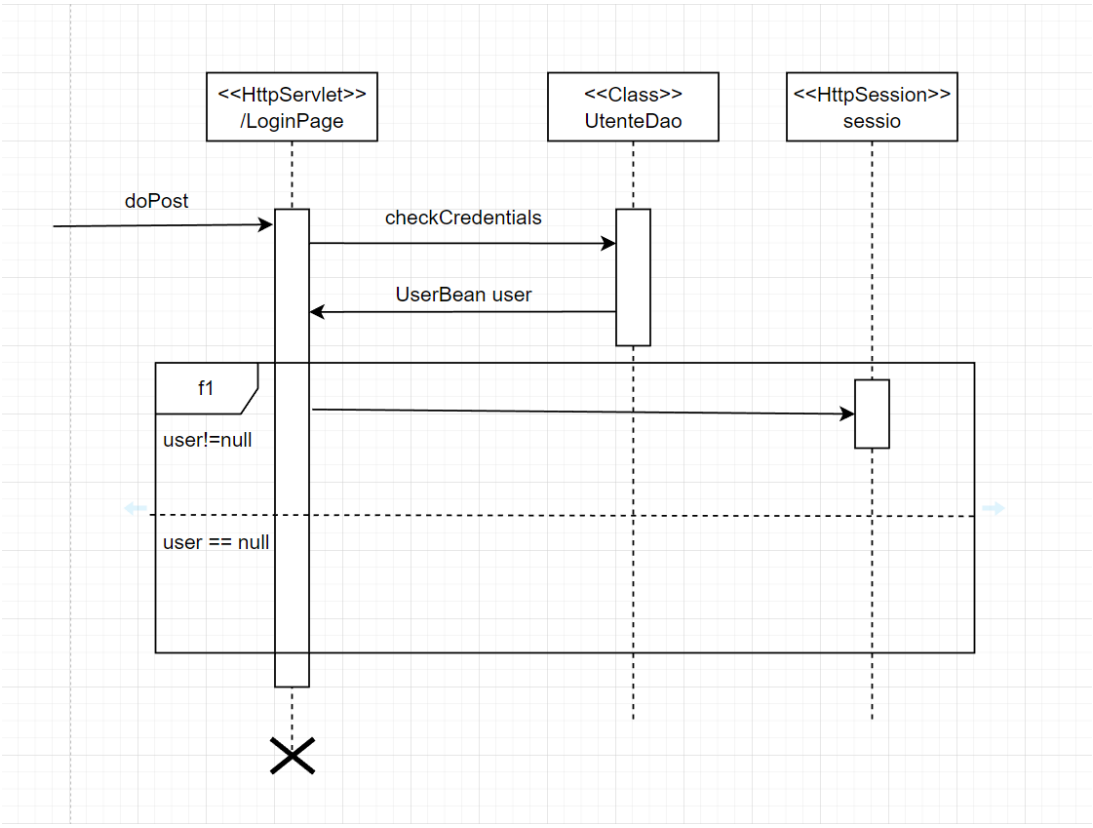


Figure 3.2: Login Interaction

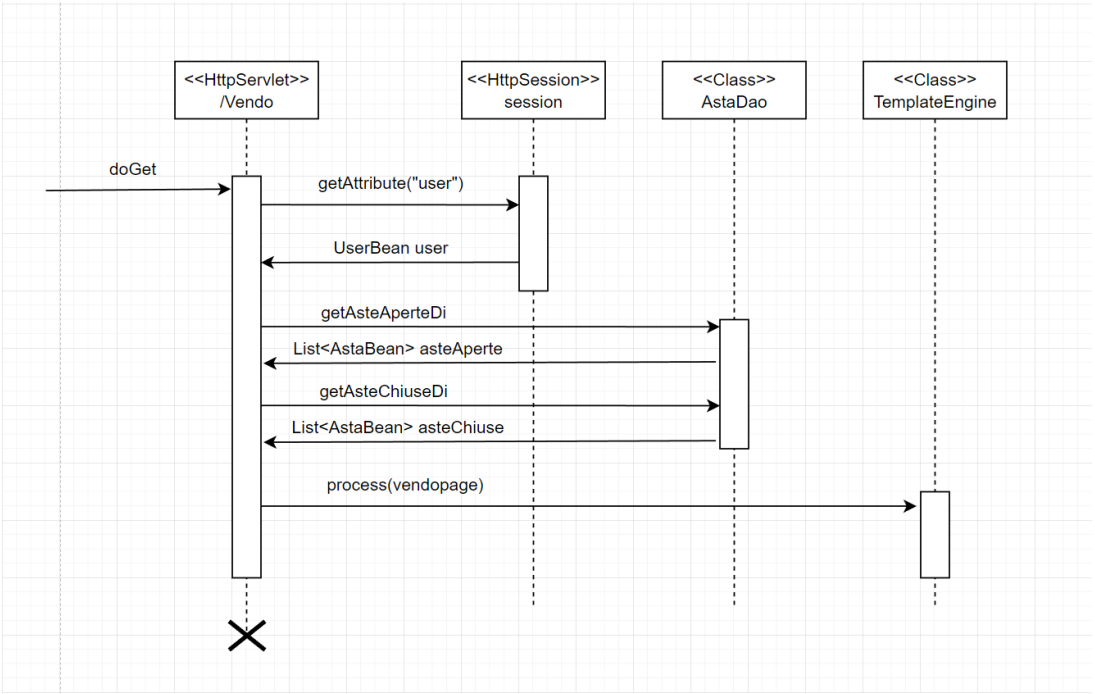


Figure 3.3: Vendo/Acquisto page get

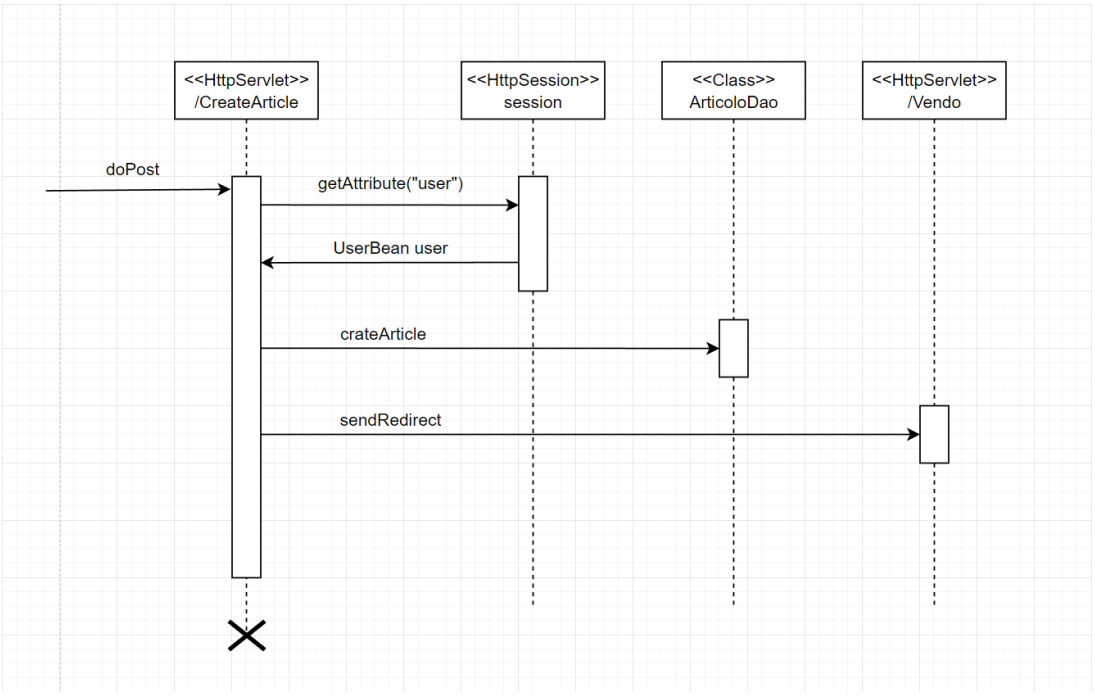


Figure 3.4: createArticle process

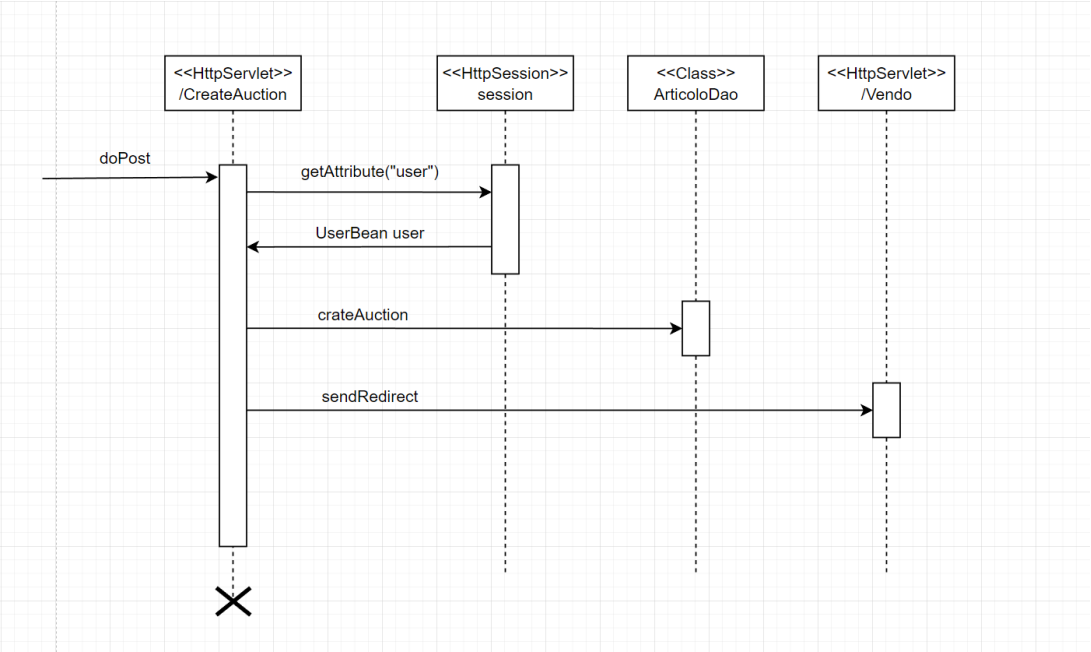


Figure 3.5: createAuction process

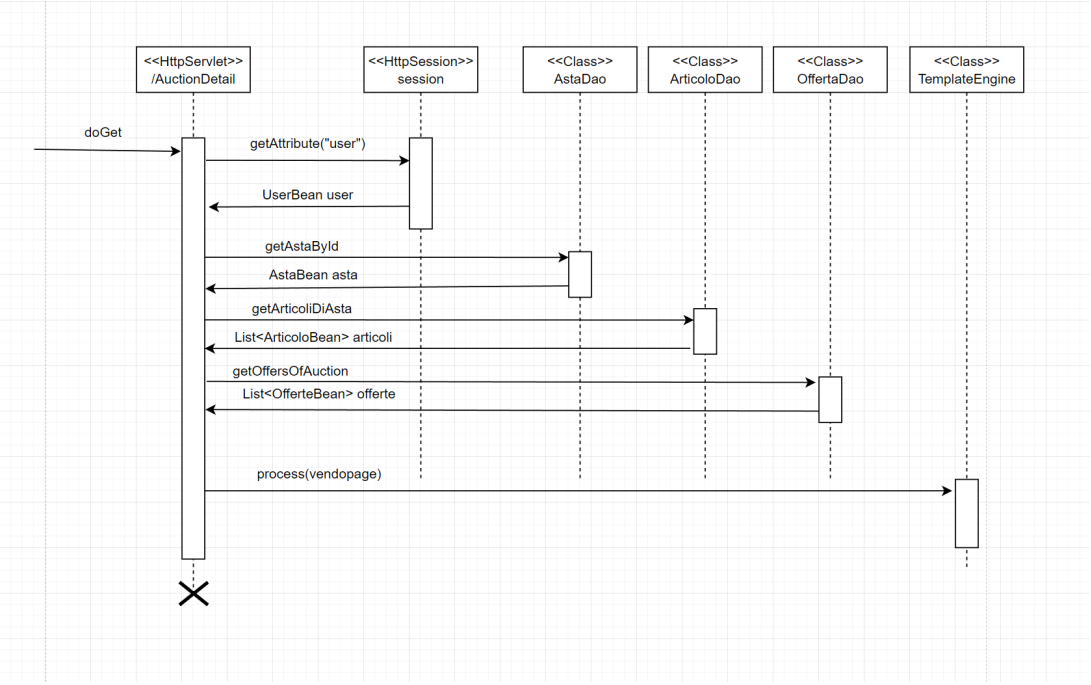


Figure 3.6: auctionDetail get

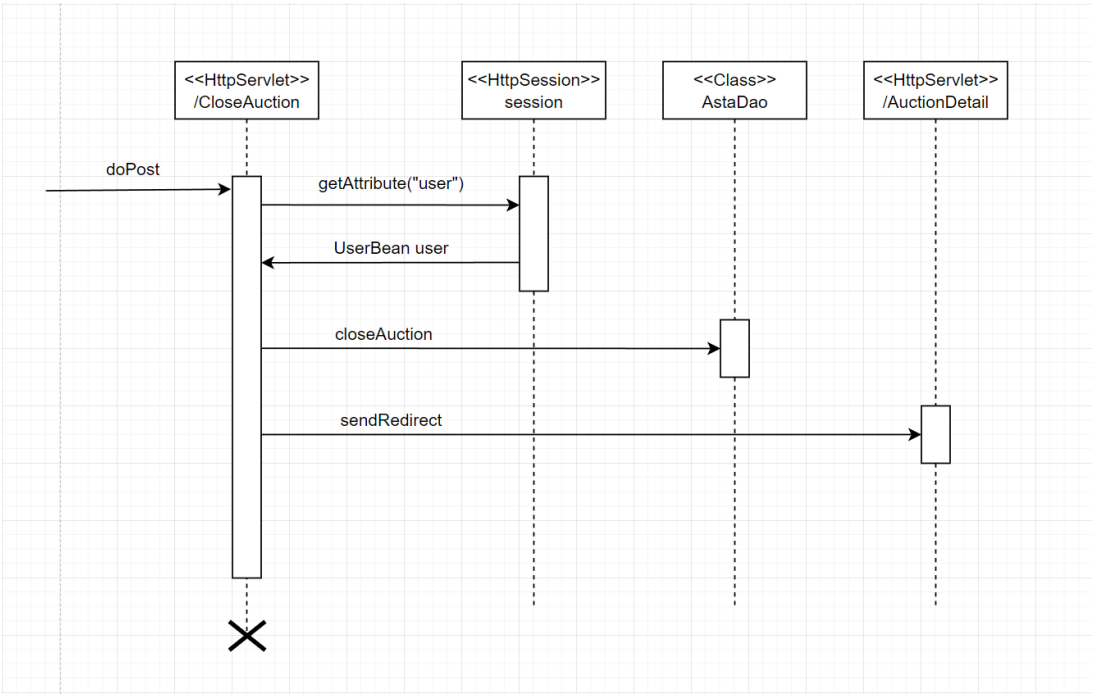


Figure 3.7: closeAuction process

3.2.2. Filtri

Per ogni pagine e ogni servlet vi è un check se lo user sia loggato, per garantire che non si possa accedere ad alcune parti della webApp senza possedere un'identità definita.

3.3. Struttura Codice

3.3.1. Beans

- **UserBean**
- **AstaBean**: la distinzione tra astaAperta e astaChiusa avviene nel costruttore nel Bean
- **OffertaBean**
- **ArticoloBean**

3.3.2. DAO

Articolo

- **inserisciArticoloInAsta**(int idAsta, int idArticolo)
- **getArticoliDiAsta**(int idAsta)
- **getArticoliVendibiliDi**(int userId)

- **aggiungiArticolo**(int idCreatore, String articleName, int price, String description, Part image)

Asta

- **getAsteAperteDi**(int userId)
- **getAsteAperte**()
- **getAsteChiuseDi**(int userId)
- **getAsteChiuse**()
- **getAstaById**(int idAsta)
- **inserisciAsta**(List<Integer> idArticoli, Date dataTermine, int userId, int rialzoMinimo)
- **getAsteAperteContenenti**(String query)
- **getAsteVinteDa**(int userId)
- **closeAuction**(int id)

Offerte

- **getOffersOfAuction**(int idAsta)
- **insertOffer**(int idCreatore, int idAsta, int prezzoOfferto)

Utente

- **checkCredentials**(String username, String password)
- **isUserPresent**(String username)
- **insertUser**(String username, String password, String indSped)
- **getUsernameById**(int id)

3.3.3. Controllers

- **Acquisto**
- **AddOffer**
- **CloseAuction**
- **CreateArticle**
- **DettaglioAsta**
- **Home**
- **LogOut**

- **Vendo**

3.3.4. Filtri

- **LoginCheck**

3.3.5. Exceptions

- **AuctionNotInserted**
- **NotImageException**
- **WrongDateException**

4 | Versione HTML Pura

4.1. Struttura Codice

4.1.1. Template THL

- **acquistopage**
- **auctiondetail**
- **home**
- **vendoPage**

5 | Versione RIA

5.1. Rappresentazione IFML

5.2. Struttura Codice

utils.js

- **showPageX**(pageName, param): indicando il nome della pagina si occupa di nascondere tutte le altre e chiama il la funzione getData relativa alla pagina da mostrare.
- **makeCall**: usata solo per comunicare contenuti dei form tramite POST alle servlet.
- **makeCallParams**: viene passata una stringa che rappresenta la request e di seguito incollata al url della servlet da chiamare.
- **Varie createTableRows**: generare il codice HTML per riempire le tabelle.
- **generateAcutionDescription**: genera codice HTML per il campo di descrizione dell'asta.
- **generateReqParametersFromJson**: prende un json e lo trasforma in una string rappresentante una request di tipo GET.
- **find and setCookie**: per leggere e scrivere i cookies.
- **addFlagForLastSeen**: viene invocata quando viene cliccato il bottone open details di un'asta.

home.js

- **BindObjects**: chiamata a loadTime per collegare tutti i bottoni statici ai loro relativi funzionamenti
- **PageUtils**: contengono la funzione getData che può prendere parametri, in tal caso esegue una chiamata a makeCallParams, con i parametri passati, viceversa invece effetua una singola chiamata a makeCall per ricevere dalle servlet i dati della pagina aggiornati

loginInteraction.js

- **LoginForm**: aggiunge il listener al bottone di submit e esegue una makeCall alla servlet LoginPage.

Pagine HTML

- **index.html**
- **Home.html**