



POLITECNICO

MILANO 1863

Progetto di Tecnologie Informatiche per il Web
A.A. 2024-2025

D. A. Onorati
C.P.: 10797474
GroupID: 118
Traccia: 1
Prof: Piero Fraternali

27 maggio 2025

Indice

1 Specifica: Esercizio 1: aste online	3
1.1 Versione HTML pura	3
1.2 Versione con JavaScript	3
2 DataBase	4
2.1 Design	4
2.2 Schema	4
3 Applicazione	6
3.1 Requisiti Generali e Struttura Dati (Comuni a Entrambe le Versioni)	6
3.2 Versione HTML Pura	6
3.3 Versione con JavaScript (RIA)	10
3.4 Design (IFML)	11
3.4.1 Versione HTML pura	11
3.4.2 Versione con JavaScript (RIA)	12
3.5 Components List	13
3.6 Sequence Diagrams	14
3.6.1 Versione HTML pura	14
3.6.2 Versione con JavaScript (RIA)	23

1 Specifica: Esercizio 1: aste online

1.1 Versione HTML pura

Un'applicazione web consente la gestione di aste online. Gli utenti accedono tramite login e possono vendere e acquistare articoli all'asta. Ogni utente ha username, password, nome, cognome e indirizzo (quest'ultimo è usato per la spedizione degli articoli comperati). Ogni articolo ha codice, nome, descrizione, immagine e prezzo. 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, uno per creare un nuovo articolo e uno per creare una nuova asta per vendere gli articoli dell'utente. Il primo form consente di inserire nel database un articolo con tutti i suoi dati, che sono obbligatori. Il secondo form mostra l'elenco degli articoli presenti nel database e disponibili per la vendita e dà la possibilità di selezionarne più di uno per creare un'asta che li comprende. Un'asta è formata da 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 nella pagina VENDO 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 nell'elenco 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 Versione con 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.

2 DataBase

2.1 Design

- **User:** rappresenta gli utenti dell'applicazione. Memorizza credenziali, informazioni personali e l'indirizzo di spedizione.
- **Item:** rappresenta gli articoli messi in vendita. Ogni articolo è creato da un utente (venditore) e contiene dettagli come codice, nome, descrizione, immagine e prezzo base.
- **Auction:** rappresenta un'asta. Un'asta è creata da un utente, comprende uno o più articoli, ha un prezzo iniziale (somma dei prezzi base degli articoli), un rialzo minimo, una scadenza e uno stato (aperta/chiusa). In caso di chiusura con offerte, memorizza l'utente vincitore e il prezzo finale.
- **Auction_Item_Link:** tabella di collegamento N:M tra aste e articoli, poiché un'asta può contenere più articoli e (teoricamente, anche se la specifica dice "Lo stesso articolo non può essere incluso in aste diverse" se aperte) un articolo potrebbe far parte di più aste se non fosse per tale vincolo. Il vincolo "stesso articolo non può essere incluso in aste diverse (aperte)" e "una volta venduto, un articolo non deve essere più disponibile" è gestito a livello applicativo e parzialmente dalla logica di query per gli item disponibili.
- **Bid:** rappresenta un'offerta fatta da un utente per un'asta. Memorizza l'importo dell'offerta e il timestamp.

Le relazioni principali sono:

- Un Utente può creare molti Articoli (1:N).
- Un Utente può creare molte Aste (1:N).
- Un'Asta è composta da uno o più Articoli (1:N, implementata con tabella di link).
- Un Utente può fare molte Offerte (1:N).
- Un'Asta può avere molte Offerte (1:N).
- Un'Asta (chiusa) può avere un Utente vincitore (1:1, opzionale).

2.2 Schema

Di seguito è riportato lo schema SQL del database `online_auctions_db`.

```
1  -- Schema online_auctions_db
2
3  DROP SCHEMA IF EXISTS 'online_auctions_db';
4  CREATE SCHEMA 'online_auctions_db' DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci ;
5  USE 'online_auctions_db' ;
6
7
8  -- Table 'user'
9
10 CREATE TABLE IF NOT EXISTS 'user' (
11     'id' INT NOT NULL AUTO_INCREMENT,
12     'username' VARCHAR(45) NOT NULL,
13     'password' VARCHAR(255) NOT NULL,
14     'name' VARCHAR(45) NOT NULL,
15     'surname' VARCHAR(45) NOT NULL,
16     'shippingAddress' VARCHAR(255) NOT NULL,
17     PRIMARY KEY ('id'),
18     UNIQUE INDEX 'username_UNIQUE' ('username' ASC) VISIBLE)
19 ENGINE = InnoDB;
20
21
22  -- Table 'item'
23
24 CREATE TABLE IF NOT EXISTS 'item' (
25     'id' INT NOT NULL AUTO_INCREMENT,
26     'itemCode' VARCHAR(50) NOT NULL,
27     'name' VARCHAR(100) NOT NULL,
28     'description' TEXT NOT NULL,
29     'image' LONGBLOB NOT NULL,
30     'basePrice' DECIMAL(10,2) NOT NULL,
31     'sellerUserId' INT NOT NULL,
32     PRIMARY KEY ('id'),
```

```

33     UNIQUE INDEX `itemCode_UNIQUE` (`itemCode` ASC) VISIBLE,
34     INDEX `fk_item_user_idx` (`sellerUserId` ASC) VISIBLE,
35     CONSTRAINT `fk_item_user`
36       FOREIGN KEY (`sellerUserId`)
37         REFERENCES `user` (`id`)
38         ON DELETE CASCADE
39         ON UPDATE NO ACTION)
40 ENGINE = InnoDB;
41
42
43 -- Table 'auction'
44
45 CREATE TABLE IF NOT EXISTS `auction` (
46   `id` INT NOT NULL AUTO_INCREMENT,
47   `initialPrice` DECIMAL(10,2) NOT NULL,
48   `minimumBidIncrement` DECIMAL(10,2) NOT NULL, -- Coerente con la specifica "numero intero di euro",
49   ↪ ma DECIMAL per flessibilità
50   `deadline` TIMESTAMP NOT NULL,
51   `creationTimestamp` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
52   `creatorUserId` INT NOT NULL,
53   `status` VARCHAR(10) NOT NULL COMMENT 'Possible values: OPEN, CLOSED',
54   `winnerUserId` INT NULL DEFAULT NULL,
55   `winningPrice` DECIMAL(10,2) NULL DEFAULT NULL,
56   PRIMARY KEY (`id`),
57   INDEX `fk_auction_creator_user_idx` (`creatorUserId` ASC) VISIBLE,
58   INDEX `fk_auction_winner_user_idx` (`winnerUserId` ASC) VISIBLE,
59   CONSTRAINT `fk_auction_creator_user`
60     FOREIGN KEY (`creatorUserId`)
61       REFERENCES `user` (`id`)
62       ON DELETE CASCADE
63       ON UPDATE NO ACTION,
64   CONSTRAINT `fk_auction_winner_user`
65     FOREIGN KEY (`winnerUserId`)
66       REFERENCES `user` (`id`)
67       ON DELETE SET NULL
68       ON UPDATE NO ACTION)
69 ENGINE = InnoDB;
70
71
72 -- Table 'auction_item_link'
73
74 CREATE TABLE IF NOT EXISTS `auction_item_link` (
75   `auction_id` INT NOT NULL,
76   `item_id` INT NOT NULL,
77   PRIMARY KEY (`auction_id`, `item_id`),
78   INDEX `fk_link_item_idx` (`item_id` ASC) VISIBLE,
79   INDEX `fk_link_auction_idx` (`auction_id` ASC) VISIBLE,
80   CONSTRAINT `fk_link_auction`
81     FOREIGN KEY (`auction_id`)
82       REFERENCES `auction` (`id`)
83       ON DELETE CASCADE
84       ON UPDATE NO ACTION,
85   CONSTRAINT `fk_link_item`
86     FOREIGN KEY (`item_id`)
87       REFERENCES `item` (`id`)
88       ON DELETE CASCADE
89       ON UPDATE NO ACTION)
90 ENGINE = InnoDB;
91
92 -- Table 'bid'
93
94 CREATE TABLE IF NOT EXISTS `bid` (
95   `id` INT NOT NULL AUTO_INCREMENT,
96   `auctionId` INT NOT NULL,
97   `userId` INT NOT NULL,
98   `amount` DECIMAL(10,2) NOT NULL,
99   `timestamp` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
100  PRIMARY KEY (`id`),
101  INDEX `fk_bid_auction_idx` (`auctionId` ASC) VISIBLE,
102  INDEX `fk_bid_user_idx` (`userId` ASC) VISIBLE,
103  CONSTRAINT `fk_bid_auction`
104    FOREIGN KEY (`auctionId`)
105      REFERENCES `auction` (`id`)
106      ON DELETE CASCADE
107      ON UPDATE NO ACTION,
108  CONSTRAINT `fk_bid_user`
109    FOREIGN KEY (`userId`)
110      REFERENCES `user` (`id`)
111      ON DELETE CASCADE
112      ON UPDATE NO ACTION)
113 ENGINE = InnoDB;

```

Listing 1: Schema del database online_auctions_db

3 Applicazione

3.1 Requisiti Generali e Struttura Dati (Comuni a Entrambe le Versioni)

1. **Requisito:** Gli utenti accedono tramite login e possono vendere e acquistare articoli all'asta.

- **Implementazione (HTML & RIA):**

- *Login:* Gestito da `LoginServlet.java` e la pagina di login (`login.html`). Le credenziali sono verificate contro il database tramite `UserDAO.java`. Un `AuthenticationFilter.java` protegge le risorse che richiedono autenticazione.
- *Sessione Utente:* Dopo il login, le informazioni dell'utente (bean `User`) e il timestamp del login sono memorizzati nella `HttpSession` (lato server) e in `sessionStorage` (lato client per la RIA).
- *Vendere/Acquistare:*
 - * Pagina VENDO: `SellPageServlet.java` e `sell.html` (HTML) / `sellPage.js` (RIA).
 - * Pagina ACQUISTO: `BuyPageServlet.java` e `buy.html` (HTML) / `buyPage.js` (RIA).

2. **Requisito:** Ogni utente ha username, password, nome, cognome e indirizzo (quest'ultimo è usato per la spedizione degli articoli comperati).

- **Implementazione (HTML & RIA):**

- *Bean:* La classe record `User.java` modella l'entità utente con tutti gli attributi richiesti.
- *Database:* La tabella `user` nel file `online_auctions.db.sql` rispecchia questa struttura.
- *DAO:* La classe `UserDAO.java` gestisce le operazioni di accesso ai dati utente.
- *Utilizzo Indirizzo:* L'attributo `shippingAddress` dell'utente vincitore viene visualizzato nella pagina di dettaglio dell'asta (`AuctionDetailServlet`, `auctionDetail.html` / `auctionDetail.js`) quando un'asta è chiusa.

3. **Requisito:** Ogni articolo ha codice, nome, descrizione, immagine e prezzo.

- **Implementazione (HTML & RIA):**

- *Bean:* La classe record `Item.java` modella l'entità articolo. Il "prezzo" è implementato come `basePrice`. L'immagine è gestita come array di byte (`byte[]`) e convertita in formato Base64 per la visualizzazione nel client.
- *Database:* La tabella `item` contiene questi attributi.
- *DAO:* La classe `ItemDAO.java` gestisce le operazioni sugli articoli.

3.2 Versione HTML Pura

4. **Requisito (HOME Page):** La HOME page contiene due link, uno per accedere alla pagina VENDO e uno per accedere alla pagina ACQUISTO.

- **Implementazione:** La servlet `HomeServlet.java` serve la pagina `home.html` (generata da Thymeleaf), che presenta i link "Sell Your Items" e "Browse and Buy Items", i quali puntano rispettivamente a `SellPageServlet` e `BuyPageServlet`.

5. **Requisito (Pagina VENDO - Struttura):** 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, uno per creare un nuovo articolo e uno per creare una nuova asta per vendere gli articoli dell'utente.

- **Implementazione:**

- `SellPageServlet.java` utilizza `AuctionDAO.findAuctionsByCreatorAndStatus()` per recuperare le aste aperte e chiuse dell'utente e `ItemDAO.findAvailableItemsBySellerId()` per gli articoli disponibili.
- Il template `sell.html` renderizza queste informazioni:
 - * Una tabella per le aste aperte ("Your Open Auctions").
 - * Una tabella per le aste chiuse ("Your Closed Auctions").
 - * Un form per la creazione di un nuovo articolo (target: `CreateItemServlet`).

* Un form per la creazione di una nuova asta (target: `CreateAuctionServlet`).

6. **Requisito (Pagina VENDO - Form Creazione Articolo):** Il primo form consente di inserire nel database un articolo con tutti i suoi dati, che sono obbligatori.

• **Implementazione:**

- Il form in `sell.html` include input per codice, nome, descrizione, immagine (upload file) e prezzo base, tutti con l'attributo HTML `required`.
- `CreateItemServlet.java` valida i dati ricevuti (presenza, tipo, unicità del codice articolo tramite `ItemDAO.itemCodeExists()`, tipo e contenuto dell'immagine).
- Se la validazione ha successo, un nuovo oggetto `Item` viene creato e persistito tramite `ItemDAO.createItem()`.

7. **Requisito (Pagina VENDO - Form Creazione Asta):** Il secondo form mostra l'elenco degli articoli presenti nel database e disponibili per la vendita e dà la possibilità di selezionarne più di uno per creare un'asta che li comprende.

• **Implementazione:**

- Il form in `sell.html` presenta un elemento `<select multiple>` popolato dinamicamente da `SellPageServlet` con gli articoli disponibili dell'utente (ottenuti da `ItemDAO.findAvailableItemsBySellerId()`). Per ogni articolo vengono mostrati nome, codice e prezzo base.
- L'utente seleziona uno o più articoli e fornisce il rialzo minimo e la data/ora di scadenza.
- La richiesta viene gestita da `CreateAuctionServlet.java`.

8. **Requisito (Asta - Definizione):** Un'asta è formata da 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).

• **Implementazione:**

- *Bean:* `Auction.java` (record) include `initialPrice`, `minimumBidIncrement`, `deadline`.
- *Database:* Tabella `auction` e tabella di collegamento `auction_item_link`.
- *DAO:* `AuctionDAO.java`.
- *Rialzo Minimo Intero:* Il form di creazione asta in `sell.html` usa `<input type="number" step="1">`. `CreateAuctionServlet` valida che il valore sia un intero positivo e ≥ 1 . L'attributo `minimumBidIncrement` nel bean è un `double`, ma la logica di validazione assicura la conformità.
- *Scadenza:* Il form usa `<input type="datetime-local">`. `CreateAuctionServlet` valida che la scadenza sia futura (minimo 10 minuti).

9. **Requisito (Asta - Prezzo Iniziale):** Il prezzo iniziale dell'asta è ottenuto come somma del prezzo degli articoli compresi nell'offerta.

• **Implementazione:** `CreateAuctionServlet.java`, prima di creare l'oggetto `Auction`, calcola `initialPrice` sommando i `basePrice` degli oggetti `Item` selezionati, recuperati tramite `ItemDAO`.

10. **Requisito (Asta - Unicità Articolo):** Lo stesso articolo non può essere incluso in aste diverse (aperte). Una volta venduto, un articolo non deve essere più disponibile per l'inserimento in ulteriori aste.

• **Implementazione:** Il metodo `ItemDAO.findAvailableItemsBySellerId()` esclude gli articoli già presenti in aste con stato 'OPEN' o in aste 'CLOSED' con un vincitore (cioè venduti). Questo metodo è usato da `SellPageServlet` per popolare la lista degli item selezionabili nel form di creazione asta.

11. **Requisito (Pagina VENDO - Ordinamento Lista Aste):** La lista delle aste nella pagina VENDO è ordinata per data+ora crescente.

• **Implementazione:** Il metodo `AuctionDAO.findAuctionsByCreatorAndStatus()` utilizza la clausola SQL `ORDER BY creationTimestamp ASC`, interpretando "data+ora crescente" come data di creazione dell'asta crescente.

12. Requisito (Pagina VENDO - Info Lista Aste): 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.

- **Implementazione:** In `sell.html`:

- Per ogni asta, si iterano i suoi item (ottenuti con `itemDAO.findItemsByAuctionId()`) mostrando codice e nome.
- L'offerta massima è recuperata con `bidDAO.findHighestBidForAuction()`.
- Il tempo mancante è calcolato da `DateTimeUtils.getTimeRemaining()` usando il `loginTimestamp` (dalla sessione) e la `deadline` dell'asta.

13. Requisito (Pagina VENDO - Click su Asta): Cliccando su un'asta nell'elenco compare una pagina DETTAGLIO ASTA.

- **Implementazione:** Le righe (`<tr>`) delle tabelle delle aste in `sell.html` sono rese cliccabili tramite l'attributo `th:onclick`, che costruisce un JavaScript per reindirizzare a `AuctionDetailServlet` con il parametro `auctionId`.

14. Requisito (Pagina DETTAGLIO ASTA - Asta Aperta): Riporta tutti i dati dell'asta e la lista delle offerte (nome utente, prezzo offerto, data e ora dell'offerta) ordinata per data+ora decrescente.

- **Implementazione:**

- `AuctionDetailServlet.java` recupera l'oggetto `Auction` (da `AuctionDAO`), la lista dei suoi `Item` (da `ItemDAO`), e la lista delle `Bid` (da `BidDAO.findBidsByAuctionId()`, che ordina per timestamp decrescente). Recupera anche i nomi utente degli offerenti (`UserDAO`).
- Il template `auctionDetail.html` visualizza queste informazioni.

15. Requisito (Pagina DETTAGLIO ASTA - Bottone CHIUDI): Un bottone CHIUDI permette all'utente di chiudere l'asta se è giunta l'ora della scadenza.

- **Implementazione:**

- In `auctionDetail.html`, il bottone "Close Auction" appare condizionatamente (`th:if`) se l'asta è 'OPEN', l'utente corrente è il creatore, e la `deadline` è passata.
- Il form associato invia a `CloseAuctionServlet.java`.
- `CloseAuctionServlet` esegue validazioni simili e, se corrette, chiama `AuctionDAO.closeAuction()` per aggiornare lo stato dell'asta, registrando l'eventuale vincitore e il prezzo di aggiudicazione.

16. Requisito (Pagina DETTAGLIO ASTA - Asta Chiusa): Riporta tutti i dati dell'asta, il nome dell'aggiudicatario, il prezzo finale e l'indirizzo (fisso) di spedizione dell'utente.

- **Implementazione:** Se lo stato dell'asta è 'CLOSED', `auctionDetail.html` mostra i dati dell'asta. Se `Auction.winnerUserId` è presente, `AuctionDetailServlet` recupera i dettagli dell'utente vincitore (incluso `shippingAddress`) tramite `UserDAO` e li passa al template per la visualizzazione, insieme al `winningPrice`.

17. Requisito (Pagina ACQUISTO - Ricerca): Form di ricerca per parola chiave. All'invio, la pagina è aggiornata mostrando aste aperte (scadenza futura) la cui parola chiave compare nel nome o descrizione di almeno un articolo.

- **Implementazione:**

- `buy.html` ha un form con metodo GET che invia a `BuyPageServlet` con il parametro `keyword`.
- `BuyPageServlet` invoca `AuctionDAO.findOpenAuctionsByKeyword()`. La query SQL sottostante filtra per `status = 'OPEN'`, `deadline > NOW()`, e cerca la keyword tramite `LIKE` nei campi nome e descrizione degli item associati all'asta.

18. Requisito (Pagina ACQUISTO - Ordinamento Risultati Ricerca): La lista è ordinata in modo decrescente in base al tempo mancante alla chiusura.

- **Implementazione:** `AuctionDAO.findOpenAuctionsByKeyword()` ordina i risultati con `ORDER BY a.deadline ASC`. Un tempo mancante minore (più urgente) corrisponde a una deadline più vicina (ascendente), quindi l'ordinamento è corretto.

19. Requisito (Pagina ACQUISTO - Click su Asta Aperta): Cliccando su un'asta aperta compare la pagina OFFERTA con dati articoli, offerte (data+ora decrescente), e form per la propria offerta.

- **Implementazione:**

- In `buy.html`, le righe delle aste nei risultati sono cliccabili (`th:onclick`) e reindirizzano a `OfferPageServlet` con l'ID dell'asta.
- `OfferPageServlet.java` recupera l'asta, i suoi item, e le offerte (ordinate da `BidDAO`).
- `offer.html` visualizza queste informazioni e il form per l'offerta.

20. Requisito (Pagina OFFERTA - Logica Offerta): L'offerta deve essere superiore all'offerta massima corrente di un importo pari almeno al rialzo minimo.

- **Implementazione:** Il form in `offer.html` invia a `PlaceBidServlet.java`. Questa servlet calcola l'offerta minima richiesta (basandosi sul prezzo iniziale o sull'offerta più alta esistente più il rialzo minimo) e valida l'offerta dell'utente. Se valida, `BidDAO.createBid()` salva la nuova offerta.

21. Requisito (Pagina OFFERTA - Aggiornamento Dopo Offerta): Dopo l'invio dell'offerta la pagina OFFERTA mostra l'elenco delle offerte aggiornate.

- **Implementazione:** `PlaceBidServlet` reindirizza a `OfferPageServlet` (passando l'ID asta e eventuali messaggi) dopo aver processato l'offerta. Questo fa sì che la pagina dell'offerta venga ricaricata con la lista aggiornata delle offerte.

22. Requisito (Pagina ACQUISTO - Aste Aggiudicate): La pagina ACQUISTO contiene anche un elenco delle offerte aggiudicate all'utente con i dati degli articoli e il prezzo finale.

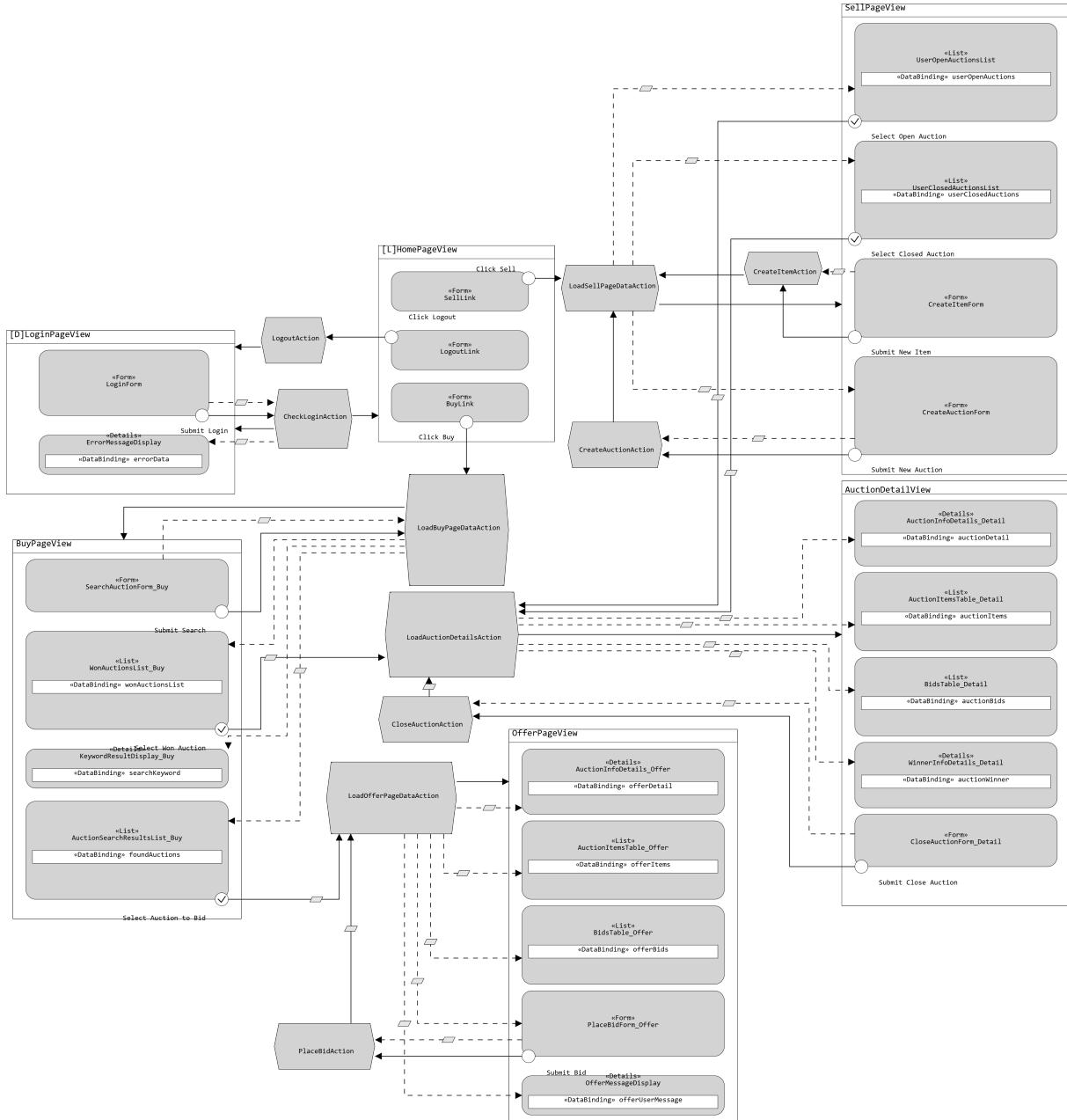
- **Implementazione:** `BuyPageServlet.java` usa `AuctionDAO.findWonAuctionsByUser()` per ottenere queste aste. `buy.html` le visualizza. Cliccando su una riga di un'asta vinta si viene reindirizzati a `AuctionDetailServlet` per i dettagli.

3.3 Versione con JavaScript (RIA)

23. **Requisito (Unica Pagina):** Dopo il login, l'intera applicazione è realizzata con un'unica pagina.
- **Implementazione:** La pagina `app.html` funge da "guscio" (shell) dell'applicazione. Il contenuto delle diverse sezioni (acquisto, vendita, dettaglio asta, offerta) viene caricato e rende- rizzato dinamicamente all'interno di `div` contenitori specifici da JavaScript, senza ricaricare l'intera pagina `app.html`. La logica di orchestrazione è in `appOrchestrator.js`.
24. **Requisito (Logica Visualizzazione Iniziale):** Se l'utente accede per la prima volta l'applica- zione 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'a- sta; altrimenti mostra il contenuto della pagina ACQUISTO con l'elenco (eventualmente vuoto) delle asta su cui l'utente ha cliccato in precedenza e che sono ancora aperte.
- **Implementazione:** Gestita in `appOrchestrator.js` (funzione `_navigateToInitialViewLogic`):
 - Vengono lette le chiavi `lastAction(userId_X)` e `visitedAuctions(userId_X)` da `localStorage` (con controllo di scadenza).
 - Se `lastAction` è '`sell_related`', viene visualizzata la vista 'sell' e `lastAction` viene rimosso/fatto scadere.
 - Altrimenti, se esistono `visitedAuctionIds` validi, viene visualizzata la vista 'buy', pas- sando questi ID a `buyPage.js`. Quest'ultimo poi effettua una chiamata a `BuyPageServlet` con il parametro `ids` per caricare solo le asta visitate che sono ancora aperte.
 - Altrimenti (primo accesso, o dati non presenti/scaduti), viene visualizzata la vista 'buy'.
25. **Requisito (Memorizzazione Lato Client per 1 Mese):** L'informazione dell'ultima azione compiuta e delle asta visitate è memorizzata a lato client per la durata di un mese.
- **Implementazione:**
 - `appOrchestrator.js` implementa le funzioni `_setLocalStorageItemWithExpiry` e `_getLocalStorageItem`.
 - Queste utility memorizzano dati in `localStorage` come oggetti JSON contenenti il valore effettivo e un timestamp `expiresAt` (impostato a 30 giorni dal momento della scrittura).
 - Le chiavi in `localStorage` sono rese utente-specifiche concatenando l'ID dell'utente alla chiave base (es., `lastAction(userId_1)`).
 - `sellPage.js` (dopo la creazione di un'asta) e `auctionDetail.js/offer.js` (alla visuali- zazione di un'asta) utilizzano queste utility per scrivere/leggere `lastAction` e `visitedAuctions`.
 - Al logout, si tenta di rimuovere questi dati specifici dell'utente da `localStorage`.
26. **Requisito (Interazioni Asincrone):** Ogni interazione dell'utente è gestita senza ricaricare com- pletamente la pagina, ma produce l'invocazione asincrona del server e l'eventuale modifica solo del contenuto da aggiornare a seguito dell'evento.
- **Implementazione:**
 - Tutte le comunicazioni con il server (ricerche, creazioni, offerte, caricamento dati) sono gestite tramite chiamate AJAX asincrone, utilizzando la funzione `makeCall` definita in `utils.js` (che internamente usa `XMLHttpRequest`).
 - Le servlet lato server sono state adattate per rispondere con dati in formato JSON. La li- breria Gson è utilizzata per la serializzazione, con adapter custom per tipi come `Timestamp` (`TimestampAdapter.java`) e `Item` (`ItemSerializer.java`, per gestire la serializzazione dell'immagine come Base64).
 - I moduli JavaScript (`buyPage.js`, `sellPage.js`, `auctionDetail.js`, `offer.js`) ricevono il JSON, effettuano il parsing e aggiornano dinamicamente le sezioni pertinenti del DOM all'interno di `app.html`, evitando ricaricamenti completi della pagina.

3.4 Design (IFML)

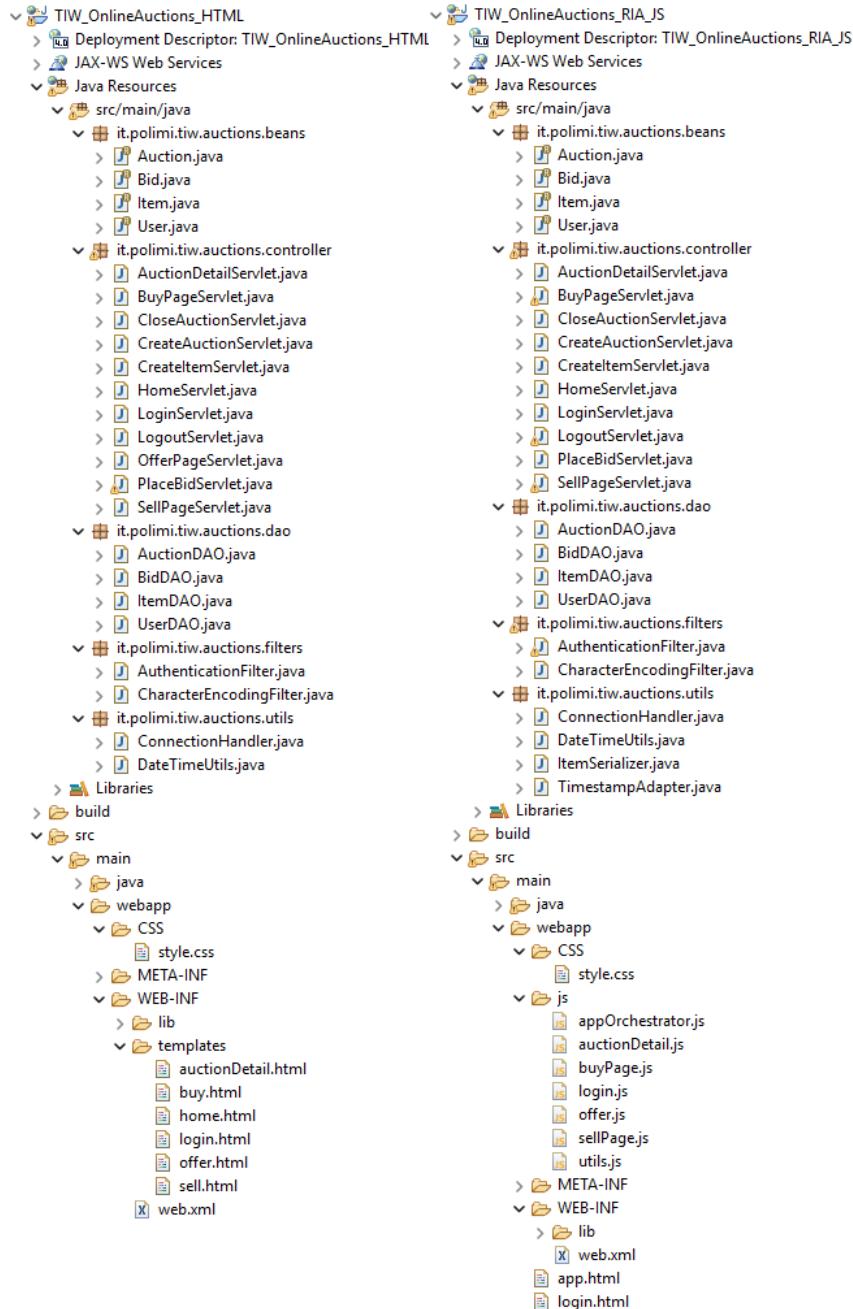
3.4.1 Versione HTML pura



3.4.2 Versione con JavaScript (RIA)

Per quanto riguarda la versione Rich Internet Application (RIA) del progetto, l'approccio di modellazione con IFML subisce una trasformazione significativa a causa della natura Single Page Application (SPA). Invece di multipli ViewContainer che rappresentano pagine HTML distinte, avremmo un ViewContainer principale (es. AppShellView) che rappresenta il file app.html. Le diverse "viste" logiche (Acquisto, Vendita, Dettaglio Asta, Offerta) diventerebbero ViewComponent dinamici contenuti all'interno di questo AppShellView. La logica di navigazione e visualizzazione, precedentemente gestita da servlet che effettuavano forward o redirect a diverse pagine, è ora incapsulata principalmente lato client, nel codice JavaScript (specificamente in appOrchestrator.js e nei moduli di pagina come buyPage.js, sellPage.js, ecc.). Questo "orchestratore client-side" potrebbe essere rappresentato in IFML come un Action centrale lato client (es. AppOrchestratorAction) che risponde agli eventi di navigazione (es. click sui link nell'header) e agli eventi di inizializzazione della pagina. Questa Action client-side sarebbe responsabile di mostrare/nascondere e popolare i vari ViewComponent dinamici all'interno dell'AppShellView senza ricaricare l'intera pagina. Le interazioni con il backend (le tue attuali Servlet) verrebbero modellate come Action server-side richiamate tramite chiamate AJAX asincrone. Queste Action server-side non restituirebbero più intere pagine HTML, ma dati strutturati (nel nostro caso, JSON). Ci sarebbero quindi DataFlow di tipo JSON tra le Action server-side e i ViewComponent dinamici (o meglio, la logica JavaScript che li gestisce), che poi si occuperebbero di aggiornare il DOM in modo selettivo. La gestione dello stato, come lastAction e visitedAuctions, che nella versione HTML pura potrebbe coinvolgere la sessione server o parametri URL, nella RIA è esplicitamente gestita lato client tramite localStorage (con logica di scadenza e specificità per utente), e potrebbe essere modellata come un'interazione tra l'AppOrchestratorAction e un ExternalComponent rappresentante il ClientLocalStorage. In sintesi, mentre i componenti di input/output (form, tavole) e le logiche di business server-side rimangono concettualmente simili, il diagramma IFML per la RIA mostrerebbe un flusso di controllo e dati molto più concentrato sul client e sull'interazione asincrona, con un unico ViewContainer persistente e aggiornamenti dinamici del suo contenuto

3.5 Components List



3.6 Sequence Diagrams

3.6.1 Versione HTML pura

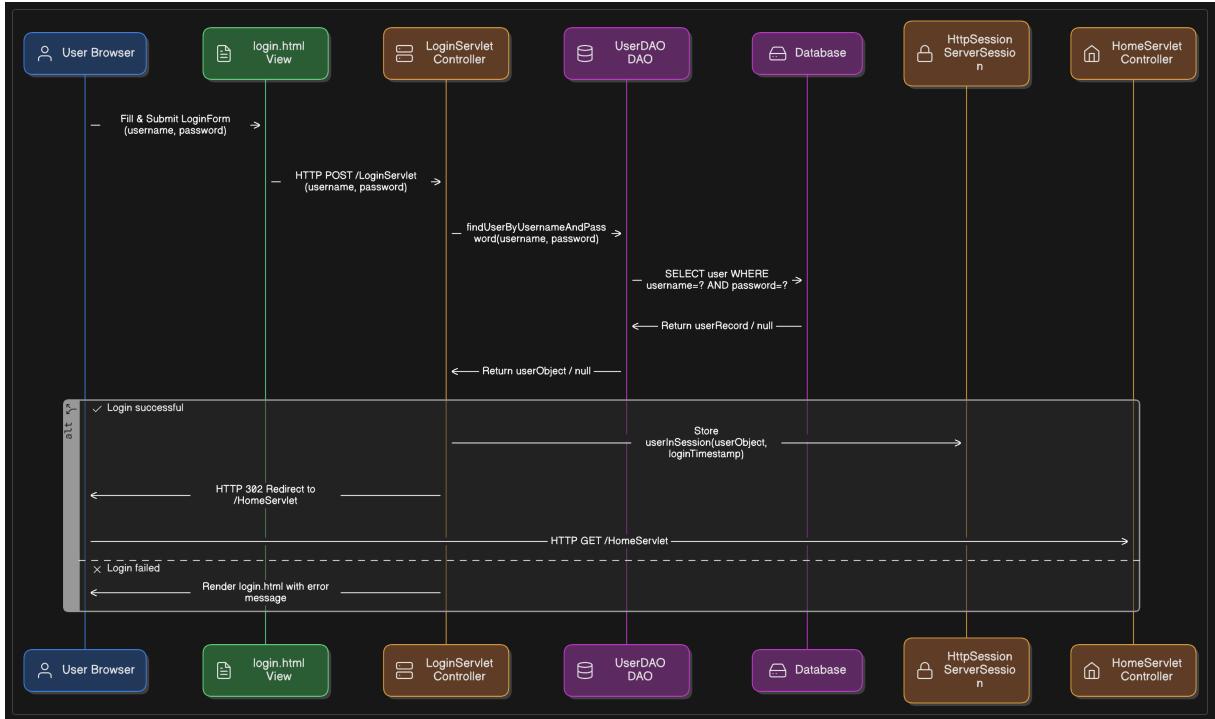


Figura 1: Login

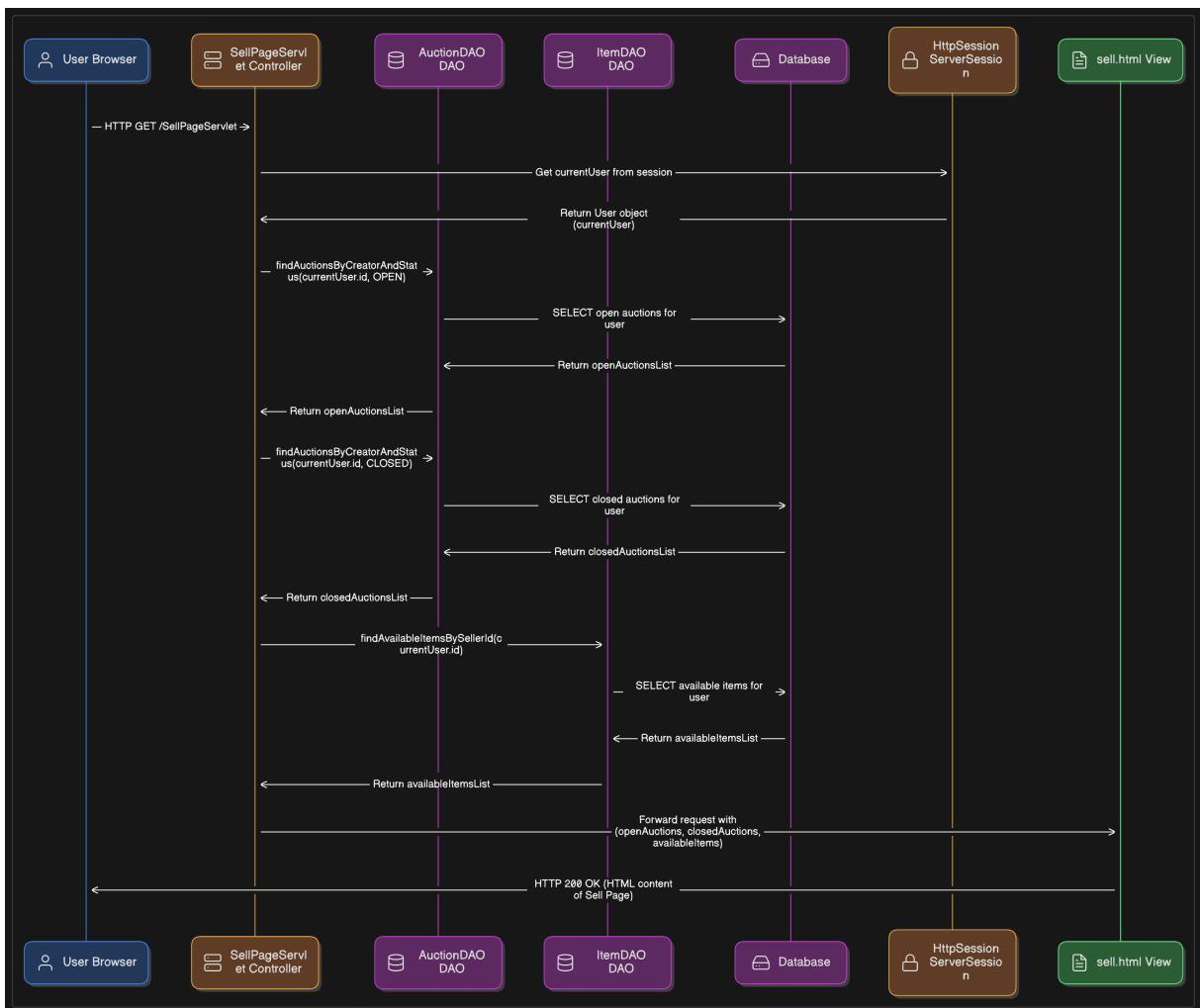


Figura 2: View Sell Page

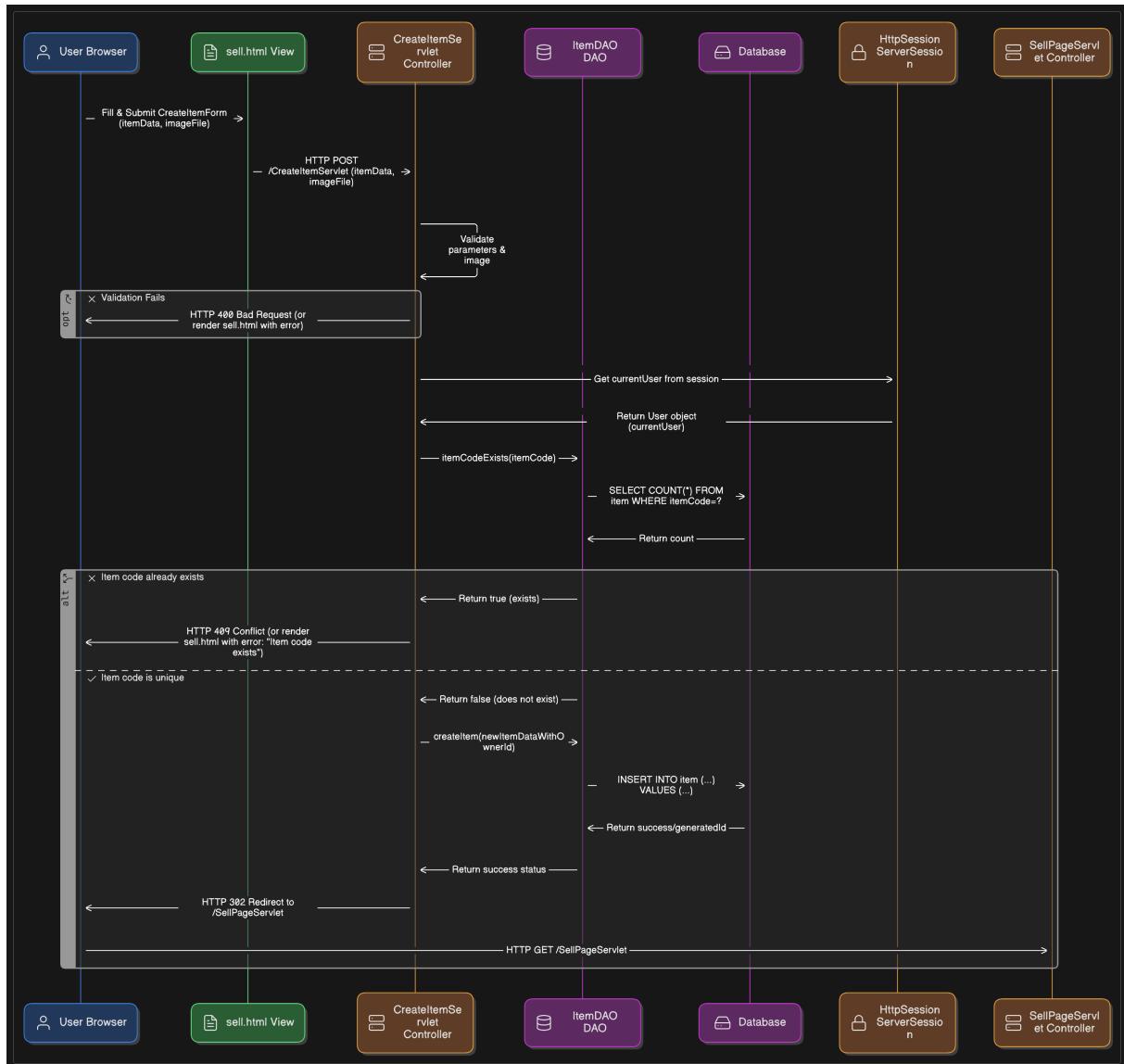


Figura 3: Create New Item

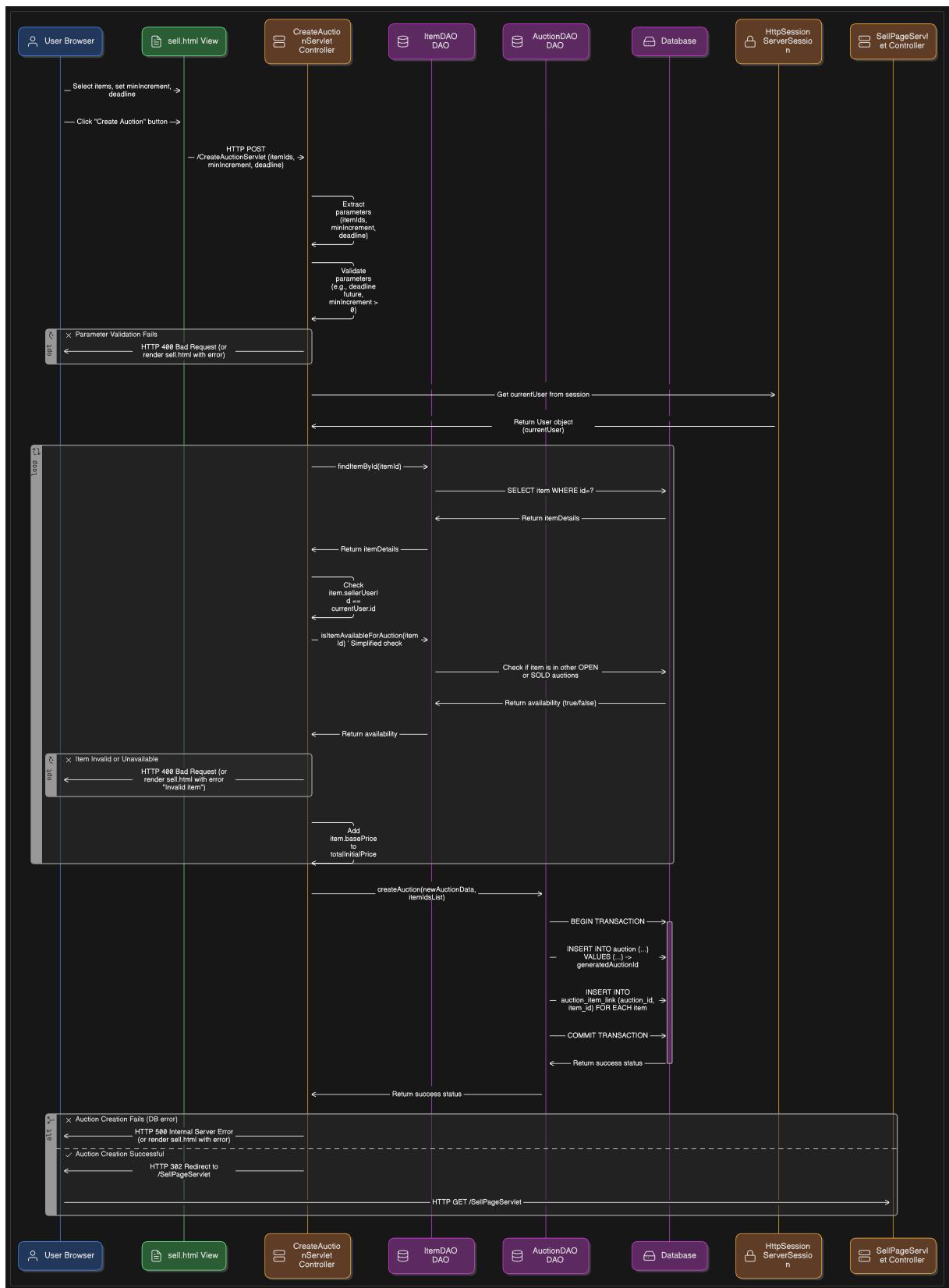


Figura 4: Create New Auction

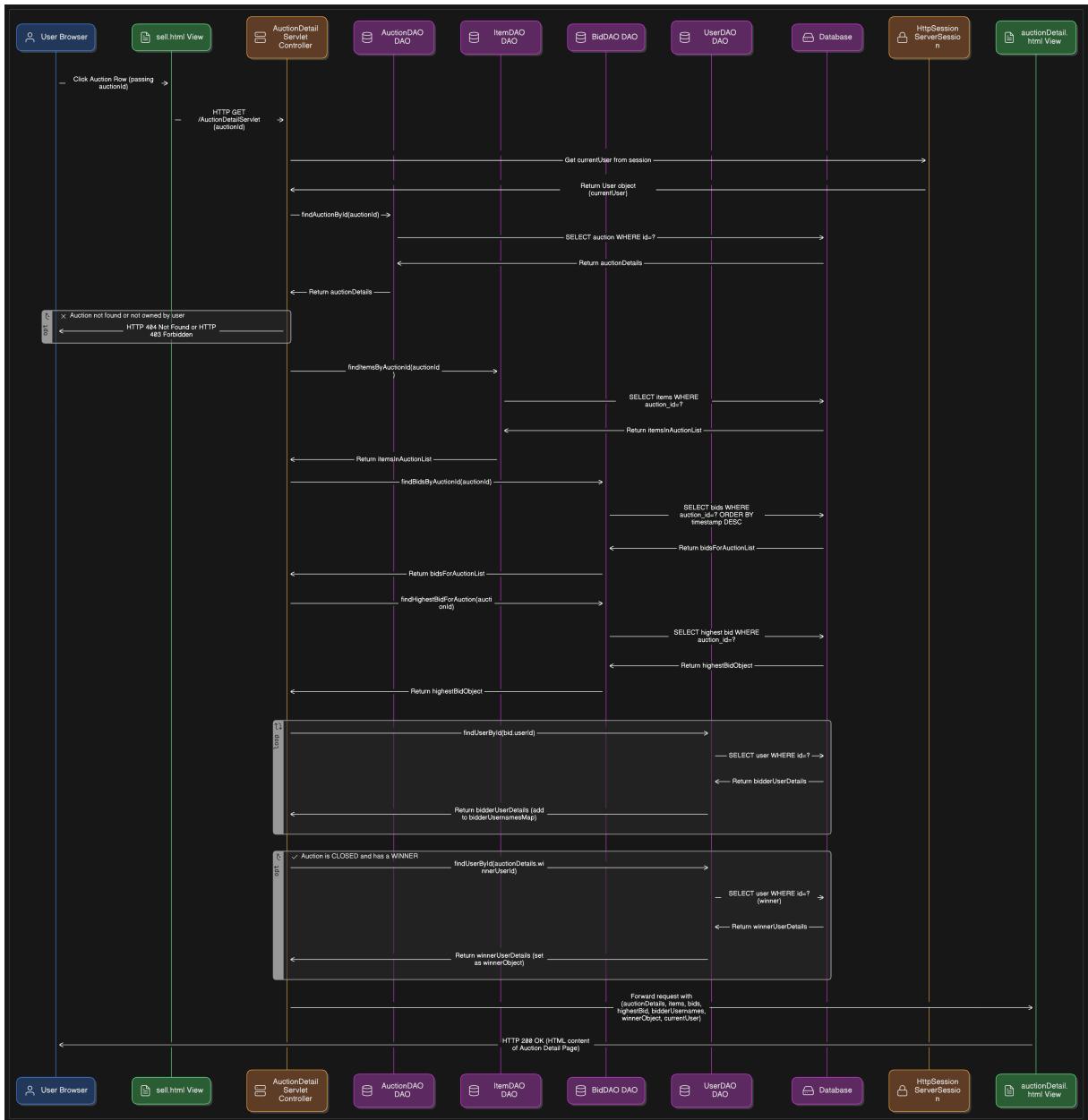


Figura 5: View Detail

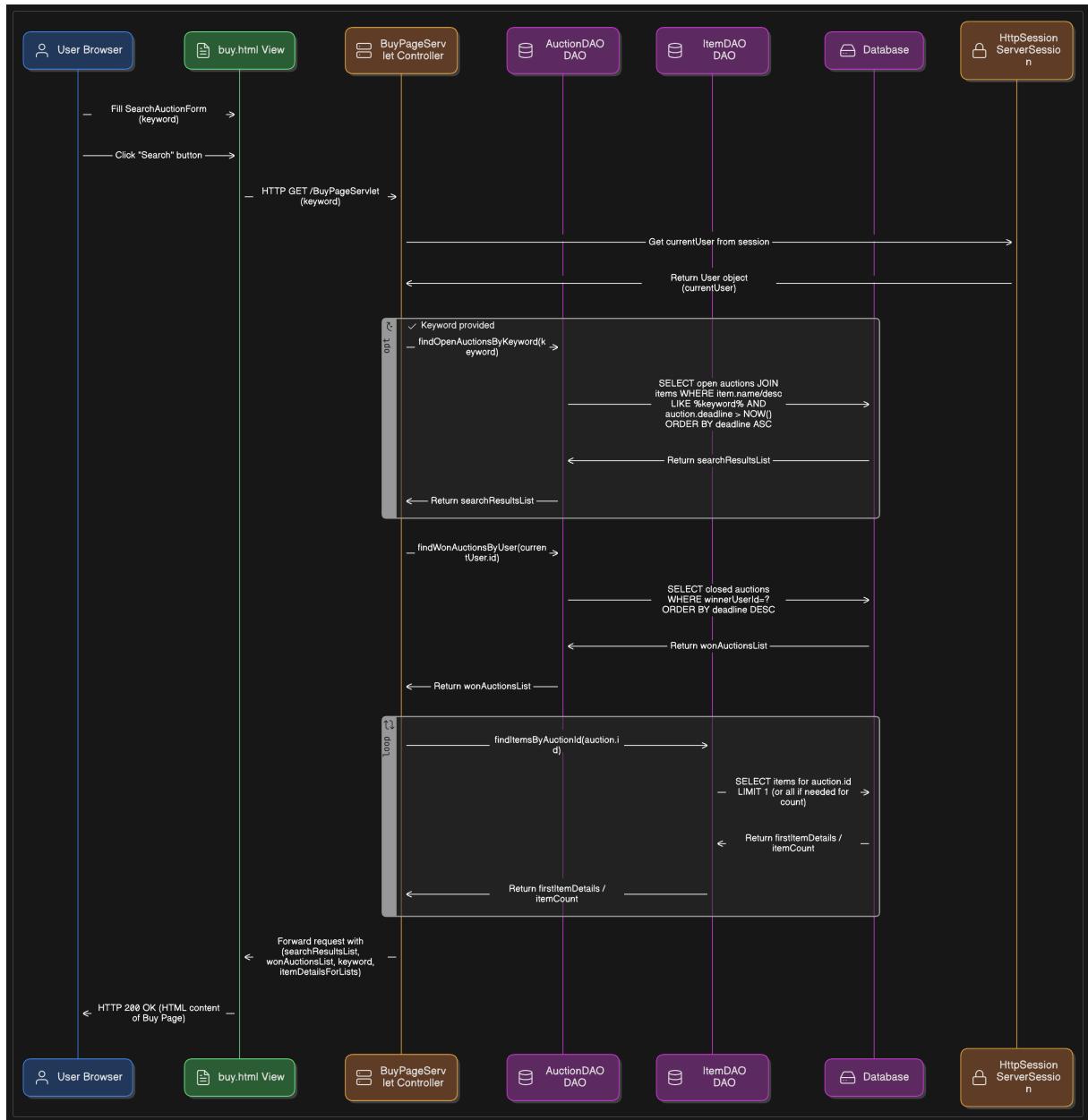


Figura 6: Search Auction

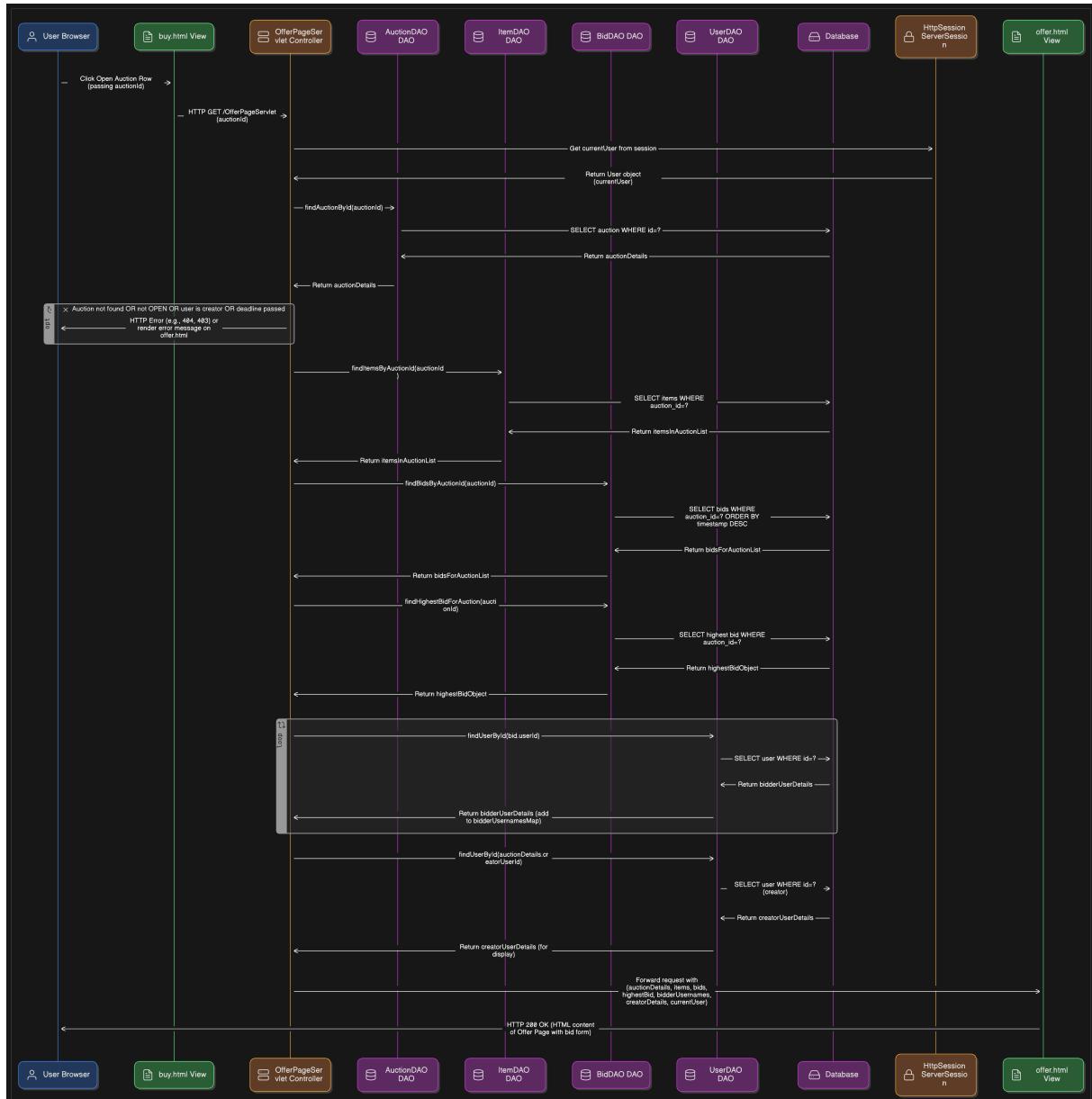


Figura 7: View Offer Page

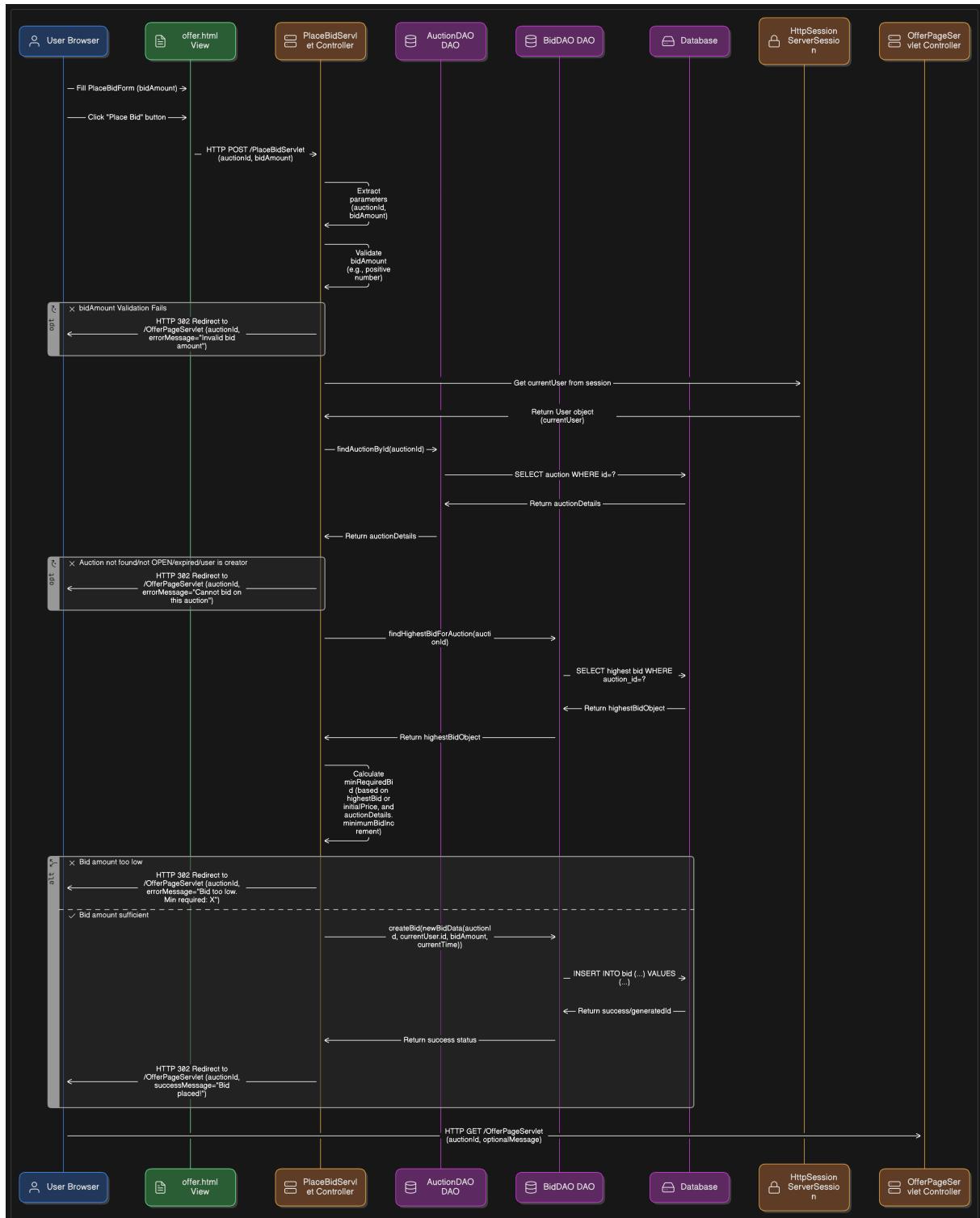


Figura 8: Make Offer

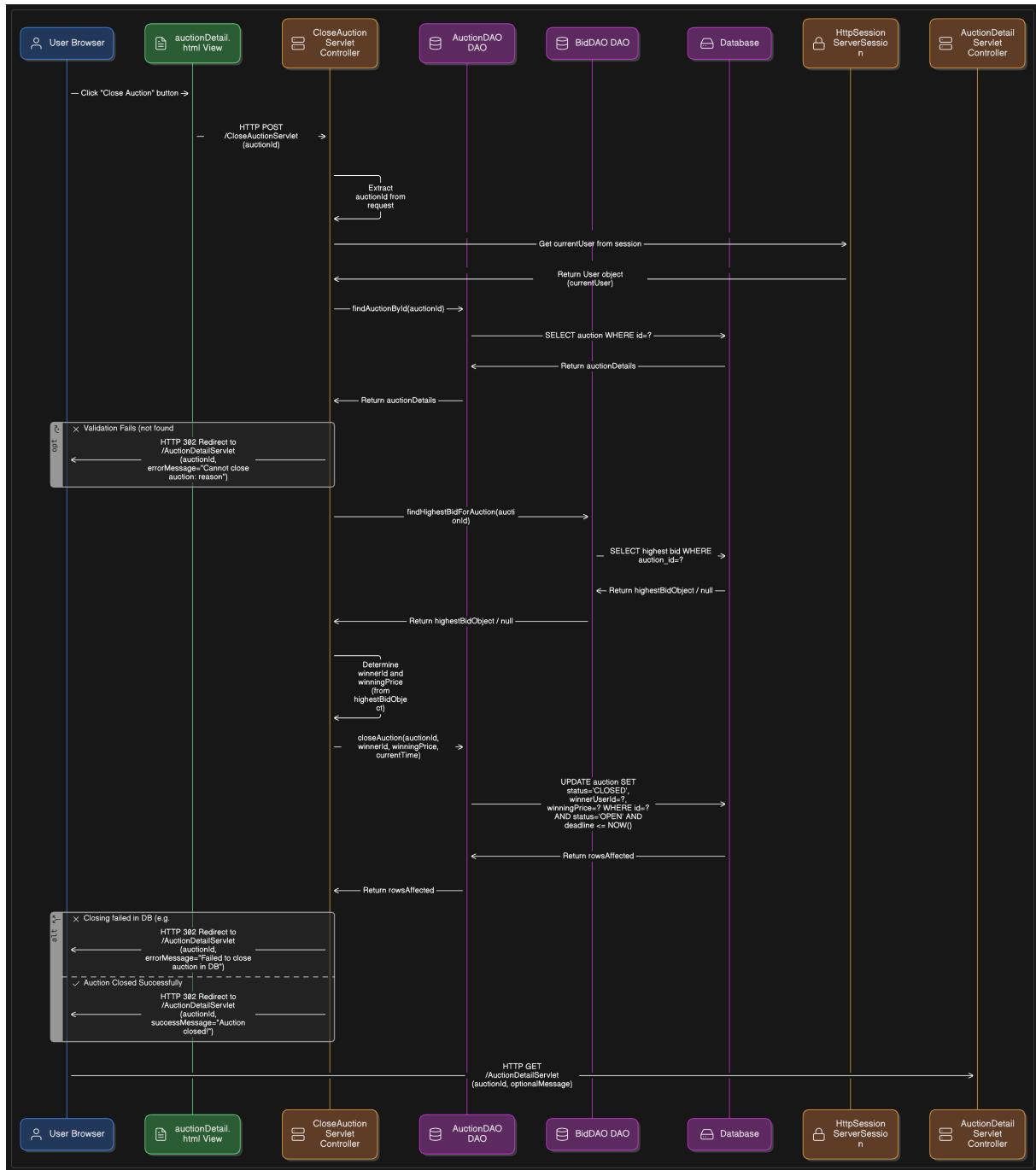


Figura 9: Close Auction

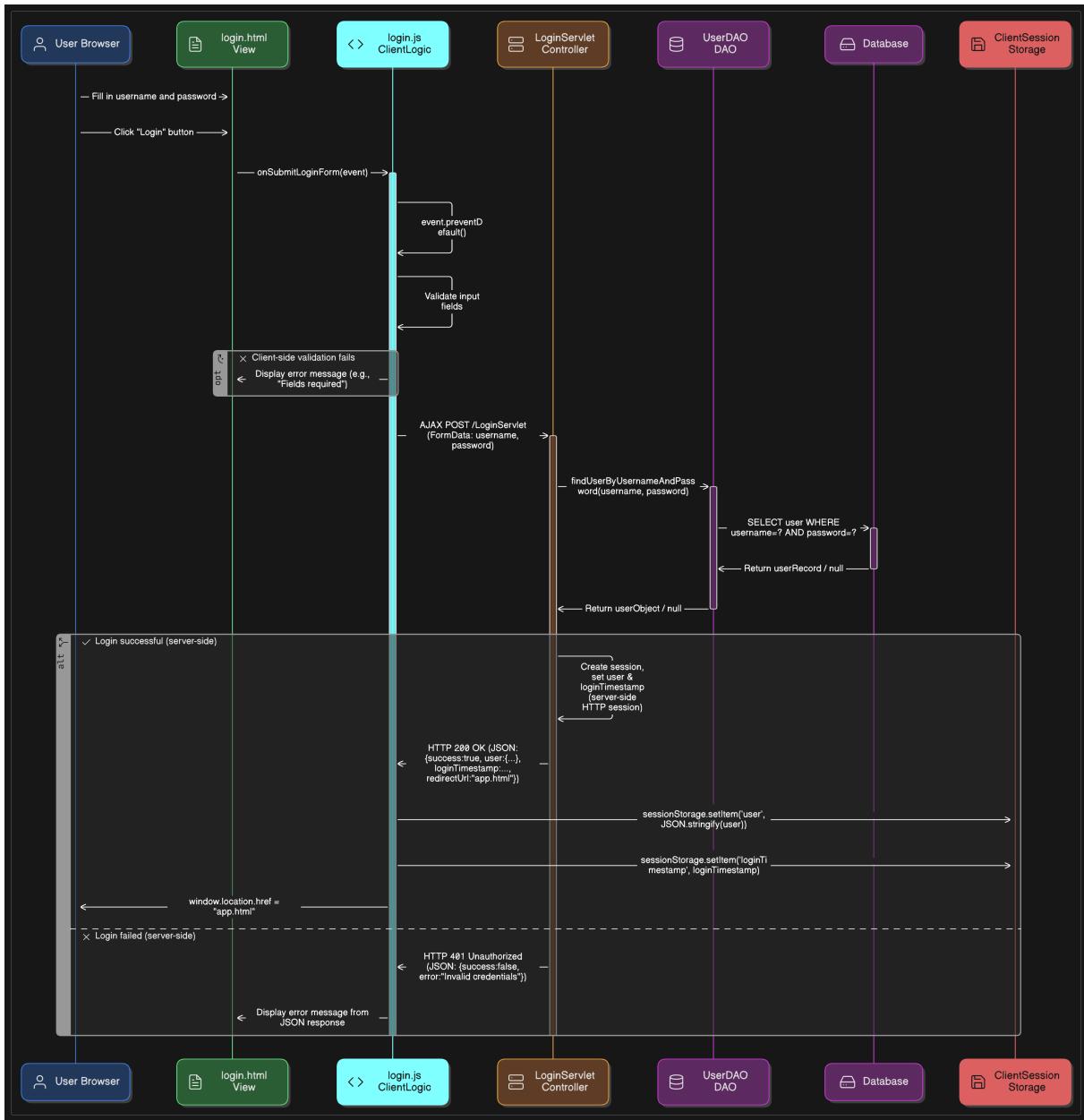
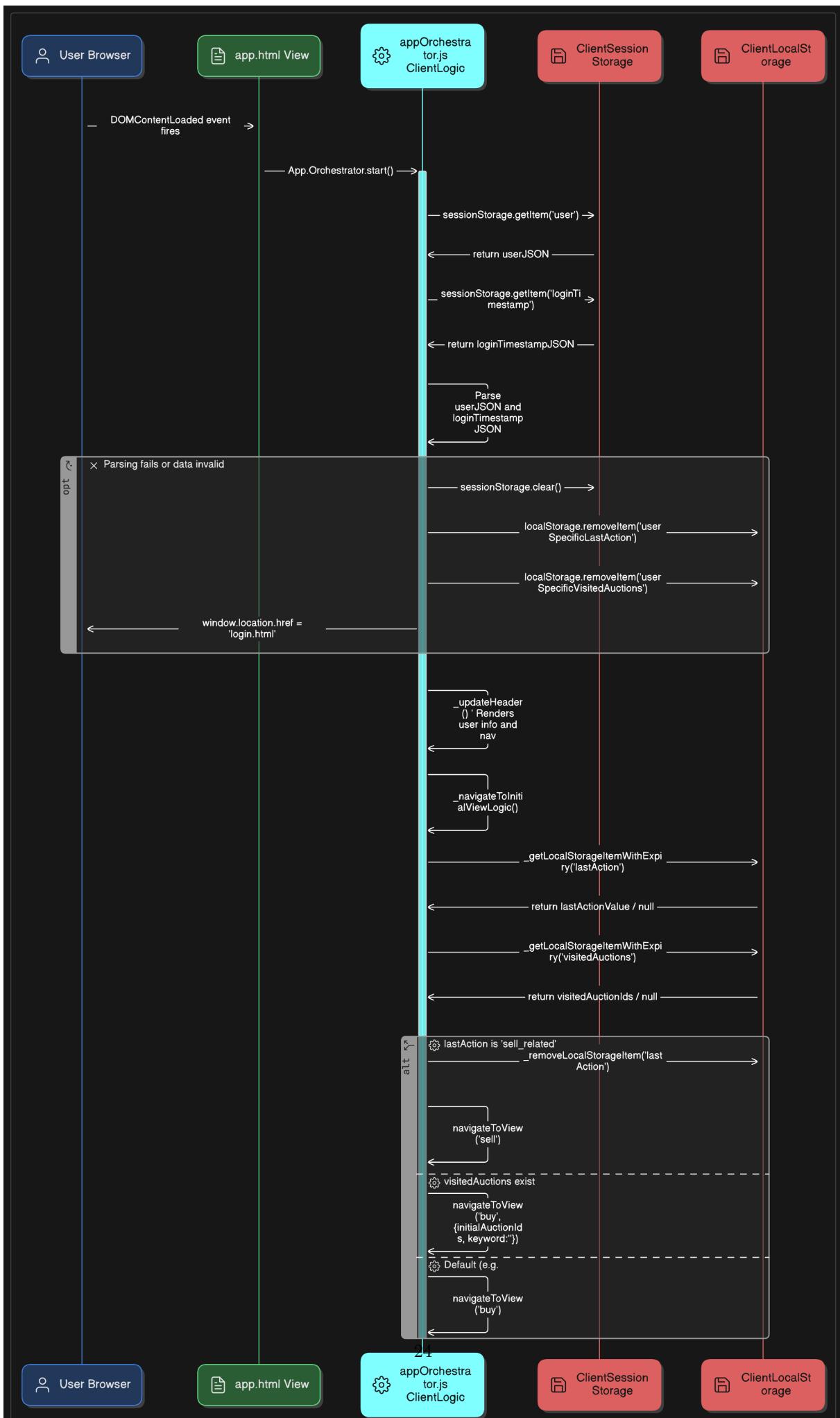


Figura 10: Login RIA

3.6.2 Versione con JavaScript (RIA)



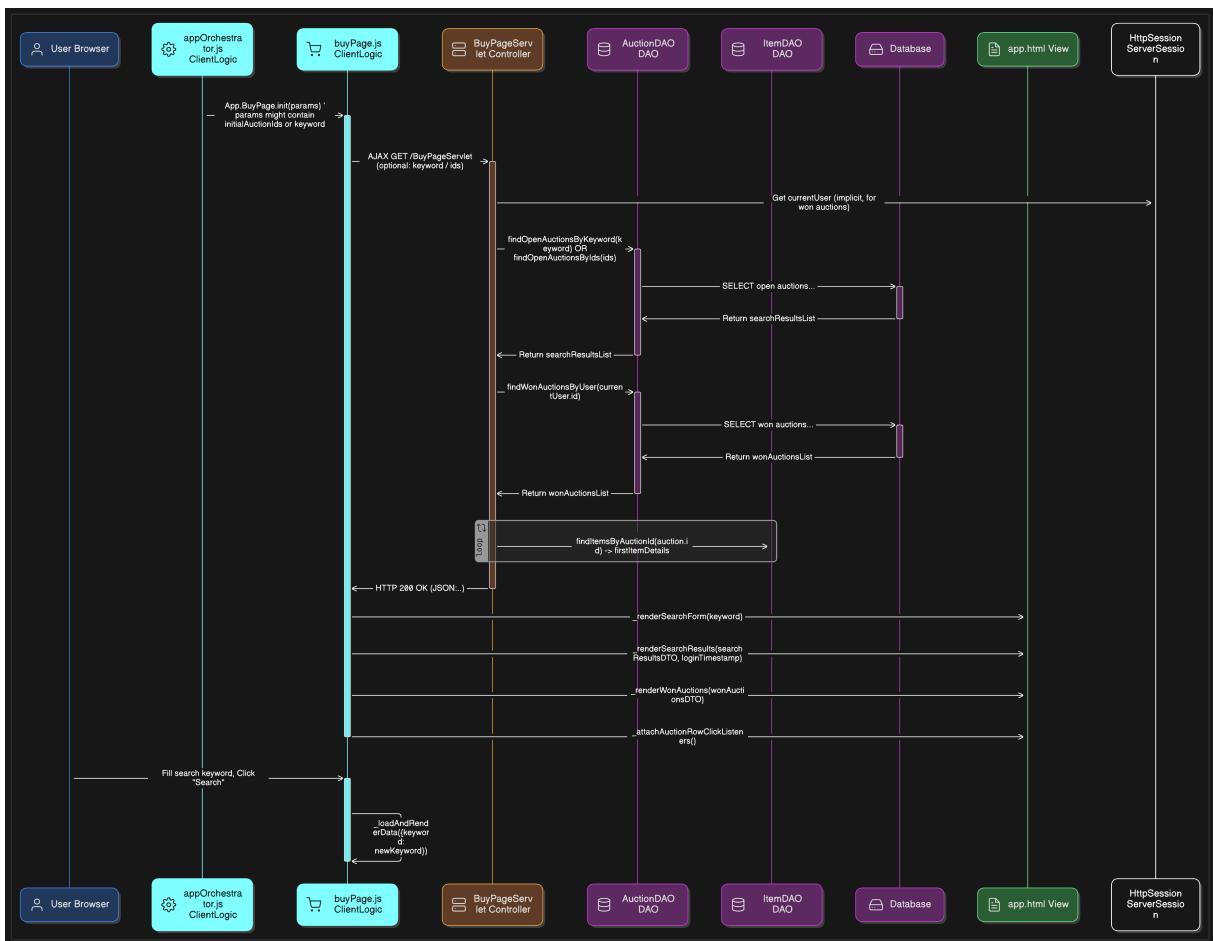


Figura 12: Buy page upload RIA

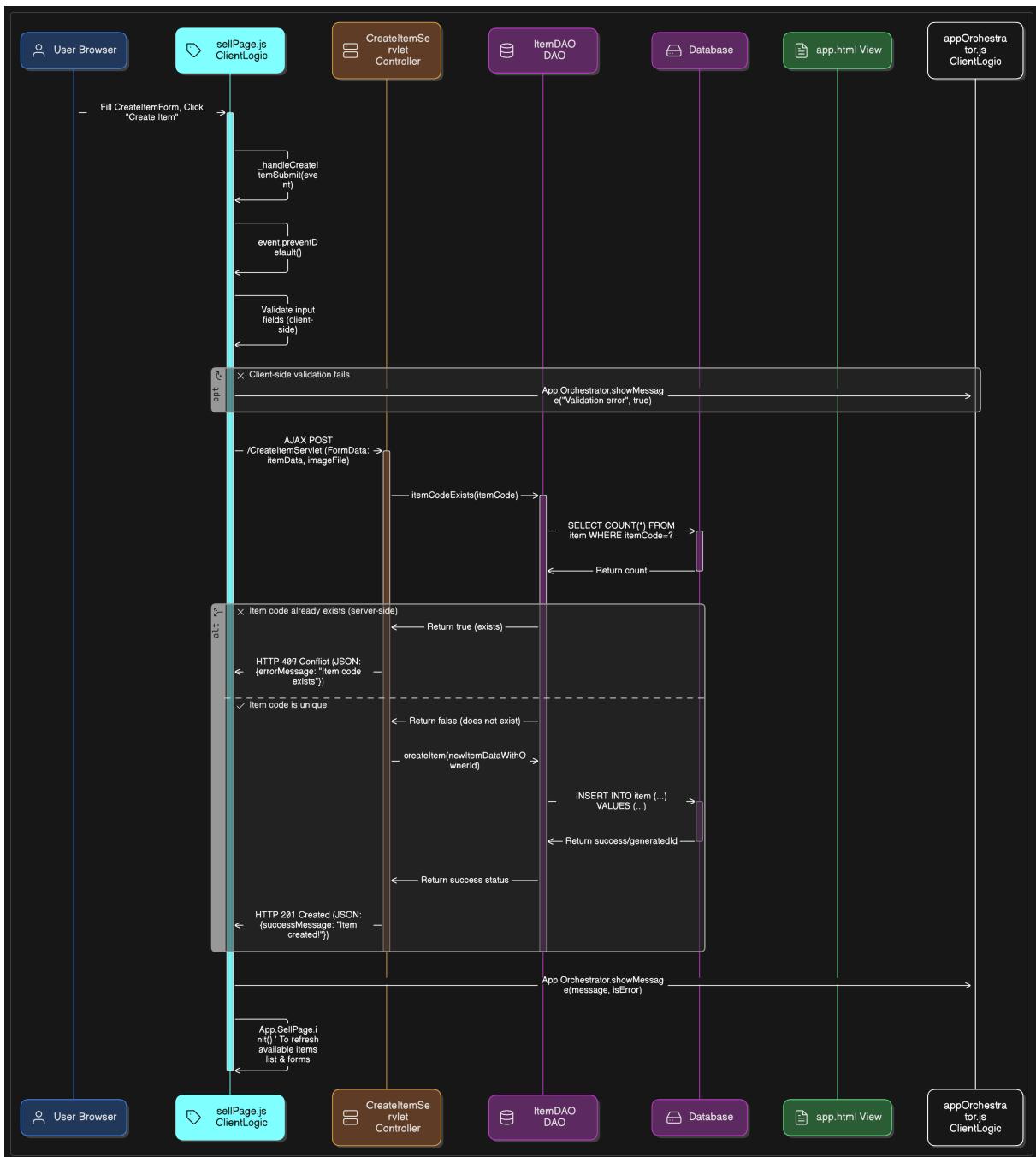


Figura 13: Create New Item RIA

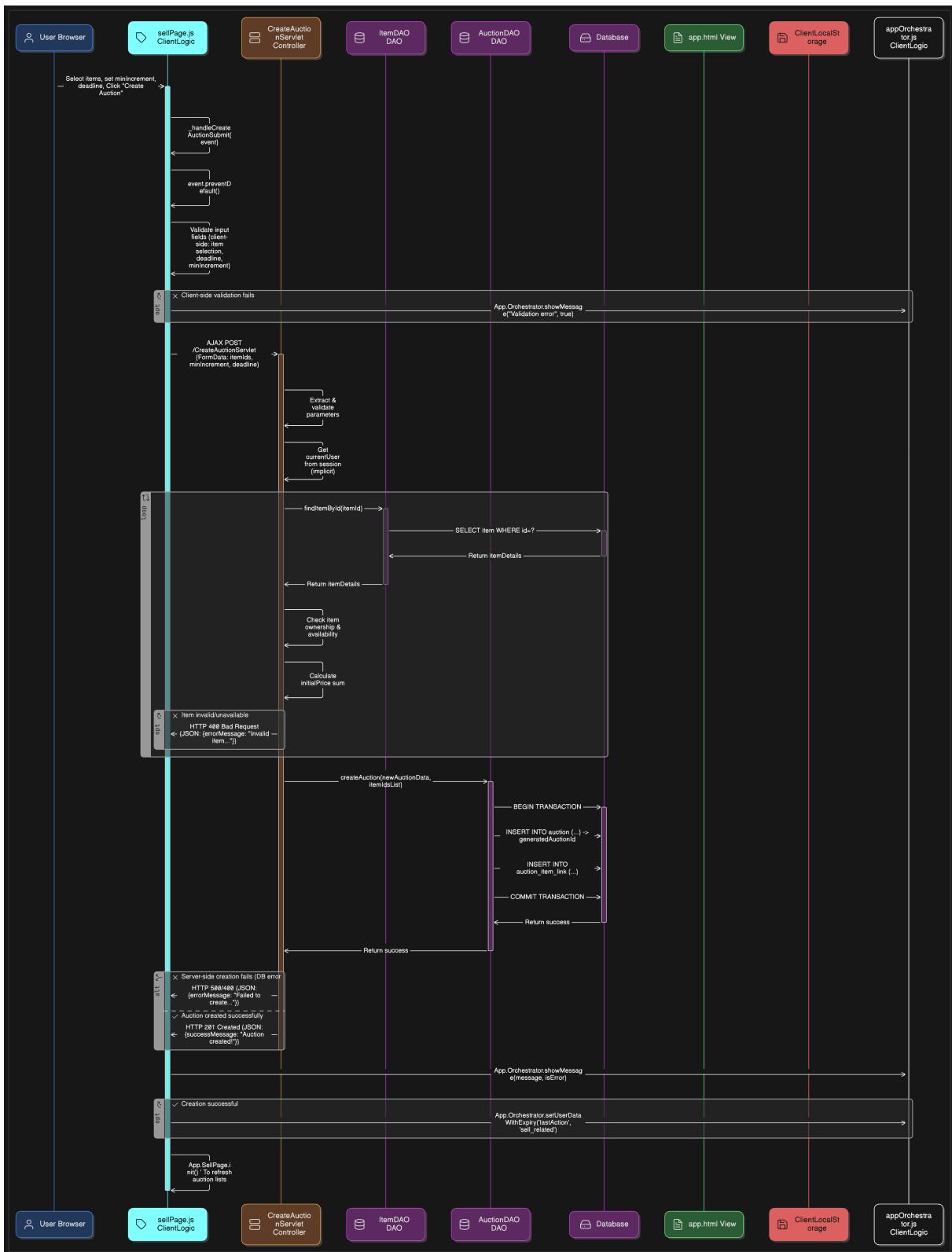


Figura 14: Create New Auction RIA

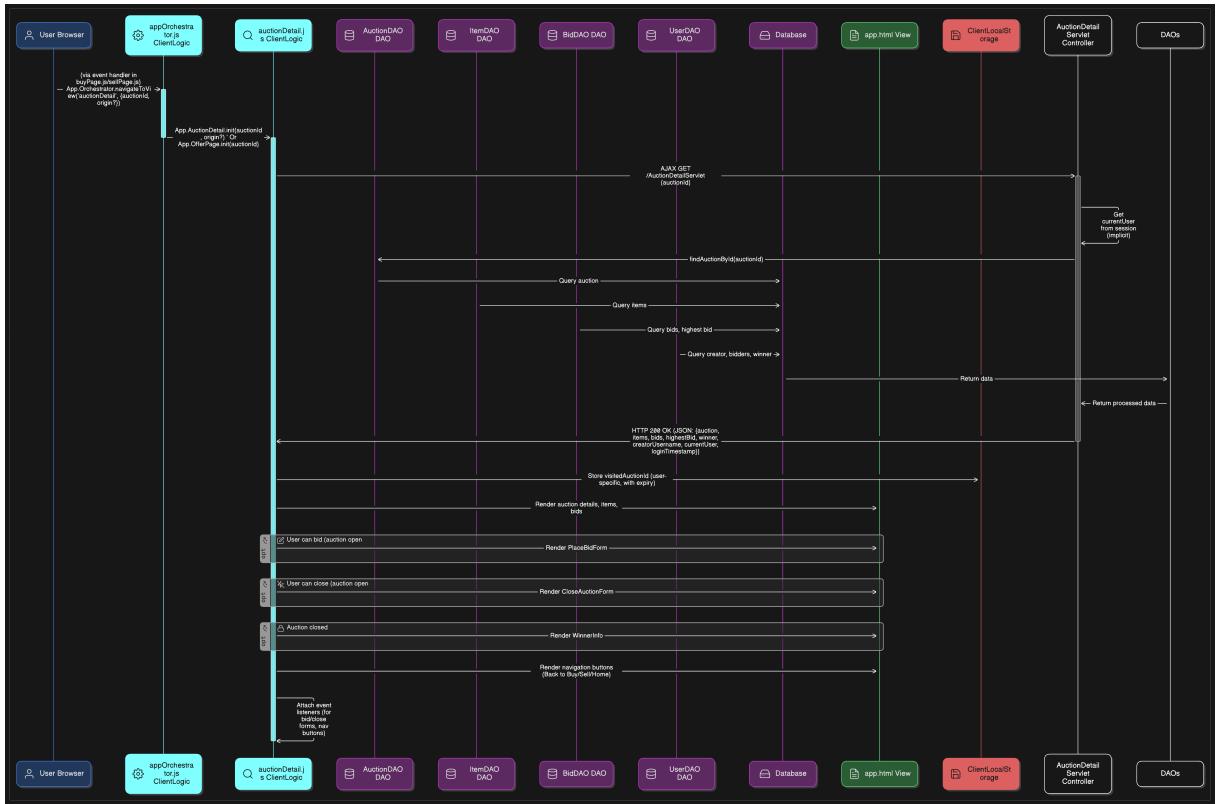


Figura 15: View Detail RIA

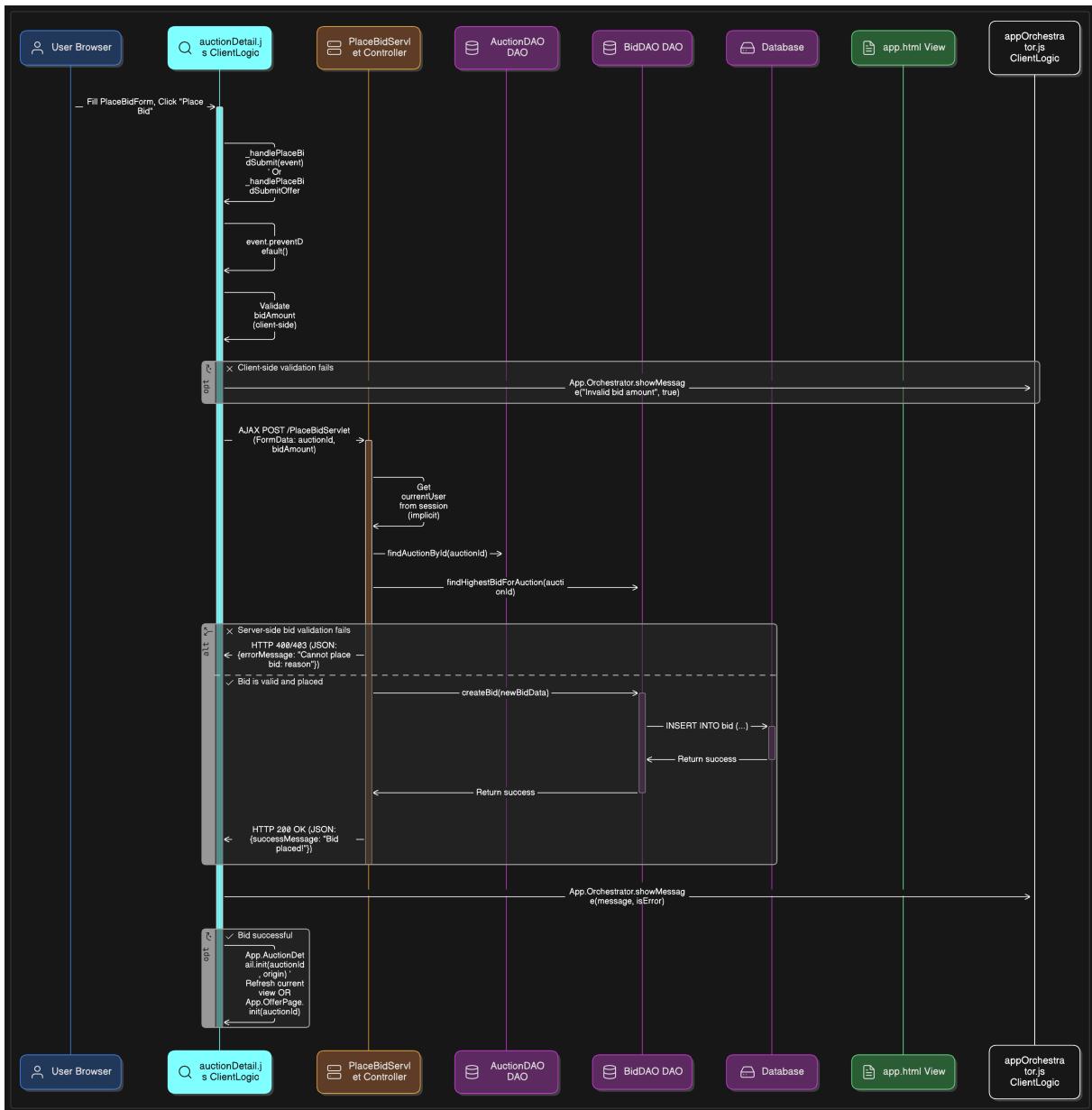


Figura 16: Make Offer RIA

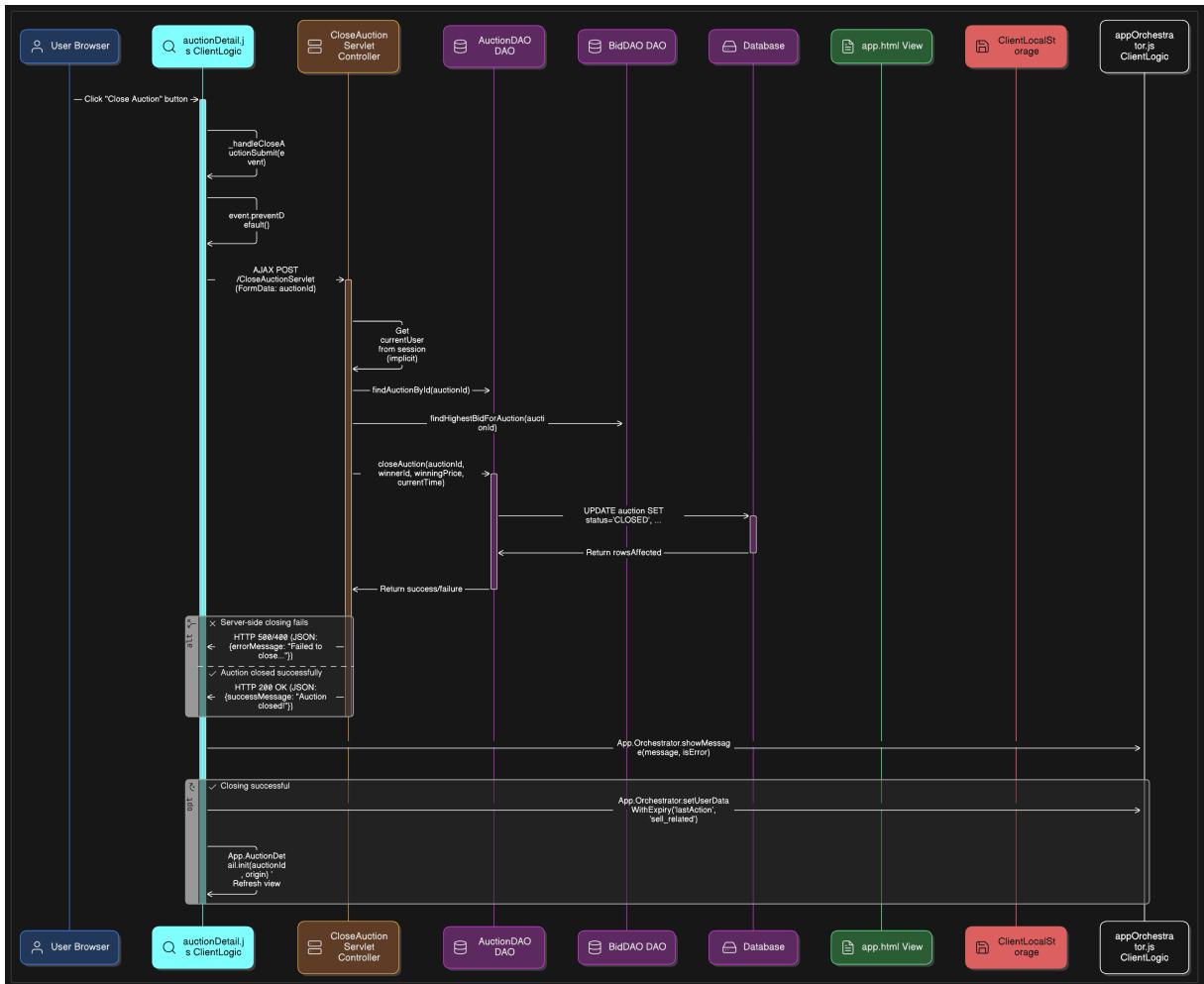


Figura 17: Close Auction RIA