

# FLEA MARKET

Progetto Django realizzato per il corso di Tecnologie Web

Vittoria Cantarelli | Matr: 134303  
UNIMORE | Corso Tecnologie Web 2021/22



## Sommario

Introduzione .....	4
User Model.....	6
Utenti vendor .....	8
Utenti costumer.....	9
Articoli .....	10
Mettere in vendita un articolo.....	10
Eseguire una ricerca.....	11
Recommendation system .....	12
Salvare un articolo .....	13
Aggiungere un articolo al carrello .....	13
Ordine e carrello.....	14
Messaggi.....	16
Notifiche.....	17
Django Crispy Form.....	18
Database .....	18
Organizzazione logica .....	19
Frontend .....	20
Html e Bootstrap.....	20
Css .....	20
JavaScript.....	20
Test.....	21
Test codice applicativo – funzione create_user e create_superuser.....	21
Test client – Register view .....	22
Test client – view add_product.....	23
Risultati.....	23
Possibili sviluppi .....	24

## Introduzione

Si è realizzata una piattaforma per un mercatino online che permette, quindi, la vendita e l'acquisto di articoli (da collezione o usati) da remoto. In particolare, supporta **tre tipi di utente**:

- utenti anonimi: possono effettuare ricerche, visualizzare la scheda tecnica di ogni prodotto con la relativa descrizione e il prezzo, visualizzare le votazioni relative al venditore (ma non i commenti), la sua valutazione media e gli altri articoli messi in vendita dallo stesso;
- utenti registrati: possono effettuare il login, modificare i propri dati (nome, cognome, indirizzo, ...) ed effettuare il cambio password. Possono acquisire i seguenti ruoli (uno, nessuno o entrambi):
  - venditori: oltre alle funzionalità degli utenti anonimi, possono mettere in vendita articoli compilando una scheda tecnica. I campi necessari sono titolo, prezzo, quantità (1 di default), categoria (da scegliere all'interno di una lista). Altri campi compilabili ma non necessari sono descrizione e immagine (se non fornita verrà visualizzata un'immagine di default). I venditori possono inoltre visualizzare le statistiche sulle proprie vendite, la cronologia degli ordini in corso e conclusi, gli articoli in vendita e, infine, vedere le recensioni con voto, commento e autore, sia proprie sia degli altri venditori;
  - acquirenti: oltre alle funzionalità degli utenti anonimi, possono inserire articoli nel carrello e concludere l'ordine e vedere le recensioni con voto, commento e autore, riferite ai venditori. Hanno a disposizione una dashboard che mostra gli ordini conclusi e in corso, con il relativo stato di avanzamento, e tutte le recensioni che hanno rilasciato. Possono anche salvare gli articoli piaciuti tra i preferiti, i quali verranno mostrati nella dashboard. Nel momento in cui l'ordine risulta concluso, ovvero giunto a destinazione, possono lasciare una **recensione** al venditore di ogni articolo presente in tale ordine;

All'atto della registrazione è possibile scegliere se registrarsi come venditore e/o acquirente (almeno uno dei due ruoli è richiesto) e viene lasciata la possibilità di acquisire l'altro ruolo in un secondo momento, reinserendo username e password.

La **ricerca degli articoli** può essere fatta:

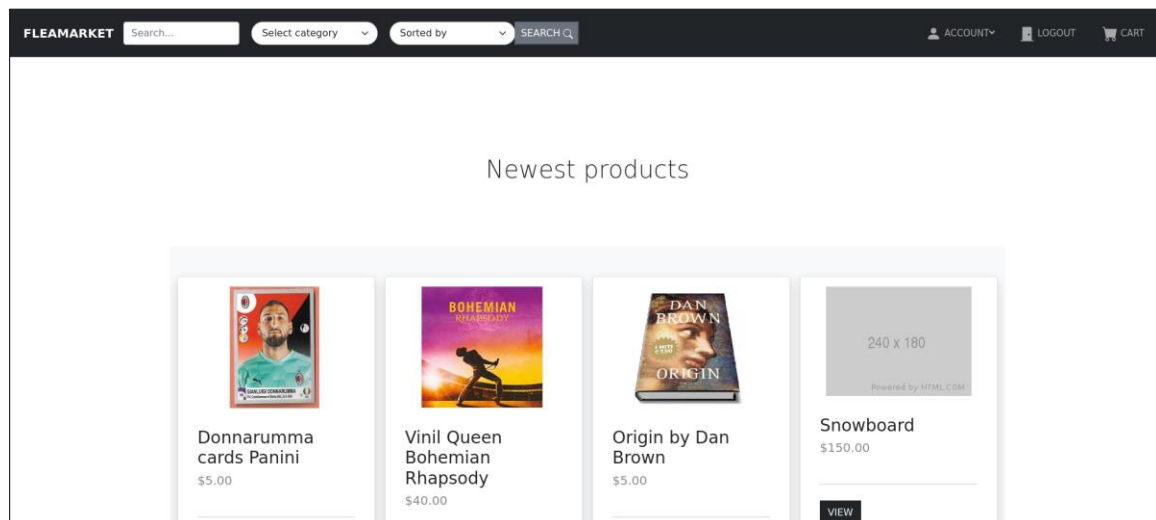
- per parola chiave: ricercata nel titolo e nella descrizione
- per categoria: vengono mostrati tutti gli articoli di una certa categoria

Per ogni tipo di ricerca, gli articoli verranno visualizzati di default in ordine di data (da quello inserito più recentemente, al più vecchio), oppure è possibile scegliere di visualizzarli in ordine di prezzo.

Nella pagina specifica di ogni prodotto, oltre alla scheda tecnica, vengono mostrati gli articoli simili per categoria e un insieme di articoli raccomandati. L'algoritmo di **recommendation system** adottato si basa sui salvataggi tra i preferiti effettuati dagli altri acquirenti.

Gli acquirenti possono inserire gli articoli nel **carrello**, eventualmente modificare la quantità o eliminarli, e procedere al pagamento (uno solo anche in caso di articoli appartenenti a venditori diversi). A questo punto, ogni venditore legato all'ordine dovrà confermare l'avvenuta spedizione e infine l'acquirente dovrà confermare la ricezione. **Ogni operazione viene notificata** all'acquirente o al venditore, a seconda di chi è l'interessato.

Il venditore, per ogni ordine concluso, accumula un **credito** che può decidere di riscattare in qualunque momento. Nella pagina di statistiche sulle vendite viene mostrato il credito disponibile, il credito già riscattato e il totale guadagnato.



*Homepage*

## User Model

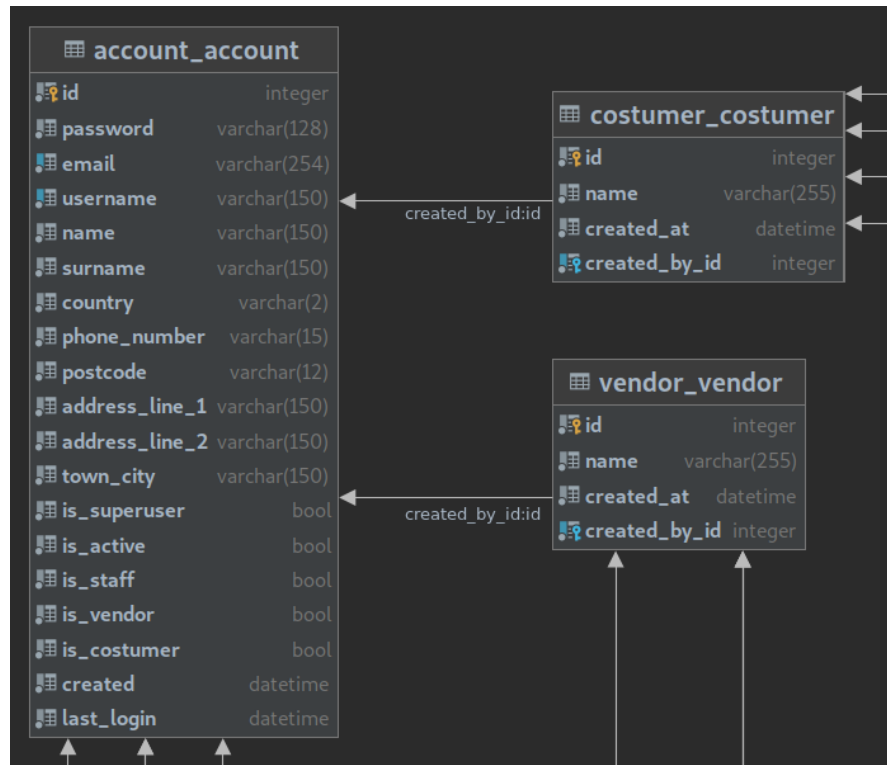


*Use case diagram delle principali funzionalità degli utenti*

Per far fronte alla necessità di avere diversi tipi di utenti, è stato creato un *costum user model*, nominato *Account*. In particolare, esso definisce un qualunque tipo di utente non anonimo, sia esso *vendor* e/o *costumer*, e include, tra gli altri campi, due valori booleani (*is\_vendor* e *is\_costumer*)

che indicano se si tratta di un venditore e/o di un cliente. Di questi due valori sono possibili tutte le combinazioni (entrambi falsi, entrambi veri, uno vero e l'altro falso); in questo modo, l'utente che desidera sia effettuare acquisti, sia mettere in vendita utilizzerà un unico username e un'unica password per l'accesso.

Ogni istanza di Account, quindi, può essere associata ad al più una istanza del modello Vendor e al più una istanza del modello Costumer. Sono questi che forniscono le funzionalità specificatamente legate al tipo di utente.



*Uml diagram del modello user*

All'atto di registrazione, quando viene richiesto il tipo di profilo che si vuole creare, vengono settati i valori di `is_vendor` e `is_costumer`, ed eventualmente create le istanze di Vendor e Costumer associate. Tuttavia, è anche possibile effettuare l'upgrade in un momento successivo.

Per limitare le azioni da parte degli utenti che non ne hanno i permessi, sono stati implementati i due decorator `@costumer_required` e `@vendor_required`, i quali, similmente a `@login_required`, quando falliscono il controllo eseguono una redirect alla pagina di upgrade. È opportuno utilizzarli in combinazione con `@login_required`, in modo tale che, se l'utente è anonimo, viene rinvio alla pagina di login, se l'utente è loggato ma non ha i requisiti per tale azione, viene rinvio alla pagina di upgrade, altrimenti può effettuare l'azione.

Diversamente, per personalizzare i template sono state utilizzate espressioni come

{% if request.user.is\_authenticated %} o {% if request.user.is\_costumer %}.

Per completezza, in fase di registrazione – o durante un upgrade successivo - vengono anche impostati i permessi legati al tipo di utente, anche se questi non vengono mai utilizzati successivamente, preferendo i decoratori già citati.

Infine, per il login, si è preferito utilizzare lo username alla e-mail, sia per la piattaforma di amministrazione, sia per l'applicazione vera e propria.

## UTENTI VENDOR

Gli utenti vendor hanno a disposizione una vendor admin da cui possono:

- Modificare il proprio nome che per impostazione predefinita coincide con lo username
- Visualizzare il proprio guadagno già riscattato, da riscattare ed eventualmente riscattarlo
- Aggiungere articoli in vendita
- Visualizzare gli articoli attualmente in vendita
- Visualizzare gli ordini ricevuti e quindi confermare l'avvenuta spedizione
- Visualizzare le recensioni ricevute

Vendor admin - ciccio99

My total gain: \$5.00  
I've already recovered: \$0  
My credit: \$5.00

RECOVER MY CREDIT

EDIT

LOG OUT

My products

ADD PRODUCT

Title	Price
<a href="#">Snowboard</a>	\$150.00

My orders

#3 - Feb. 13, 2022, 9:49 a.m.

Name: Marco Bianchi  
Address: via campi 213b  
Country, town, postcode: IT, Modena, 41124  
E-mail: marco.84@gmail.com  
Phone: 332478503

Title	Price	Quantity	Total	Paid	Shipped	Received	Total
Book of the little prince	\$5.00	1	\$5.00	No	Yes	Yes	\$5.00

Delivered successfully

My reviews

Average rate: 5.0/5.0

Rate	Comment	Written by	Date
5	I ordered him the book "Little Prince" and it has arrived within 3 days! Amazing!	marco840	Feb. 13, 2022, 10:49 a.m.

*Vendor admin*



## UTENTI COSTUMER

Gli utenti costumer hanno a disposizione una costumer admin da cui possono:

- Modificare il proprio nome che per impostazione predefinita coincide con lo username
- Visualizzare gli articoli salvati tra i preferiti ancora in vendita
- Visualizzare gli ordini effettuati e quindi confermarne la ricezione e rilasciare una recensione

FLEAMARKET

Search...

Select category

Sorted by

SEARCH Q

ACCOUNT

LOGOUT

CART

Costumer admin - marco840

EDIT

LOG OUT


NOTIFICATIONS (0)

PERSONAL DATA

ACTIVATE SELLER PROFILE


INFO ABOUT MY ORDERS

Saved products



Canon camera  
\$200.00

VIEW



Macbook  
\$500.00

VIEW

My orders

#3 - Feb. 13, 2022, 9:49 a.m.

Address: via campi 213b

Country, town, postcode: IT, Modena, 41124

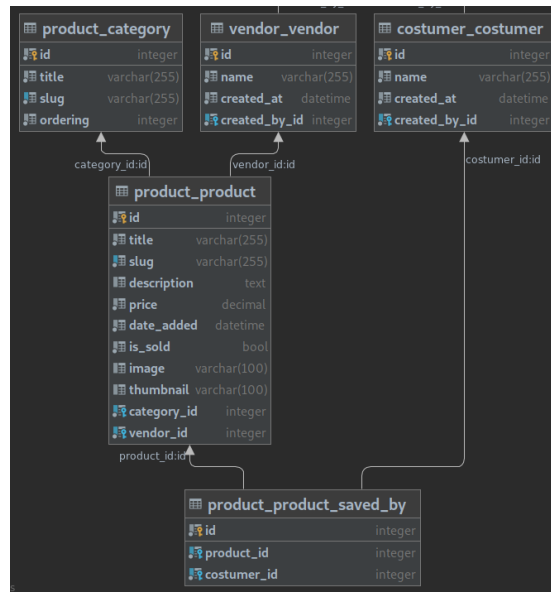
E-mail: marco.84@gmail.com

Phone: 332478503

Title	Price	Quantity	Total	Bought from	Paid	Shipped	Received
Book of the little prince	\$5.00	1	\$5.00	<a href="#">ciccio99</a>	Yes	Yes	Yes <a href="#">Leave a review</a>
Monopoly Star Wars	\$15.00	1	\$15.00	<a href="#">superman91</a>	Yes	Yes	No <a href="#">Confirm the successful delivery</a>

*Costumer admin*

## Articoli



*Uml diagram degli articoli*

### METTERE IN VENDITA UN ARTICOLO

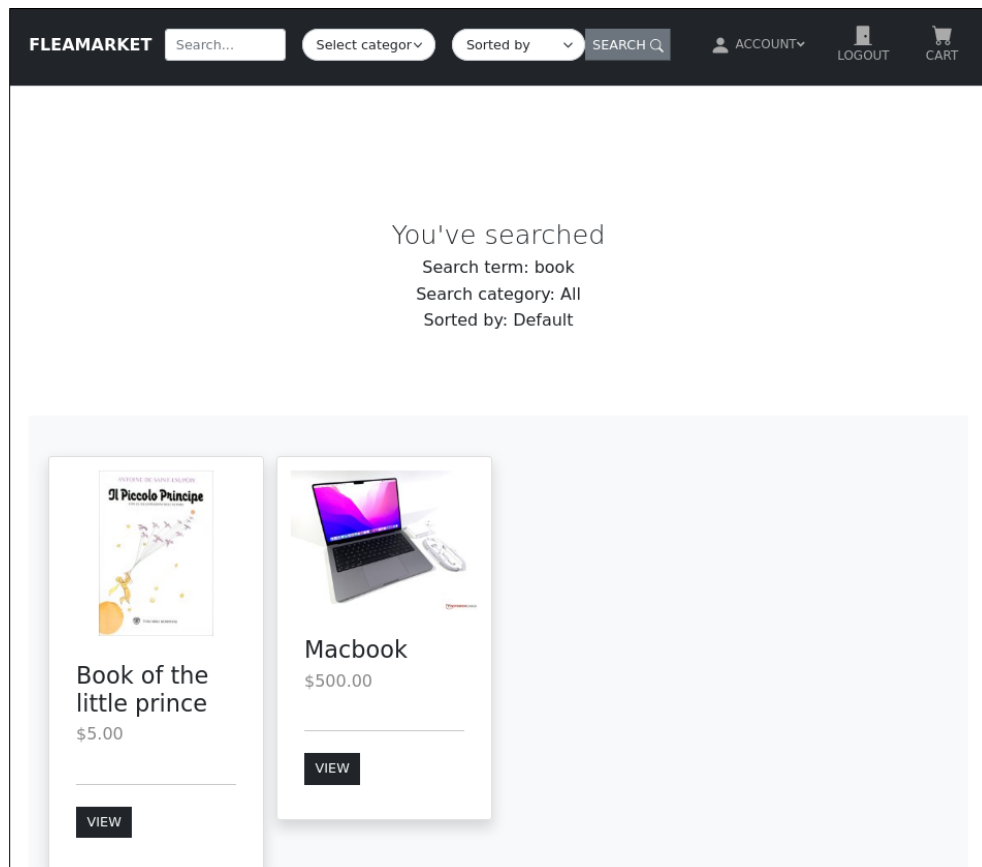
Solo gli utenti vendor hanno la facoltà di mettere in vendita un articolo. Per farlo, è sufficiente andare nella propria vendor admin e cliccare su “Add product”. A questo punto, l’utente sarà reindirizzato ad un’altra pagina in cui verrà lui richiesto di inserire i dati. In particolare, dovrà obbligatoriamente digitare un titolo, il prezzo e la quantità e scegliere una categoria tra quelle a disposizione, ovvero non può aggiungere categorie a piacere (possibile solo dalla pagina di amministrazione). Eventualmente può caricare un’immagine ed inserire una descrizione. Dalla propria vendor admin è quindi possibile vedere tutte le proprie inserzioni attualmente attive.

The screenshot shows the 'Add product' form. It includes a 'Category\*' dropdown menu, an 'Image' field with a file selection button and the text 'Nessun file selezionato.', a 'Title\*' text input, a 'Description' text area, a 'Quantity\*' spinner set to 1, a 'Price\*' spinner, and an 'ADD' button at the bottom.

*Template con form per l’aggiunta di un articolo*

## ESEGUIRE UNA RICERCA

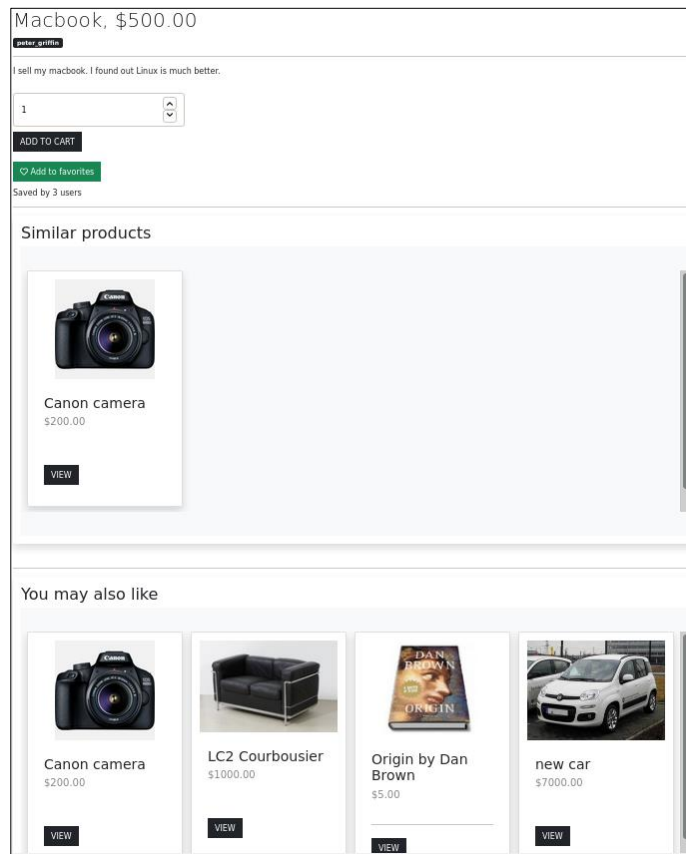
Tutti gli utenti possono eseguire una ricerca, utilizzando la barra di navigazione superiore presente in ogni pagina del sito. È possibile digitare una parola chiave, che verrà cercata all'interno del titolo e della descrizione e filtrare una categoria. I risultati possono essere ordinati dal più al meno recente, e viceversa, e dal più al meno costoso, e viceversa. Per ogni filtro esistono dei valori di default, per cui è anche possibile eseguire una ricerca senza inserire alcuna parola chiave, cercare in tutte le categorie e applicare l'ordine predefinito. Banalmente verranno mostrati tutti gli articoli disponibili, in ordine di inserimento (dal più nuovo).



*Esempio di ricerca*

## RECOMMENDATION SYSTEM

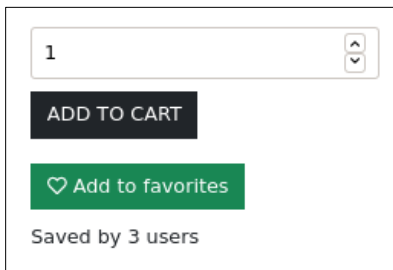
Ogni articolo dispone di una propria scheda tecnica in cui sono visualizzabili da qualunque utente titolo, immagine, descrizione, prezzo e venditore corrispondente. Scrollando la pagina verso il basso si notano due sezioni: “similar products” e “You may also like”. I prodotti simili sono articoli appartenenti alla stessa categoria, mentre per la seconda sezione è stato implementato un algoritmo di recommendation system, con l’intento di mostrare suggerimenti rilevanti e massimizzare le vendite. Negli e-commerce viene spesso adottato un metodo del tipo “Chi ha comprato questo ha comprato anche ...”, ma, data la natura dell’applicazione, ovvero con la maggior parte dei prodotti disponibili in quantità singola, risultava inapplicabile. È stato quindi utilizzato il metodo “Chi ha salvato questo ha salvato anche ...” nell’ipotesi che se all’utente A piacciono sia l’articolo X sia l’articolo Y, allora l’utente B, che sta visualizzando l’articolo X, potrebbe apprezzare anche l’articolo Y. Questo sistema potrebbe anche mostrare articoli con caratteristiche completamente diverse, non paragonabili, ed è per questo che è abbinato al primo metodo mostrante le inserzioni simili. Il difetto dell’approccio adottato per il sistema di raccomandazione, ovvero di tipo collaborativo, è il cold start. Infatti, quando un articolo è appena creato, non sarà ancora salvato da nessuno e quindi non ritornato tra i suggerimenti.



*Esempio di risultato dell'algoritmo di recommendation system*

## SALVARE UN ARTICOLO

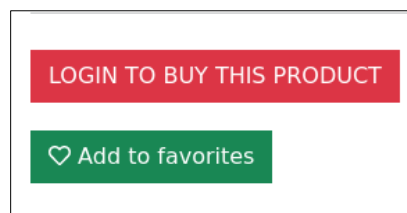
Sempre dalla scheda tecnica di un articolo, è possibile scegliere di salvarlo tra i preferiti. Questa azione è fruibile solo da profili di tipo costumer. Al contrario, se l'articolo è già salvato, allora si può decidere di eliminarlo. Viene inoltre mostrato il numero di utenti che hanno già salvato quell'articolo. Nella costumer admin sono consultabili tutti gli articoli salvati tra i preferiti e attualmente in vendita.



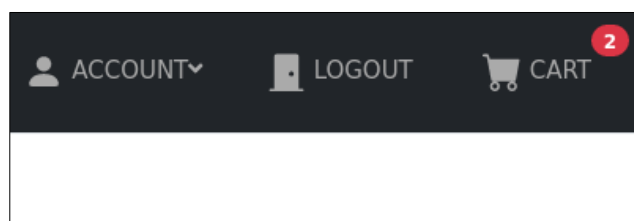
*Buttons per aggiungere un articolo al carrello e per aggiungere/eliminare un articolo ai preferiti*

## AGGIUNGERE UN ARTICOLO AL CARRELLO

Dalla pagina di ogni singolo articolo è possibile scegliere di aggiungere quell'articolo al carrello, in quantità da 1 a n, dove n è il numero di pezzi disponibili per la vendita. Naturalmente è necessario essere utenti di tipo costumer, altrimenti si verrà reindirizzati alla pagina di login o alla pagina di upgrade a costumer.

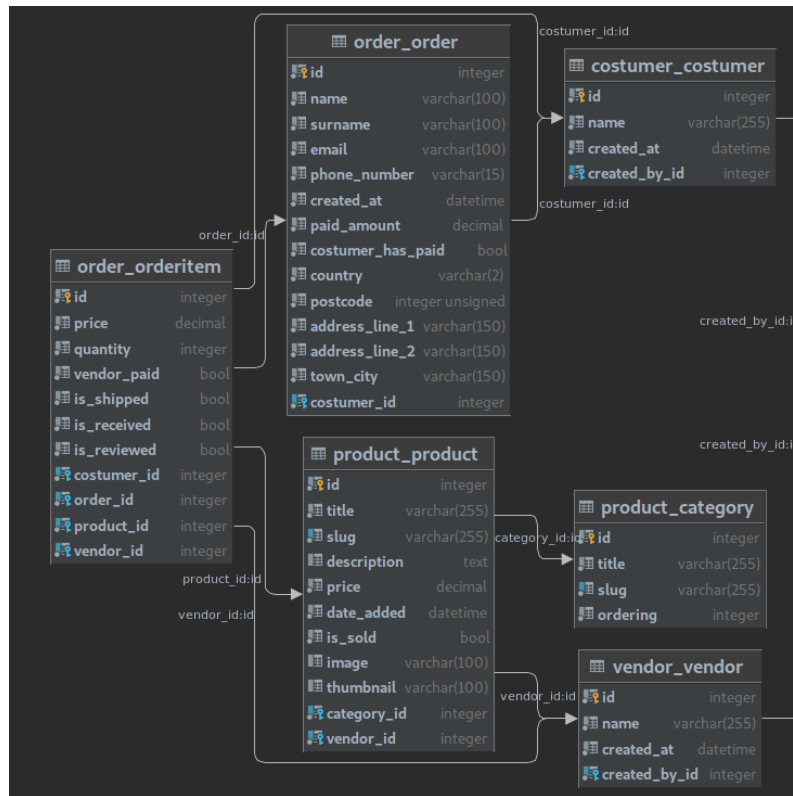


*Il button per comprare un articolo dipende dal tipo di utente*



*Viene tenuta traccia del numero di articoli nel carrello*

## Ordine e carrello



*Uml diagram delle entità relative all'ordine*



Nel carrello, visualizzabile solo dagli utenti customer, sono riportati tutti gli articoli inseriti, con la possibilità di modificare la quantità, nei limiti di disponibilità, ed eliminarli. Per procedere all'ordine è necessario compilare la form sottostante, inserendo i dati personali, i dati di spedizione e i dati di pagamento. I primi due, se inseriti dall'utente in fase di registrazione o in un momento successivo, saranno precompilati, con possibilità di modificarli. Ammesso che la form venga compilata correttamente, l'utente sarà reindirizzato alla pagina di successo.

Il fatto che un ordine possa contenere più articoli eventualmente di venditori diversi ha indotto a creare, a livello di database, due istanze: Order e OrderItem. Order rappresenta l'ordine come visualizzato dall'acquirente, mentre OrderItem identifica ogni singolo articolo dell'ordine, associato ad un preciso venditore e con un proprio stato di spedizione, indipendente dagli altri articoli dello stesso ordine.

Nel momento in cui l'ordine viene confermato, la quantità di ciascun articolo presente in esso viene decrementata e, se scende a 0, tale articolo viene marcato come venduto (is\_sold = True nel database). Nonostante gli articoli messi in vendita su questo tipo di piattaforma siano spesso in quantità singola, si è comunque voluta gestire la possibilità di quantità multipla. In questo caso,

l'acquirente può decidere la quantità da comprare del corrispondente articolo, il quale sarà identificato come venduto solo quando questa viene esaurita.

I passaggi che seguono riguardano la conferma di spedizione e la conferma di ricezione. Infine, l'acquirente può decidere se lasciare una recensione. Tutti questi step si fanno dalla vendor admin e dalla customer admin e sono accompagnati da una notifica (si veda il cap. Messaggi).

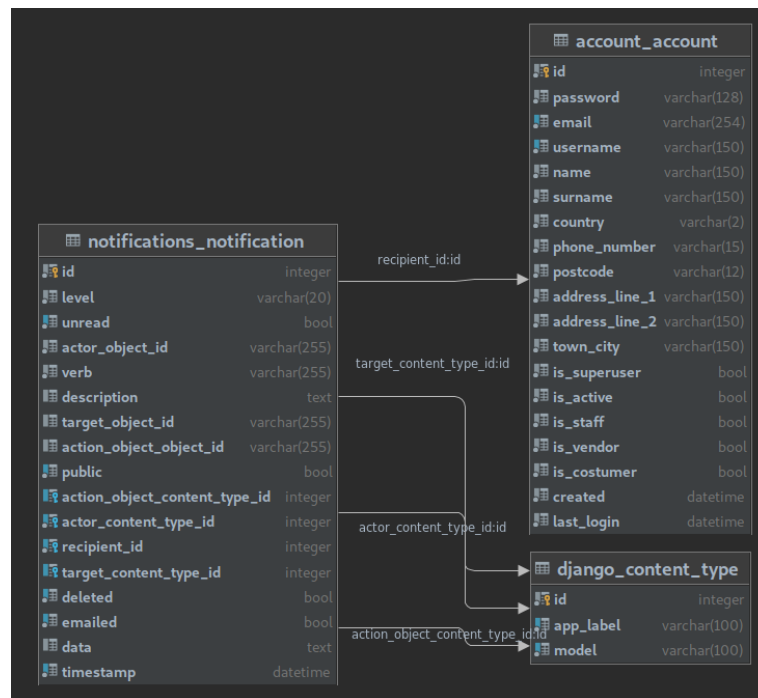
Cart			
Product	Quantity	Price	
 <a href="#">Donnarumma cards Panini</a>	1 <span>-</span> <span>+</span>	\$5.00	<a href="#">X remove</a>
 <a href="#">Vinil Queen Bohemian Rhapsody</a>	3 <span>-</span> <span>+</span>	\$120.00	<a href="#">X remove</a>
<b>Total cost</b>	<b>4</b>	<b>\$125.00</b>	

*Esempio di carrello con possibilità di modificare gli articoli "al volo"*

Contact and payment information	
Name*	<input type="text" value="francesco"/>
Surname*	<input type="text" value="ciccioli"/>
Email*	<input type="text" value="ciccio99@abc.com"/>
Phone number*	<input type="text" value="39272917940"/>
Country*	<div>Italy ▾</div>
Town city*	<input type="text" value="Milan"/>
Postcode*	<input type="text" value="41124"/>
Address line 1*	<input type="text" value="via campi 213b"/>
Address line 2	<input type="text"/>
Credit card*	<input type="text"/>
Cvv*	<input type="text"/>
CONFIRM AND PAY	

*Form precompilata per procedere all'acquisto*

## Messaggi



Uml diagram per i messaggi

Si è voluto includere nell'applicazione un sistema che permettesse la ricezione di messaggi – così chiamati per distinguerli dalle notifiche popup spiegate nel capitolo successivo - tra acquirente e venditore di un articolo. A tal fine, si è sfruttata l'app django-notifications. Una volta installata, inclusa tra le `INSTALLED_APPS` in `settings.py` e creato l'url di cui necessita<sup>1</sup>, il suo funzionamento è piuttosto elementare: è sufficiente utilizzare l'istruzione `notify.send(actor, recipient, verb, action_object, target, level, description, public, timestamp, **kwargs)` per inviare un messaggio. Siccome si è preferito limitare questa funzionalità a un insieme di messaggi preimpostati inviati automaticamente in corrispondenza dei principali step di un ordine, ovvero:

- Ricezione di un nuovo ordine, da notificare al venditore
- Avvenuta spedizione, da notificare all'acquirente
- Conferma ricezione dell'articolo, da notificare al venditore
- Ricezione di una recensione, da notificare al venditore

e inoltre:

---

<sup>1</sup> Per maggiori info si consulti la documentazione ufficiale al link <https://pypi.org/project/django-notifications-hq/>



- Salvataggio di un proprio articolo in vendita tra i preferiti da parte di un acquirente, da notificare al venditore
- Delezione di un proprio articolo in vendita tra i preferiti da parte di un acquirente, da notificare al venditore

è stata scritta una funzione intermedia che definisce i possibili messaggi in un dizionario<sup>2</sup>.

A questo punto, ciascun utente registrato potrà visualizzare i messaggi ricevuti - cliccando sull'apposito Button nella barra di navigazione superiore - visualizzati in ordine cronologico, ed eventualmente segnarli come "letti".

MARK ALL NOTIFICATIONS AS READ				
from	object	Message	When	Read
marco840	Snowboard	marco840 removed Snowboard from favorites	Feb. 23, 2022, 9:38 a.m.	✓ Unread
marco840	Snowboard	marco840 added Snowboard to favorites	Feb. 23, 2022, 9:38 a.m.	✓ Unread
ciccio99	Snowboard	ciccio99 added Snowboard to favorites	Feb. 13, 2022, 8:21 p.m.	✓ Unread
ciccio99	Snowboard	ciccio99 removed Snowboard from favorites	Feb. 13, 2022, 8:19 p.m.	✓ Unread
ciccio99	Snowboard	ciccio99 added Snowboard to favorites	Feb. 13, 2022, 8:15 p.m.	✓ Unread
johnlen	Snowboard	johnlen added Snowboard to favorites	Feb. 13, 2022, 8:04 p.m.	✓ Unread
marco84	None	marco84 left you a review	Feb. 13, 2022, 10:43 a.m.	✓ Unread
marco840	Book of the little prince	you have a new order of Book of the little prince	Feb. 13, 2022, 9:49 a.m.	✓ Read

*Messaggi ricevuti*

## Notifiche

Per migliorare l'esperienza utente, l'applicazione è stata dotata di un sistema di notifiche, mostrate sotto forma di elementi pop-up. Per questo scopo, si è utilizzato il supporto presente di default in Django e fornito dall'app `django.contrib.messages`<sup>3</sup>. Il meccanismo di storage backend scelto è quello di default, ovvero basato sulle sessioni.

Vengono mostrate notifiche per la maggior parte delle operazioni effettuate, dalla registrazione, al cambio password, all'aggiunta di un prodotto al carrello, fino alla conclusione dell'ordine. Le notifiche possono essere suddivise in due sottocategorie: operazione avvenuta con successo ed errore.

The product has been removed from favorites successfully ✕	The product has been added to favorites successfully ✕
--	--

*Esempio di notifica negativa*

*Esempio di notifica positiva*

<sup>2</sup> `send_notifications(sender, recipient, action, product=None)` in `/apps/order/utilities.py`

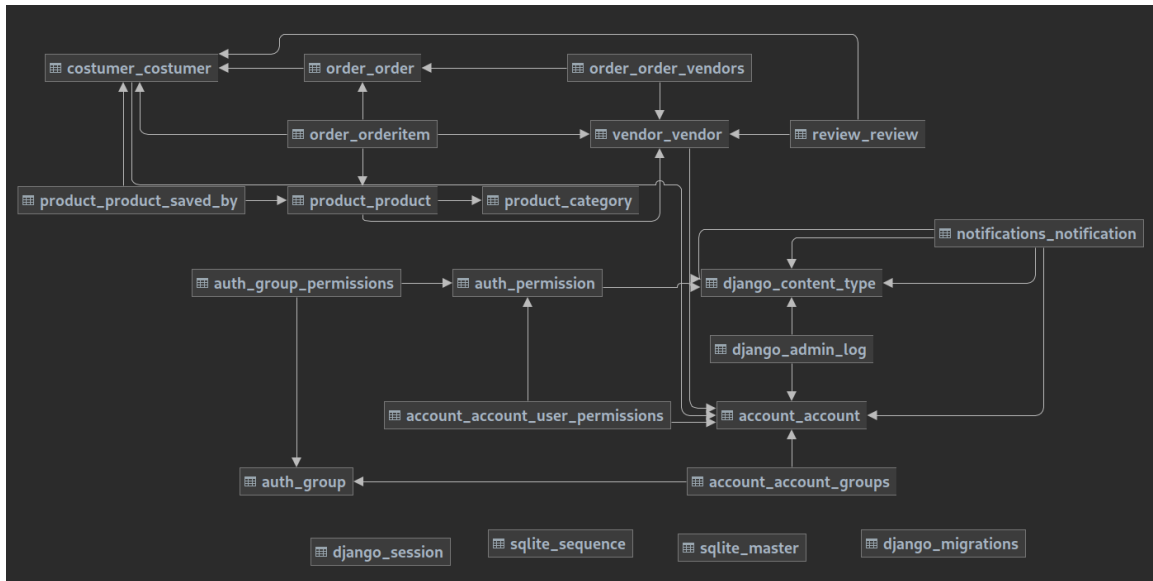
<sup>3</sup> Per maggiori info si consulti la documentazione ufficiale al link <https://docs.djangoproject.com/en/4.0/ref/contrib/messages/>

## Django Crispy Form

Per il rendering delle varie form utilizzate, si è fatto largo impiego delle crispy form, per la loro semplicità di utilizzo. Si è sfruttato sia il filtro `{% crispy %}` per le form create appositamente, sia il tag `|crispy` per le form già disponibili, come la `AuthenticationForm` o la `ChangePasswordForm`, nelle quali parte del lavoro è stato customizzato all'interno del template (ad esempio lo stile dei Submit button).

## Database

Si è scelto di utilizzare il database configurato di default in Django, ovvero `SQLite`, per semplicità di utilizzo e perché considerato sufficiente per soddisfare i requisiti progettuali.



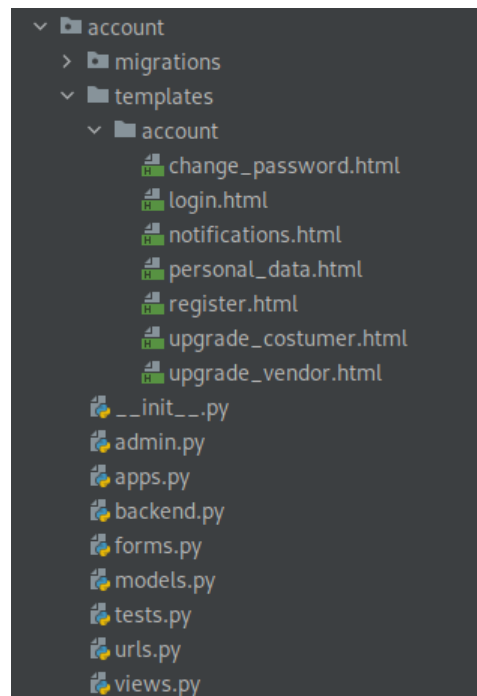
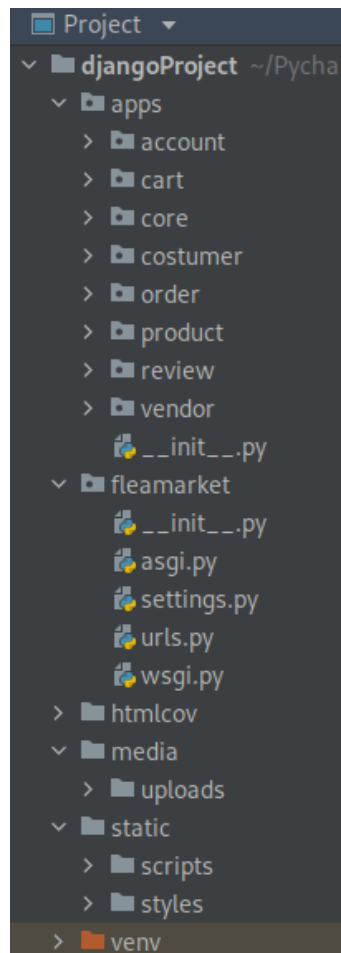
*Uml diagram riassuntivo di tutte le entità*

## Organizzazione logica

L'applicazione è stata organizzata in modo da preferire la semplicità delle singole apps, a discapito di un numero maggiore, nell'idea di rendere il progetto chiaro e modulabile. In particolare, la directory /fleamarket raggruppa i file di configurazione del progetto. Ogni sottodirectory di /apps contiene i file che implementano una funzionalità (e non necessariamente un modello).

I file HTML sono raggruppati all'interno della directory /templates dell'app a cui fanno riferimento.

Infine, la directory /media raccoglie tutte le immagini utilizzate per gli articoli del database e dentro /static sono contenuti gli aspetti che riguardano l'aspetto (css) e il comportamento (javascript) dell'HTML.



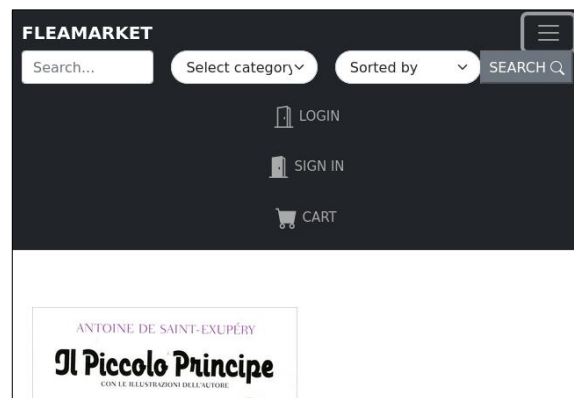
## Frontend

### HTML E BOOTSTRAP

Il frontend è stato realizzato utilizzando il framework Bootstrap 5. Per rendere l'applicazione più accattivante, ma soprattutto più intuitiva, sono state inserite delle icone prese in parte dalla libreria fornita direttamente da Bootstrap, e in parte dalla libreria Font Awesome.

Per questioni di compatibilità, per il rendering delle crispy forms è stato utilizzato il template pack Bootstrap 4 senza particolari problemi sul risultato finale.

L'intero sito è appositamente ottimizzato anche per la versione mobile, oltre che per quella desktop. Ad esempio, tutte le tabelle sono state rese responsive con scroll orizzontale. La stessa barra di navigazione superiore si adatta ai piccoli schermi.



*Vista mobile della barra superiore*

### CSS

L'aggiunta di styling tramite css non è stata quasi mai indispensabile, ma dettata solamente da ragioni estetiche: ad esempio, è stata utilizzata per definire margini, dimensioni, colore e posizione degli elementi principali, definire il carattere del testo e renderlo automaticamente maiuscolo nei button e nella navbar superiore. L'unico caso in cui si è resa necessaria è stato per far comparire le notifiche pop-up in primo piano, altrimenti coperte dagli altri elementi.

### JAVASCRIPT

L'uso di codice JavaScript è stato ridotto al minimo, cercando di sfruttare, laddove possibile, funzionalità pre-implementate in Bootstrap stesso. Tuttavia, è stata utilizzata una funzione JavaScript per impostare un timer alle le notifiche pop-up oltre il quale chiuderle automaticamente, oltre alla possibilità di chiuderle anticipatamente.

## Test

Per i test svolti si è utilizzato il modulo unittest fornito da Django, unito al tool coverage.py per tenere traccia dei frammenti di codice testati (e non) e per generare il report finale.

Sono stati realizzati test principalmente di tipo Functional e sfruttando la tecnica WhiteBox.

Come funzione di codice applicativo si è scelto di testare le funzioni `create_user` e `create_superuser` che sono state ridefinite avendo creato un modello di utente customizzato. Per il testing del client è stata testata la view di registrazione utente, che fa a sua volta uso della funzione `create_user`, la view che consente l'aggiunta di un prodotto in vendita che sfrutta i decoratori `@login_required` e `@vendor_required`.

### TEST CODICE APPLICATIVO – FUNZIONE CREATE\_USER E CREATE\_SUPERUSER

È stato testato il comportamento in caso di:

1. Creazione utente con parametri validi → funzionamento corretto, controllo che il settaggio dei campi associati all'utente sia quello aspettato
2. Creazione utente senza username → sollevamento del ValueError previsto
3. Creazione utente senza e-mail → sollevamento del ValueError previsto
4. Creazione utente senza password → generazione di una password casuale
5. Creazione superutente con parametri validi → funzionamento corretto, controllo che il settaggio dei campi associati all'utente sia quello aspettato
6. Creazione superutente con `is_vendor = False` → sollevamento del ValueError previsto
7. Creazione superutente con `is_costumer = False` → sollevamento del ValueError previsto
8. Creazione superutente con `is_superuser = False` → sollevamento del ValueError previsto

```
def test_create_user_valid(self):
    start_time = timezone.now()
    user = get_user_model().objects.create_user(
        username="JohnLennon", password="secretpassword", name="John",
        surname="Lennon", email="john.lennon@gmail.com"
    )
    end_time = timezone.now()
    self.assertEqual(user.email, 'john.lennon@gmail.com')
    self.assertEqual(user.username, "JohnLennon")
    self.assertEqual(user.name, "John")
    self.assertEqual(user.surname, "Lennon")
    self.assertFalse(user.is_superuser)
    self.assertFalse(user.is_vendor)
    self.assertFalse(user.is_costumer)
    self.assertFalse(user.is_active)
    self.assertFalse(user.is_staff)
    self.assertTrue(end_time > user.created > start_time)
    self.assertTrue(end_time > user.last_login > start_time)
    self.assertTrue(user.check_password("secretpassword"))
    expected_representation_account = "JohnLennon"
    self.assertEqual(expected_representation_account, str(user))
```

*Codice test 1*

## TEST CLIENT – REGISTER VIEW

1. GET eseguita da un AnonymouUser → funzionamento corretto, status code 200 e template utilizzato corretto
2. POST con form valida → funzionamento corretto, verifica che la form sia valida e l'utente sia stato creato, reindirizzamento all'homepage e controllo che i permessi dell'utente siano corretti
3. POST con password troppo corta → verifica che la form non sia valida e controllo che l'utente non venga creato
4. POST con e-mail non valida → verifica che la form non sia valida e controllo che l'utente non venga creato
5. POST con password che non coincidono → verifica che la form non sia valida e controllo che l'utente non venga creato
6. POST con username già utilizzato → verifica che la form non sia valida e controllo che l'utente non sia stato creato
7. POST con e-mail già utilizzata → verifica che la form non sia valida e controllo che l'utente non sia stato creato
8. POST senza uno dei campi required → verifica che la form non sia valida e controllo che l'utente non sia stato creato

```
def test_can_register_user(self):
    form = RegistrationForm(self.user)
    self.assertTrue(form.is_valid())
    self.assertEqual(form.clean_password2(), self.user['password2'])

    response = self.client.post(self.register_url, self.user, format='text/html')
    self.assertEqual(response.status_code, 302)
    self.assertRedirects(response, reverse("frontpage"))

    users = get_user_model().objects.all()
    self.assertEqual(users.count(), 1)

    my_user = get_user_model().objects.get(username=self.user["username"])
    self.assertTrue(my_user.has_perms(
        ["account.search_item", "account.view_item", "account.view_rate", "account.be_notified",
         "account.sell_item", "account.confirm_shipment", "account.get_paid", "account.view_vendor_admin",
         "account.read_review", "account.buy_item", "account.view_costumer_admin", "account.confirm_delivery",
         "account.write_review", "account.save_item"])))
```

*Codice test 2*

## TEST CLIENT – VIEW ADD\_PRODUCT

È stato testato il comportamento in caso di:

1. GET/POST eseguita da un AnonymouUser → reindirizzamento alla pagina di login, a cui, in caso di successo, segue la pagina per aggiungere un prodotto
2. GET/POST eseguita da un utente loggato ma non di tipo Vendor → reindirizzamento alla pagina per l'upgrade a vendor, a cui, in caso di successo, segue la pagina per aggiungere un prodotto
3. GET eseguita da utente loggato e di tipo Vendor → funzionamento corretto, status code 200 e template utilizzato corretto.
4. POST eseguita da utente loggato e di tipo Vendor con solo campi vuoti nella form → sollevamento degli errori per i campi required, rimanendo sulla stessa pagina
5. POST eseguita da utente loggato e di tipo Vendor con categoria non valida nella form → sollevamento dell'errore per il campo categoria, rimanendo sulla stessa pagina
6. POST eseguita da utente loggato e di tipo Vendor con form valida → funzionamento corretto, verifica che la form sia valida e reindirizzamento alla vendor\_admin

```
def test_add_product_view_with_vendor_post_valid_form(self):
    self.client.force_login(get_user_model().objects.filter(username="user0").first())
    user = auth.get_user(self.client)

    product = {'category': 1, 'title': 'test', 'price': 10, 'quantity': 1, }
    form = ProductForm(product)

    self.assertTrue(form.is_valid())
    response = self.client.post(self.add_product_url, product)
    self.assertEqual(response.status_code, 302)
    self.assertRedirects(response, reverse("vendor_admin"))
```

*Codice test 6*

## RISULTATI

Risultati indicate da coverage:

	<i>senza i test</i>	<i>con i test</i>
<i>apps/account/models.py</i>	55%	100%
<i>apps/vendors/views.py</i>	28%	50% (100% per la view analizzata)
<i>TOT</i>	43%	73%

## Possibili sviluppi

Tra i possibili sviluppi del progetto si considera la possibilità di inserire un sistema di messaggistica tra possibile acquirente e venditore. Questo potrebbe avvenire utilizzando la già sfruttata API django-notifications, oppure tramite scambio di e-mail. In quest'ultimo caso si rende necessaria la verifica dell'e-mail all'atto di registrazione.

Si potrebbe anche introdurre la funzione “salva articolo per dopo” e la possibilità di comprare solo parte degli articoli nel carrello.

Ulteriormente, avendo già implementato la funzione che permette di salvare articoli, si potrebbero introdurre notifiche come “L'articolo che hai tra i preferiti è stato salvato da un altro utente, non lasciartelo scappare!” oppure “L'articolo che hai tra i preferiti è stato venduto”.

Si considera, infine, interessante la possibilità di caricare più di una immagine per articolo.