



# Relazione Progetto: Piattaforma di Aste Online asteRoy

## 1 Introduzione

Questo capitolo serve a presentare il progetto.

### 1.1 Obiettivo del Progetto

L'obiettivo di questo progetto è stata la realizzazione di una piattaforma web completa e funzionale per la gestione di aste online, ispirata a sistemi moderni come eBay. L'applicazione, sviluppata con il framework Django, è stata progettata per connettere due tipologie di utenti, Venditori e Acquirenti, in un ambiente sicuro e intuitivo. Le funzionalità principali includono la creazione di aste, un sistema di offerte in tempo reale tramite WebSockets, la gestione di feedback per la costruzione della reputazione degli utenti e un sistema di ricerca avanzata per esplorare i prodotti.

### 1.2 Traccia d'Esame

Il progetto consiste nella realizzazione di una piattaforma web per la gestione di aste online, pensata per mettere in comunicazione venditori e acquirenti in modo semplice, sicuro e trasparente.

Il sistema prende ispirazione da piattaforme esistenti come eBay o Vinted, con l'obiettivo di offrire un'esperienza d'uso chiara ed efficace, valorizzando la reputazione degli utenti e facilitando l'esplorazione dei prodotti.

L'accesso al sito è consentito sia agli utenti anonimi che a quelli registrati. Gli utenti anonimi possono visualizzare le aste in corso, consultare i dettagli dei prodotti e leggere i profili pubblici dei venditori, ma non possono partecipare attivamente al sistema.

Gli utenti registrati, al momento della registrazione, devono selezionare un ruolo tra i seguenti:

- **Acquirente:** può partecipare alle aste effettuando offerte, salvare prodotti nella propria lista dei desideri (preferiti), visualizzare lo storico delle offerte e delle aste vinte, e lasciare feedback ai venditori e ai prodotti acquistati.
- **Venditore:** può creare nuove aste, gestirne lo stato, visualizzare lo storico delle aste concluse e consultare i feedback ricevuti.

Questa distinzione dei ruoli è pensata per semplificare l'esperienza dell'utente, offrendo un'interfaccia coerente con le azioni previste per ciascun tipo di profilo.

Il venditore, tramite il proprio pannello personale, può creare nuove aste inserendo tutte le informazioni necessarie: titolo, descrizione, immagine, categoria, prezzo base, durata dell'asta e rilancio minimo. Finché non viene effettuata la prima offerta, l'asta può essere modificata o annullata. Le aste in corso e concluse sono gestibili attraverso un'apposita area dedicata.

L'acquirente può visualizzare tutte le aste disponibili, partecipare effettuando rilanci e, se interessato, "salvare" un prodotto in una **lista dei desideri**. Ogni rilancio deve rispettare il rilancio minimo e supera automaticamente l'offerta precedente. Il sistema invia notifiche per aggiornare l'utente sull'andamento delle aste a cui partecipa, in particolare quando viene superato.

Alla scadenza del tempo previsto, l'asta si conclude automaticamente e il prodotto viene assegnato al miglior offerente, che riceve una notifica con le istruzioni per il completamento dell'acquisto. Il venditore, a sua volta, riceve i dettagli dell'acquirente vincitore.

Un aspetto centrale della piattaforma è la gestione dei **feedback**, che possono essere lasciati sia per il **venditore** che per il **prodotto acquistato**. Al termine di un'asta, l'acquirente può esprimere una valutazione da **1 a 5 stelle**, accompagnata da un eventuale commento testuale. I feedback rivolti al venditore contribuiscono a costruirne la reputazione e sono pubblicamente visibili nel suo profilo. Le recensioni sui prodotti, invece, aiutano gli altri utenti a valutarne la qualità prima di fare un'offerta.

Ogni utente ha accesso a un'area personale per consultare lo **storico delle proprie attività**. Gli acquirenti possono vedere le offerte fatte, le aste vinte, i feedback lasciati e la propria lista dei desideri. I venditori, invece, possono

gestire le aste pubblicate, verificare l'andamento delle vendite e analizzare le recensioni ricevute.

Il sistema di navigazione e ricerca è progettato per essere semplice ma avanzato. Gli utenti possono cercare le aste attraverso filtri per categoria, parola chiave, prezzo e durata residua. Inoltre, è possibile **ordinare i risultati** secondo diversi criteri:

- Prezzo (crescente o decrescente)
- Tempo rimanente (aste in scadenza prima o dopo)
- Reputazione del venditore (numero e media dei feedback ricevuti)
- Valutazione dei prodotti

## FACOLTATIVO

I prodotti aggiunti alla **lista dei desideri** vengono visualizzati nel profilo dell'acquirente sotto forma di una griglia. Ogni riga della griglia mostra a sinistra il prodotto salvato, mentre a destra vengono visualizzati **tre prodotti consigliati** in base alle preferenze e ai comportamenti di acquisto degli altri utenti.

Il sistema sfrutta una logica del tipo:

| "Chi ha acquistato questo prodotto, ha anche acquistato: ..."

Questo meccanismo di raccomandazione rende la sezione "preferiti" non solo una raccolta di oggetti salvati, ma anche uno strumento attivo per scoprire nuovi prodotti potenzialmente interessanti.

## 2 Analisi e Progettazione

In questa sezione mostrerò come ho pianificato l'applicazione prima di scrivere il codice, usando i diagrammi UML.

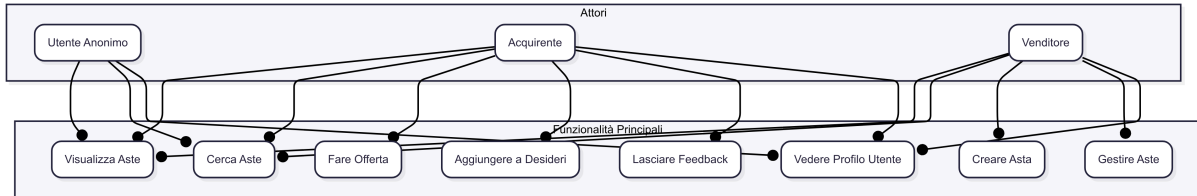
### 2.1 Diagramma dei Use Case (Casi d'Uso)

Per rappresentare le funzionalità del sistema e le interazioni con gli utenti, è stato realizzato un diagramma dei casi d'uso. Sono stati identificati tre attori principali:

- **Utente Anonimo:** Un visitatore non registrato.
- **Acquirente:** Un utente registrato con il ruolo di acquirente.

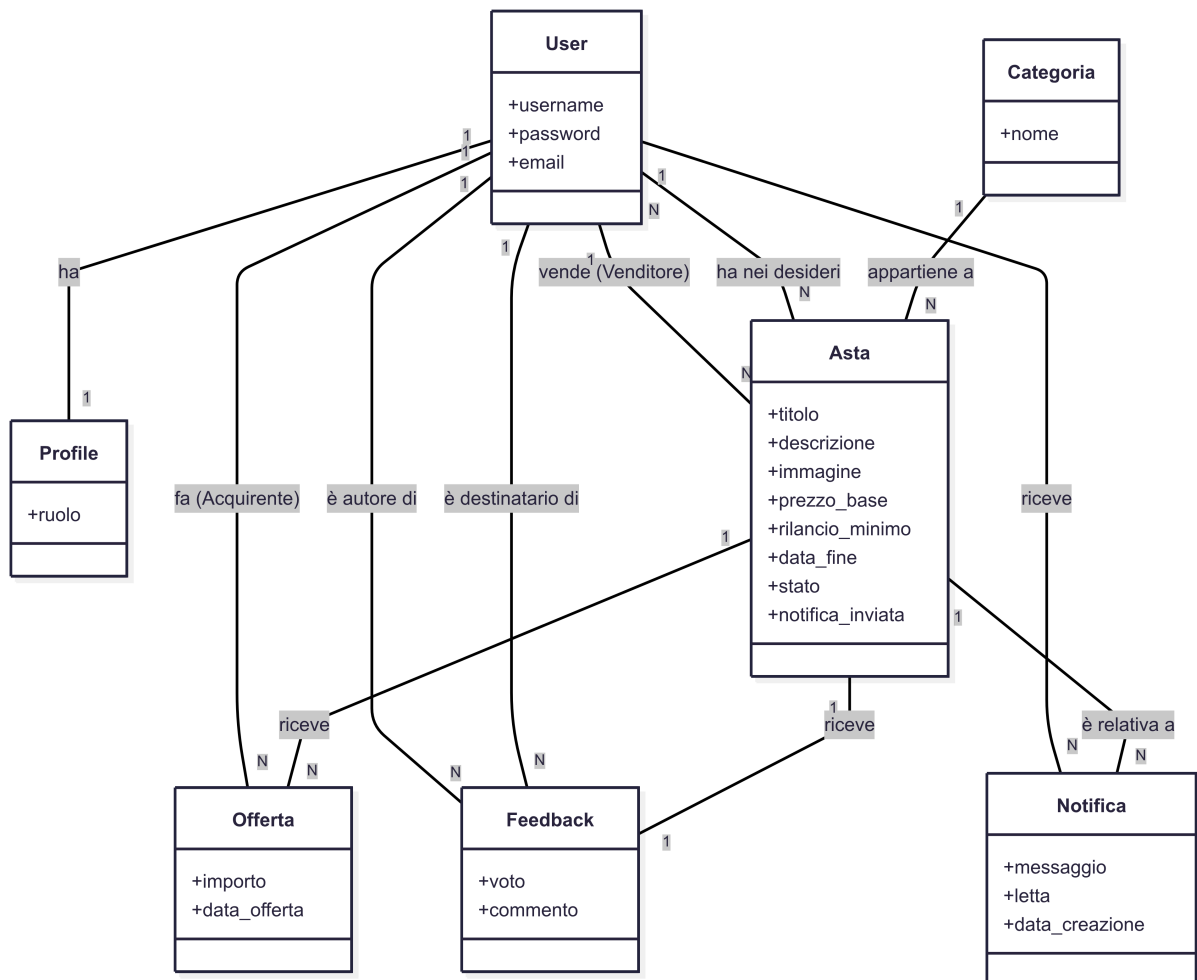
- **Venditore:** Un utente registrato con il ruolo di venditore.

Il diagramma seguente illustra le azioni che ogni attore può compiere all'interno della piattaforma.



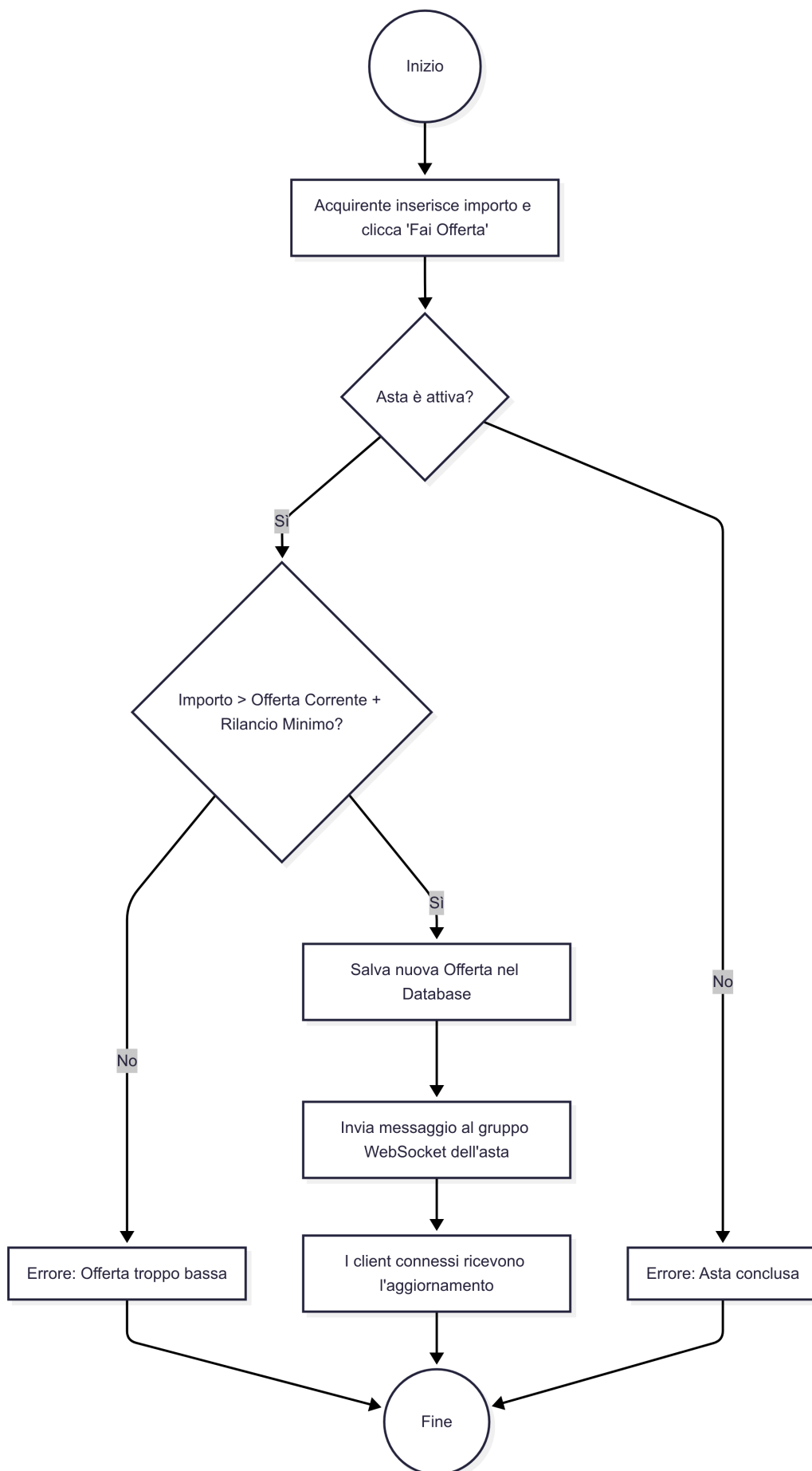
## 2.2 Diagramma delle Classi del Database

La struttura del database è stata definita tramite l'ORM (Object-Relational Mapper) di Django. Il diagramma delle classi seguente rappresenta i modelli principali dell'applicazione e le loro relazioni. Le classi definite sono: **User** (fornito da Django), **Profile**, **Categoria**, **Asta**, **Offerta**, **Feedback** e **Notifica**.



## 2.3 Diagramma di Attività: "Fare un'offerta"

Per illustrare un processo logico complesso, è stato scelto il flusso di "fare un'offerta", che include interazione utente, validazione lato server e comunicazione in tempo reale. Il seguente diagramma di attività ne descrive i passaggi.



## 3 Tecnologie e Architettura

In questo capitolo descriverò gli strumenti che ho usato e come ho organizzato il codice del progetto.

### 3.1 Tecnologie Utilizzate

Per la realizzazione della piattaforma sono state utilizzate le seguenti tecnologie:

- **Python:** Il linguaggio di programmazione principale su cui si basa l'intero backend dell'applicazione.
- **Django:** Framework web di alto livello per Python, utilizzato per la sua architettura Model-View-Template (MVT) che ha permesso uno sviluppo rapido e strutturato. Ha gestito la logica del server, l'interazione con il database e il routing degli URL.
- **Django Channels:** Estensione di Django che ha abilitato il supporto per il protocollo WebSocket, fondamentale per implementare la funzionalità di aggiornamento in tempo reale delle offerte senza che l'utente dovesse ricaricare la pagina.
- **SQLite:** Il database di default fornito da Django, utilizzato per la sua semplicità e per la gestione di tutti i dati dell'applicazione (utenti, aste, offerte, ecc.).
- **HTML, CSS, JavaScript:** Le tecnologie standard per la costruzione dell'interfaccia utente lato client.
- **Bootstrap:** Framework CSS utilizzato per la creazione di un layout responsive e di componenti UI stilisticamente coerenti (navbar, card, form, ecc.) in modo efficiente.
- **noUiSlider:** Libreria JavaScript esterna impiegata per la realizzazione dello slider a doppio cursore per la selezione della fascia di prezzo nella pagina di ricerca avanzata.
- **Pipenv:** Strumento utilizzato per la gestione delle dipendenze del progetto e per la creazione di un ambiente virtuale isolato, garantendo la riproducibilità dell'ambiente di sviluppo.

### 3.2 Architettura e Organizzazione del Codice

L'architettura del progetto segue il pattern **Model-View-Template (MVT)**, la variante del classico MVC adottata da Django:

- **Model:** Definiti in `aste/models.py`, rappresentano la struttura dei dati e le relazioni tra di essi (Asta, Offerta, User, etc.). Sono il tramite tra l'applicazione e il database SQLite.
- **View:** Contenute in `aste/views.py`, gestiscono la logica di business. Ricevono le richieste HTTP degli utenti, interagiscono con i modelli per recuperare o salvare dati e scelgono quale template utilizzare per la risposta. Nel progetto è stato adottato un approccio ibrido, utilizzando sia Class-Based Views (CBV) per le operazioni CRUD standard (es. `ListView`, `CreateView`) sia Function-Based Views (FBV) per logiche più specifiche (es. `fai_offerta`).
- **Template:** Situati nelle cartelle `templates/`, sono i file HTML che definiscono la struttura della presentazione. Contengono speciali tag (DTL - Django Template Language) che vengono riempiti dinamicamente dalle viste con i dati richiesti.

A livello di file, il progetto è stato suddiviso in due componenti principali, secondo le convenzioni di Django:

1. `aste_project`: La cartella di configurazione del progetto, che contiene file globali come `settings.py` (per le impostazioni), `urls.py` (per il routing principale) e `asgi.py` (per la configurazione di Django Channels).
2. `aste`: L'applicazione Django che racchiude tutta la logica specifica della piattaforma d'aste. Questa modularità permette di mantenere il codice organizzato e potenzialmente riutilizzabile. I file chiave al suo interno sono `models.py`, `views.py`, `forms.py`, `urls.py` e `consumers.py` (per la logica WebSocket).

## 4 Scelte Implementative Rilevanti

Durante lo sviluppo della piattaforma, sono state prese diverse decisioni architetturali e implementative per soddisfare i requisiti della traccia in modo robusto, efficiente e manutenibile. Questo capitolo illustra le scelte più significative relative alla gestione dei permessi, alle notifiche in tempo reale e all'interattività dell'interfaccia utente.

### 4.1 Gestione dei Permessi con Profile e Mixin



Per gestire la distinzione tra i ruoli di "Acquirente" e "Venditore", si è scelto di estendere il modello `User` standard di Django attraverso un modello `Profile` collegato con una relazione uno-a-uno. Questo approccio, rispetto all'uso dei gruppi di Django, permette una maggiore flessibilità e una più semplice integrazione della logica dei ruoli direttamente nel codice.

Il modello `Profile` è definito come segue:

```
# Questa classe estende le informazioni dell'utente standard di Django.
class Profile(models.Model):
    # Relazione 1:1 con il modello User,
    # Se un User viene cancellato allora anche il suo profilo associato verrà car
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    # Scelte possibili per il ruolo
    RUOLI = (
        ('acquirente', 'Acquirente'),
        ('venditore', 'Venditore'),
    )

    ruolo = models.CharField(max_length=10, choices=RUOLI, default='acquire
    # Definisce come un oggetto verrà rappresentato
    def __str__(self):

        return f"Profilo di {self.user.username} - Ruolo: {self.get_ruolo_display()}'
```

Sulla base di questo modello, sono stati creati dei **Mixin** personalizzati per proteggere le viste.

I Mixin sono classi riutilizzabili che aggiungono funzionalità specifiche alle Class-Based Views (CBV). Ad esempio, per garantire che solo i venditori possano creare o modificare le aste, sono stati implementati *VenditoreRequiredMixin* e *VenditoreAstaOwnerMixin*.

```
class VenditoreAstaOwnerMixin(UserPassesTestMixin):
    """
    Questo Mixin verifica che l'utente loggato sia il venditore dell'asta
    e che l'asta non abbia ancora ricevuto offerte.
    """
```

```

def test_func(self):
    asta = self.get_object()
    return self.request.user == asta.venditore and asta.offerte.count() == 0

def handle_no_permission(self):
    # Se test_func ritorna False, l'utente viene reindirizzato con un messaggio
    messages.error(self.request, "Azione non consentita: puoi modificare solo le tue aste")
    return redirect('aste:home')

class VenditoreRequiredMixin(AccessMixin):
    """
    Questo Mixin personalizzato verifica che l'utente sia loggato
    e che il suo profilo abbia il ruolo 'venditore'.
    """
    def dispatch(self, request, *args, **kwargs):
        if not request.user.is_authenticated:
            return self.handle_no_permission()
        try:
            if request.user.profile.ruolo != 'venditore':
                # Se l'utente è loggato ma non è un venditore, solleviamo un'eccezione
                # che può essere gestita mostrando un errore 403 (Forbidden).
                from django.core.exceptions import PermissionDenied
                raise PermissionDenied("Non hai i permessi per accedere a questa pagina")
            except Profile.DoesNotExist:
                # Se l'utente non ha un profilo (caso anomalo), neghiamo l'accesso.
                return self.handle_no_permission()

        return super().dispatch(request, *args, **kwargs)

```

Questo approccio permette di dichiarare i permessi in modo pulito ed esplicito direttamente nella definizione della vista (es. `class ModificaAstaView(VenditoreAstaOwnerMixin, UpdateView):`), seguendo il principio **DRY (Don't Repeat Yourself)**.

## 4.2 Notifiche in Tempo Reale con WebSockets

Per soddisfare il requisito di notificare gli utenti in tempo reale quando vengono superati in un'asta, si è scartata l'opzione di un polling AJAX periodico, in quanto inefficiente e non realmente istantaneo. La scelta è ricaduta sull'utilizzo

del protocollo **WebSocket**, implementato in Django attraverso la libreria **Channels**.

Il flusso implementato è il seguente:

1. Un utente invia un'offerta tramite una richiesta AJAX alla vista `fai_offerta`.
2. Dopo aver validato e salvato l'offerta nel database, la vista invia un messaggio al channel layer, in un gruppo specifico per quell'asta.

```
@login_required
def fai_offerta(request, pk):
    # ...
    channel_layer = get_channel_layer()
    async_to_sync(channel_layer.group_send)(
        f'asta_{asta.id}',
        {
            'type': 'offerta_aggiornata',
            'dati_offerta': {
                'nuovo_prezzo': f"{nuova_offerta.importo:.2f}",
                'acquirente': nuova_offerta.acquirente.username,
            }
        }
    )
    return JsonResponse({
        'success': True,
        'nuovo_prezzo': f"{nuova_offerta.importo:.2f}",
        'acquirente': nuova_offerta.acquirente.username
    })

# ...
```

3. L' `AstaConsumer`, in ascolto su quel gruppo, riceve il messaggio e lo trasmette a tutti i browser connessi a quella pagina d'asta tramite WebSocket.
4. Il JavaScript lato client riceve i dati e aggiorna dinamicamente il prezzo e il nome dell'ultimo offerente nell'interfaccia, senza che l'utente debba ricaricare la pagina.

Questa architettura garantisce un'esperienza utente fluida e reattiva, ed è significativamente più performante per applicazioni ad alta interazione come un

sito d'aste.

## 4.3 Interfaccia Utente Dinamica con JavaScript: la Lista dei Desideri

Per migliorare l'esperienza utente e renderla più fluida e reattiva, diverse funzionalità sono state rese dinamiche tramite JavaScript. Un esempio chiave è la **Lista dei Desideri**, che permette agli utenti di salvare le aste di loro interesse con un solo click, senza interrompere la navigazione.

Cliccando sull'icona a forma di cuore, viene attivata una funzione JavaScript che esegue una richiesta `fetch` asincrona a una vista Django dedicata (`gestisci_lista_desideri`). Questa vista si occupa di aggiornare il database e restituisce una semplice risposta in formato JSON per confermare l'esito dell'operazione.

```
// Selettore per TUTTI i pulsanti "desideri", sia nella lista che nel dettaglio.
const wishListButtons = document.querySelectorAll('.desideri-btn');

wishListButtons.forEach(button => {
  button.addEventListener('click', function(event) {
    const astald = this.dataset.astald;
    const url = `/asta/${astald}/desideri/`; // L'URL della nostra API view

    fetch(url, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
        'X-CSRFToken': csrftoken // Usiamo il token preso dal cookie
      },
    },
  ))
  .then(response => response.json())
  .then(data => {
    if (data.success) {
      // Aggiorniamo il pulsante cliccato
      if (data.added) {
        this.innerHTML = '♥';
        this.classList.remove('btn-outline-danger');
        this.classList.add('btn-danger');
      } else {
```

```

        this.innerHTML = '♥';
        this.classList.remove('btn-danger');
        this.classList.add('btn-outline-danger');
    }
}
})
.catch(error => console.error('Errore durante la richiesta desiderata:', erro
});
});

```

Una volta ricevuta la conferma dal server, il codice JavaScript aggiorna l'icona del cuore direttamente nella pagina, fornendo un feedback visivo immediato all'utente. Questa tecnica rende l'interazione con la piattaforma più veloce e moderna, evitando i ricaricamenti completi della pagina per piccole azioni.

## 5 Test Eseguiti

Per garantire la stabilità e il corretto funzionamento dell'applicazione, è stata implementata una suite di test automatici utilizzando il framework di testing integrato in Django. I test sono stati scritti nel file `aste/tests.py` e sono stati progettati per coprire sia la logica di business dei modelli sia il comportamento delle viste.

### 5.1 Test di Logica del Modello ( `AstaModelTests` )

La classe `AstaModelTests` è stata creata per verificare la correttezza della logica implementata nel modello `Asta`. In particolare, è stato testato il metodo `aggiorna_stato_se_scaduta`, che è cruciale per la gestione del ciclo di vita di un'asta. I test si assicurano che:

1. Lo stato di un'asta venga correttamente modificato in 'conclusa' quando la sua data di fine è passata.
2. Lo stato di un'asta rimanga 'attiva' se la sua data di fine è ancora nel futuro.

Questo garantisce che la logica di business principale del modello sia affidabile e funzioni come previsto in diversi scenari.

```

class AstaModelTests(TestCase):

```

```

def setUp(self):
    """
    Il metodo setUp viene eseguito prima di ogni test.
    Lo usiamo per creare oggetti comuni che serviranno in più test.
    """

    # Creiamo un utente venditore e una categoria di test
    self.venditore = User.objects.create_user(username='venditore_test', password='12345678')
    self.profile = Profile.objects.create(user=self.venditore, ruolo='venditore') # Assumiamo il ruolo di venditore
    self.categoria = Categoria.objects.create(nome='Elettronica')

def test_aggiorna_stato_se_scaduta(self):
    """
    Verifica che lo stato di un'asta cambi correttamente in 'conclusa' quando
    Questo è il test della "funzione di codice applicativo".
    """

    # 1. Creiamo un'asta la cui data di fine è nel passato (ieri)
    asta_scaduta = Asta.objects.create(
        venditore=self.venditore,
        titolo="Asta Scaduta",
        descrizione="Test per asta scaduta",
        prezzo_base=10.00,
        rilancio_minimo=1.00,
        categoria=self.categoria,
        data_fine=timezone.now() - timedelta(days=1)
    )

    # Chiamiamo il metodo che vogliamo testare
    asta_scaduta.aggiorna_stato_se_scaduta()

    # Verifichiamo che lo stato sia stato aggiornato a 'conclusa'
    self.assertEqual(asta_scaduta.stato, 'conclusa')

def test_non_aggiorna_stato_se_non_scaduta(self):
    """
    Verifica che lo stato di un'asta attiva non cambi se non è ancora scaduta.
    """

    # 1. Creiamo un'asta la cui data di fine è nel futuro (domani)
    asta_attiva = Asta.objects.create(

```

```

        venditore=self.venditore,
        titolo="Asta Attiva",
        descrizione="Test per asta attiva",
        prezzo_base=10.00,
        rilancio_minimo=1.00,
        categoria=self.categoria,
        data_fine=timezone.now() + timedelta(days=1)
    )

    # Chiamiamo il metodo
    asta_attiva.aggiorna_stato_se_scaduta()

    # Verifichiamo che lo stato sia rimasto 'attiva'
    self.assertEqual(asta_attiva.stato, 'attiva')

```

## 5.2 Test della Vista ( **HomeViewTests** )

Per verificare il corretto funzionamento del rendering delle pagine, la classe **HomeViewTests** utilizza il **Test Client** di Django per simulare le richieste di un utente alla homepage dell'applicazione. Questi test verificano diversi aspetti della vista **HomeAsteView** :

1. **Status Code**: Si accerta che la pagina risponda con un codice di stato **200 OK**, indicando che è stata caricata con successo.
2. **Template Utilizzato**: Controlla che la vista stia utilizzando il template corretto ( **aste/home.html** ).
3. **Contenuto della Pagina**: Verifica che il contenuto HTML generato contenga i dati attesi, come il titolo di un'asta creata appositamente per il test, e che mostri il messaggio corretto quando non ci sono aste disponibili.

```

class HomeViewTests(TestCase):

    def setUp(self):
        # Oltre al venditore, creiamo anche un'asta per i test
        self.venditore = User.objects.create_user(username='venditore_test_2', p
        Profile.objects.create(user=self.venditore, ruolo='venditore')
        self.categoria = Categoria.objects.create(nome='Libri')
        # Creiamo un'immagine GIF 1×1 pixel in memoria. È il modo standard

```

```

# per fornire un file valido a un ImageField durante i test.
dummy_image = SimpleUploadedFile(
    name='test_image.gif',
    content=b'GIF89a\x01\x00\x01\x00\x80\x00\x00\x00\x00\xff\xff\
    content_type='image/gif'
)
self.asta = Asta.objects.create(
    venditore=self.venditore,
    titolo="Un Titolo di Prova",
    descrizione="Descrizione di prova",
    immagine=dummy_image, # ← MODIFICA: Assegniamo l'immagine fir
    prezzo_base=50.00,
    rilancio_minimo=5.00,
    categoria=self.categoria,
    data_fine=timezone.now() + timedelta(days=5)
)

def test_home_view_status_code(self):
    """
    Verifica che la homepage risponda con un codice di stato 200 (OK).
    Questo è il test della "vista".
    """
    # Usiamo il client di test per fare una richiesta GET alla homepage
    response = self.client.get(reverse('aste:home'))
    self.assertEqual(response.status_code, 200)

def test_home_view_uses_correct_template(self):
    """
    Verifica che la homepage usi il template corretto.
    """
    response = self.client.get(reverse('aste:home'))
    self.assertTemplateUsed(response, 'aste/home.html')
    self.assertTemplateUsed(response, 'aste/_asta_card.html')

def test_home_view_displays_asta(self):
    """
    Verifica che la homepage mostri il titolo dell'asta che abbiamo creato.
    """

```



```

response = self.client.get(reverse('aste:home'))
# Controlliamo che il titolo della nostra asta sia presente nel contenuto HTML
self.assertContains(response, "Un Titolo di Prova")

def test_home_view_no_aste_message(self):
    """
    Verifica che, se non ci sono aste, venga mostrato il messaggio corretto.
    """
    # Cancelliamo l'asta creata nel setUp per simulare un DB vuoto
    self.asta.delete()
    response = self.client.get(reverse('aste:home'))
    self.assertContains(response, "Al momento non ci sono aste attive.")

```

```

PS C:\Users\Riccardo\Desktop\techweb definitivo\asteRoy> python manage.py test aste
USING SETTINGS: None
CHANNELS ACTIVE? False
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.....
-----
Ran 6 tests in 4.923s

OK
Destroying test database for alias 'default'...
PS C:\Users\Riccardo\Desktop\techweb definitivo\asteRoy>

```

## 6 Risultati: Screenshot dell'Applicazione

### ANONIMO

Aste Online

Cerca un prodotto...

Cerca

Registrati Login

Logout effettuato con successo. Arrivederci!

### Aste attualmente attive

**DVD "Django Unchained"**

Nel Sud degli Stati Uniti due anni prima dello scoppio della Guerra Civile, lo schiavo ...

Scade il: 16 Jul 2025, 16:50

Prezzo attuale: **€50.00**

Vedi Dettagli

**DVD "Django"**

Sulla frontiera fra Stati Uniti e Messico si fronteggiano due gruppi armati irregolari: una setta ...

Scade il: 16 Jul 2025, 16:50

Prezzo attuale: **€80.00**

Vedi Dettagli

**SIRIO di LAZZA AUTOGRAFATO**

Il disco "Sirio" di Lazza mescola sonorità trap, hip-hop e reggaeton, firmando un equilibrio perfetto ...

Scade il: 16 Jul 2025, 16:50

Prezzo attuale: **€400.00**

Vedi Dettagli

Home

Aste Online

Cerca un prodotto...

Cerca

Registrati Login

## DVD "Django"

**Venditore:** [Riccardo](#)

**Categoria:** FILM

Sulla frontiera fra Stati Uniti e Messico si fronteggiano due gruppi armati irregolari: una setta razzista di uomini col cappuccio rosso; e i rivoluzionari messicani. Un giorno arriva Django, un giustiziere dal passato misterioso che cammina trascinandosi dietro una cassa da morto. Uno degli spaghetti western più violenti di sempre. È il personaggio che dà il nome al Django di Django Unchained.

---

**Prezzo base:** € 80.00

Nessuna offerta ancora ricevuta.

**Rilancio minimo:** € 2.00

**Scade tra:** 21h 59m 20s

[Accedi](#) per fare un'offerta.

### Feedback per questa Asta

Nessun feedback ancora ricevuto per questa asta.

© 2025 Piattaforma Aste.

Realizzato da: Riccardo Mattioli, Numero Matricola: 166300, [Università](#), [Facoltà di scienze informatiche](#), [Corso Tecnologie Web](#).

Powered by: [Django](#), [Python](#), [Bootstrap](#), [Channels](#), [noUiSlider](#), [5 star rating](#)

Se le va mi seguea



## Dettaglio Asta Anonimo

## Profilo di Riccardo

Questo venditore non ha ancora ricevuto feedback.

### Feedback ricevuti:

Nessun feedback disponibile.

© 2025 Piattaforma Aste.

Realizzato da: Riccardo Mattioli, Numero Matricola: 166300, [Università](#), [Facoltà di scienze informatiche](#), [Corso Tecnologie Web](#).

Powered by: [Django](#), [Python](#), [Bootstrap](#), [Channels](#), [noUiSlider](#), [5 star rating](#)

Se le va mi segua



Profilo venditore

## Ricerca Avanzata

Parola chiave:

Es. Smartphone, libro antico...

Categoria:

Tutte le categorie

Intervallo di Prezzo



Durata residua:

Qualsiasi durata

Ordina per:

Tempo rimanente

☐ Includi concluse

[Filtra / Ordina](#)

## Risultati Trovati (9)



### DVD "Django Unchained"

Nel Sud degli Stati Uniti due anni prima dello scoppio della Guerra Civile, lo schiavo ...

Scade il: 16 Jul 2025, 16:50

Prezzo attuale: **€50.00**

[Vedi Dettagli](#)



### DVD "Django"

Sulla frontiera fra Stati Uniti e Messico si fronteggiano due gruppi armati irregolari: una setta ...

Scade il: 16 Jul 2025, 16:50

Prezzo attuale: **€80.00**

[Vedi Dettagli](#)



### SIRIO di LAZZA AUTOGRAFATO

Il disco "Sirio" di Lazza mescola sonorità trap, hip-hop e reggaeton, firmando un equilibrio perfetto ...

Scade il: 16 Jul 2025, 16:50

Prezzo attuale: **€400.00**

[Vedi Dettagli](#)

Ricerca

## VENDITORE

## Accedi

Username\*

Riccardo

Password\*

.....

[Accedi](#)

Login

## Pannello di Riccardo

Login effettuato con successo! Bentornato, Riccardo.

- **Username:** Riccardo
- **Email:** ricky@gmail.com
- **Ruolo:** Venditore

### Le tue Aste Pubblicate

LOCURA di LAZZA AUTOGRAFATO - Stato: Attiva

J di LAZZA AUTOGRAFATO - Stato: Attiva

SIRIO di LAZZA AUTOGRAFATO - Stato: Attiva

DVD "Django" - Stato: Attiva

DVD "Django Unchained" - Stato: Attiva

### Pannello personale con conferma di login

## Crea una Nuova Asta

Titolo\*

Descrizione\*

Immagine\*


Nessun file selezionato

Categoria\*

Prezzo base\*

Rilancio minimo\*

Data fine\*

gg/mm/aaaa --:-- 

Crea Asta

## Modifica: LOCURA di LAZZA AUTOGRAFATO

Titolo\*

LOCURA di LAZZA AUTOGRAFATO

Descrizione\*

"Locura" è il titolo del nuovo album di Lanza, che in spagnolo significa "follia". L'album è stato presentato con un evento speciale intitolato "LOCURA OPERA N. 1", che ha visto la partecipazione dell'Orchestra Sinfonica di Milano. Alcune edizioni dell'album sono autografate e numerate, come il cofanetto "LOCURA JAM + OPERA" che include 4 LP in vinile e una card autografata. Il concept dell'album è ispirato all'opera grafica di Francisco Goya sulla tauromachia.

Immagine\*

Currently [aste\\_images/locura\\_opera.jpg](#)

Scegli file Nessun file selezionato

Categoria\* MUSICA

Prezzo base\*

900,00

Rilancio minimo\*

5,00

Data fine\*

16/07/2025 16:50

Conferma Modifica

Modifica Asta

## Acquirente

## Pannello di Alessia

- **Username:** Alessia
- **Email:** alessia@gmail.com
- **Ruolo:** Acquirente

### Le tue Aste Vinte

Non hai ancora vinto nessuna asta.

## Lista dei Desideri con Suggerimenti

### Lista dei Desideri



#### DVD "Django Unchained"

Nel Sud degli Stati Uniti due anni prima dello scoppio della Guerra Civile, lo schiavo ...

Scade il: 16 Jul 2025, 16:50

Prezzo attuale: €

[Vedi Dettagli](#)




### Altri utenti hanno acquistato anche:

Nessun suggerimento disponibile.

## Pannello Personale Acquirente

Aste Online

Ciao, Alessia!
Pannello Personale
Logout



## LOCURA di LAZZA AUTOGRAFATO

**Venditore:** Riccardo

**Categoria:** MUSICA

"Locura" è il titolo del nuovo album di Lazza, che in spagnolo significa "follia". L'album è stato presentato con un evento speciale intitolato "LOCURA OPERA N. 1", che ha visto la partecipazione dell'Orchestra Sinfonica di Milano. Alcune edizioni dell'album sono autografate e numerate, come il cofanetto "LOCURA JAM + OPERA" che include 4 LP in vinile e una card autografata. Il concept dell'album è ispirato all'opera grafica di Francisco Goya sulla tauromachia.

---

**Prezzo base:** € 900.00

Nessuna offerta ancora ricevuta.

**Rilancio minimo:** € 5.00

**Scade tra:** 21h 53m 4s

La tua offerta:

### Dettaglio Asta Acquirente

Aste Online

Ciao, Alessia!
Pannello Personale
Logout

### Le Tue Notifiche

Non hai nessuna notifica al momento.

### Notifiche Acquirente

## 7 Conclusioni

Il progetto ha permesso di realizzare una piattaforma di aste online completa, soddisfacendo tutti i requisiti funzionali della traccia d'esame. Attraverso l'integrazione di Django, Channels per i WebSockets e JavaScript per le interazioni client-side, è stata creata un'applicazione robusta, interattiva e moderna.

Lo sviluppo di questo progetto ha rappresentato un'importante opportunità per applicare concretamente i concetti teorici del corso, dalla progettazione del database con l'ORM di Django alla gestione di comunicazioni in tempo reale, fino all'implementazione di una suite di test automatici per garantire la qualità



del software. Il risultato è un'applicazione web pienamente funzionante che dimostra una solida comprensione delle tecnologie full-stack.