

Progetto e-commerce

Basi del progetto

Si è voluto modellare un sito web per un negozio che cerca di vendere dei prodotti (quali anime e manga) online.

Il sito cerca quindi di essere tenuto semplice ma al contempo di avere le funzionalità vitali per un e-commerce, per esempio la gestione delle vendite di prodotti.

Si è partiti col pensare ai tipi di utente e ai tipi di prodotti che possono esistere all'interno di questa applicazione.

Prodotti

Esistono due tipi di prodotti: manga e anime. Presentano però attributi uguali, per esempio entrambi hanno titolo, autore/i, prezzo e così via.

Si è quindi deciso di non usare alcun attributo per riconoscere un manga da un anime in quanto verrà specificato direttamente nella descrizione del venditore.

Categorie

Ogni prodotto potrà però far parte di una certa categoria. Si è deciso di modellare questo insieme di 'oggetti' in quanto potrà tornare utile per capire i gusti dei vari compratori e per far sì che venga data una spiegazione meno dettagliata senza che il venditore specifichi ulteriori cose sul prodotto.

Utenti

Esistono 4 tipi di utenti in questo sito. Vengono elencati qua sotto dal meno importante a quello con privilegi maggiori:

- Anonimo; Utente senza particolari privilegi.
- Cliente; Un cliente è l'utente che si è appena loggato (dopo registrazione o meno).
- Venditore; E' l'utente creato dal tecnico amministrativo del sito.
- Tecnico; Amministratore del sistema che può aggiungere nuovi addetti alla vendita.

Anonimo

E' l'utente che può visionare e mettere nel carrello i prodotti che vuole ma non potrà né comprarli né recensirli. Se vorrà accedere a contenuti aggiuntivi per i clienti dovrà registrarsi o, se possiede già un account, loggarsi nel sito.

Cliente

Dopo aver effettuato il Login nel sito potrà godere di alcuni privilegi. Essendo il compratore del sistema sarà in grado di comprare e recensire prodotti(questo anche più volte). Dopo aver comprato i prodotti potrà vedere i suoi ordini passati.

Inoltre dopo aver fatto alcuni ordini potrà vedere delle raccomandazioni che consiglieranno cosa comprare in base alle categorie da lui preferite.

Potrà poi cambiare il suo profilo inserendo un indirizzo diverso, username e email non sarà possibile cambiarli.

Venditore

Il venditore viene creato solo da un utente tecnico per evitare che possano farlo in tanti, per modificare un utente normale in venditore o viceversa bisognerà andare nel pannello admin.

Esso potrà aggiungere nuove categorie e cambiare i prodotti dopo averli creati.

Non sarà possibile per il venditore comprare, recensire e visionare la propria storia degli ordini passati.

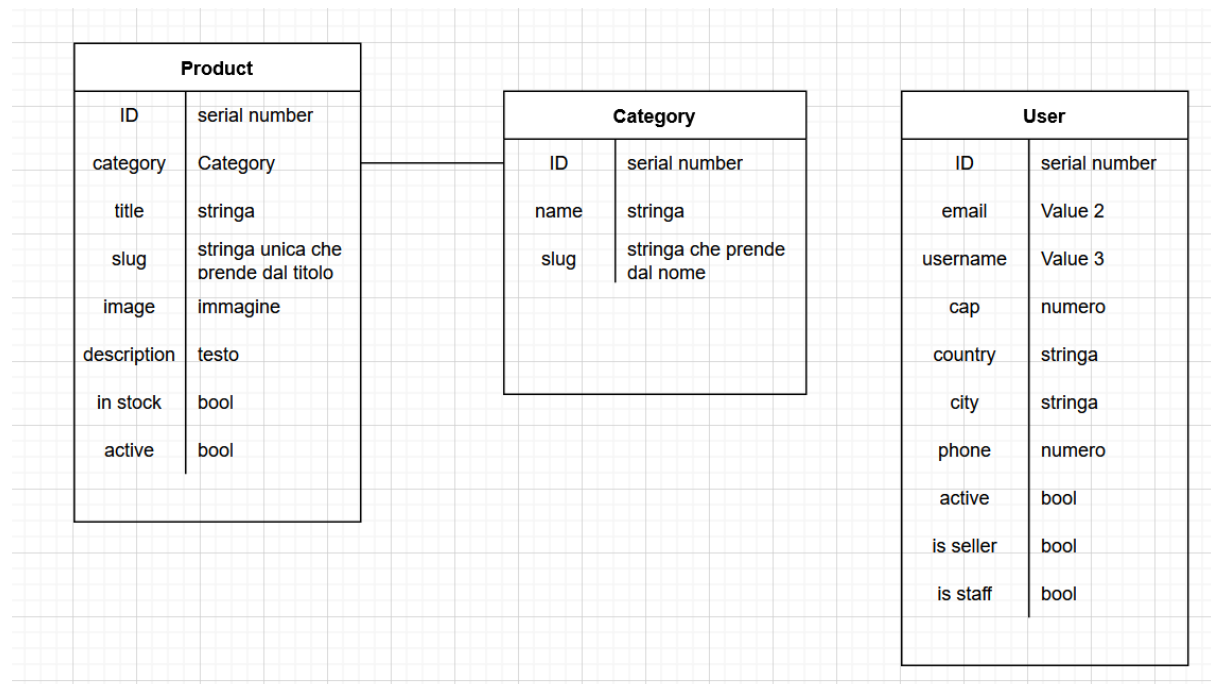
Tecnico

Anche conosciuto come Admin, è colui che ha i privilegi maggiori e potrà quindi compiere ogni azione dal suo pannello di controllo.

Al contrario, nel sito, non potrà svolgere qualsiasi azione anzi sarà molto simile all'utente venditore con l'aggiunta che potrà aggiungere nuovi venditori e che non sarà per lui possibile eliminare l'account admin.

In ogni caso per ogni user tranne l'anonimo verrà usato lo stesso oggetto dove però si attiveranno alcune variabili booleane per distinguere il tipo di utente.

Riportiamo di seguito uno schema che mostra ad alto livello i vari oggetti creati al momento.



Notiamo però che per ora non possiamo modellare oggetti come le recensioni e gli ordini dei clienti. Introduciamo quindi questi elementi riportando poi le rispettive tabelle.

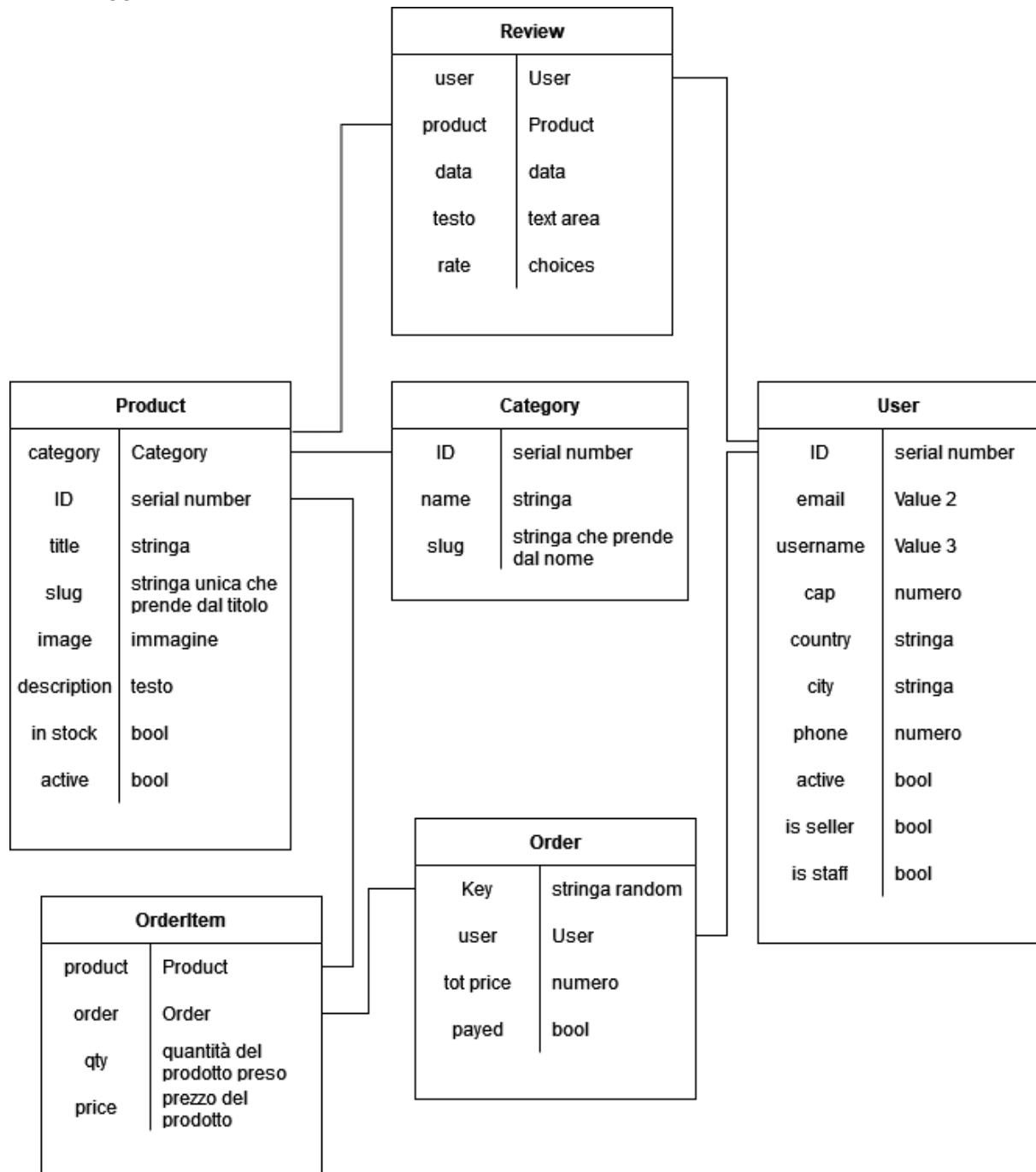
Recensione

Le recensioni vengono create da un utente cliente per poter dare una valutazione del prodotto. Avranno quindi una valutazione in numeri da 1 a 5 e poi una descrizione per poter dare ad altri utenti la possibilità di capire cosa è piaciuto o meno del prodotto.

Ordine

Un ordine è composto dal compratore e dai singoli prodotti che l'utente ha acquistato. Si è quindi deciso di creare due tipi di ordini, uno che collega un certo ordine con l'utente e l'altro che collega quello specifico ordine con ogni prodotto acquistato.

Tabella aggiornata

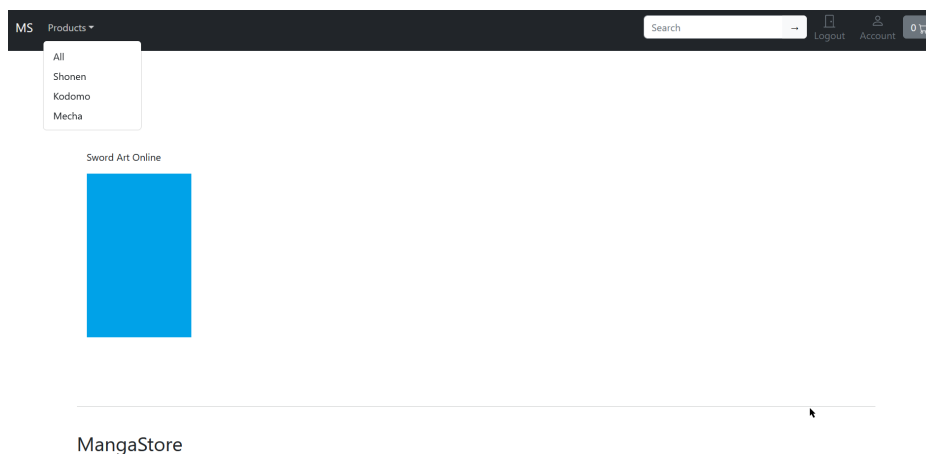
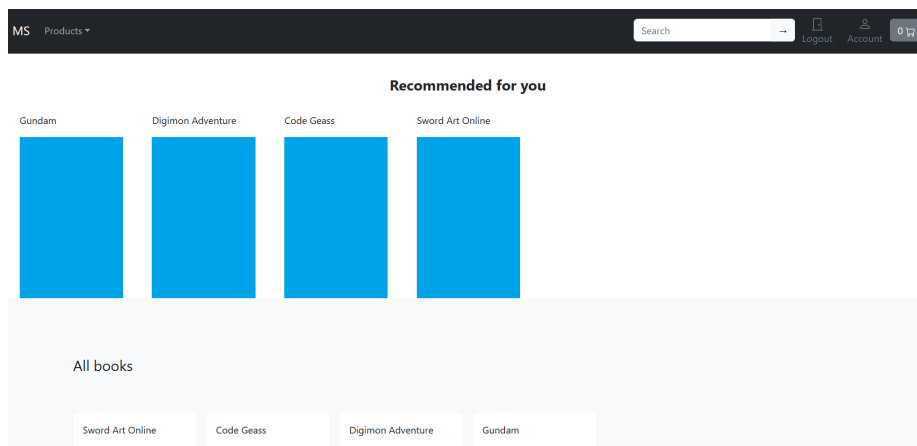


Funzioni di base

Passiamo quindi a parlare dell'implementazione di tutte le funzioni del progetto.

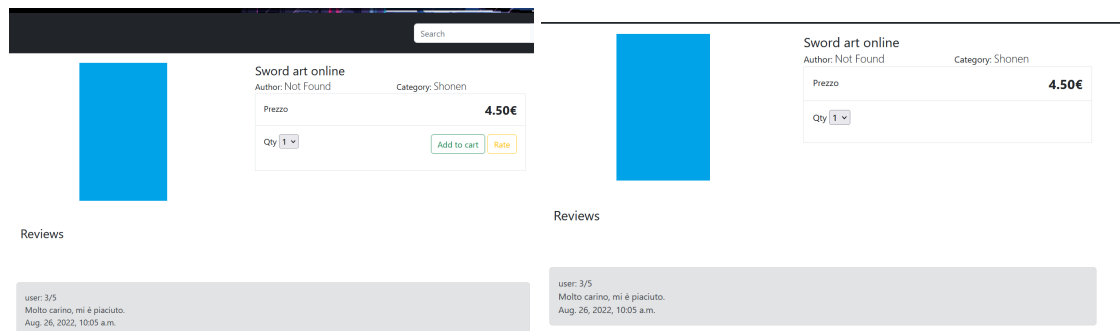
Si è deciso di partire dalla homepage che ospiterà una visione d'insieme di ogni prodotto e un menù a tendina per poter vedere solo i prodotti di una certa categoria. Inoltre, se l'utente è loggato come cliente può vedere la parte di raccomandazioni basate su ciò che ha comperato in passato e quello che in generale è piaciuto di più agli utenti.

```
def categories(request):  
    """ Ritorna tutte le categorie nel database. """  
  
    return { 'categories': Category.objects.all() }  
  
def product_all(request):  
    """  
        Ritorna tutti i prodotti nel database.  
        Serve per la home page.  
    """  
  
    products = Product.objects.all()  
    ctx = { 'products': products, 'recommend': recommend(request) }  
    return render(request, template_name='store/home.html', context=ctx)
```



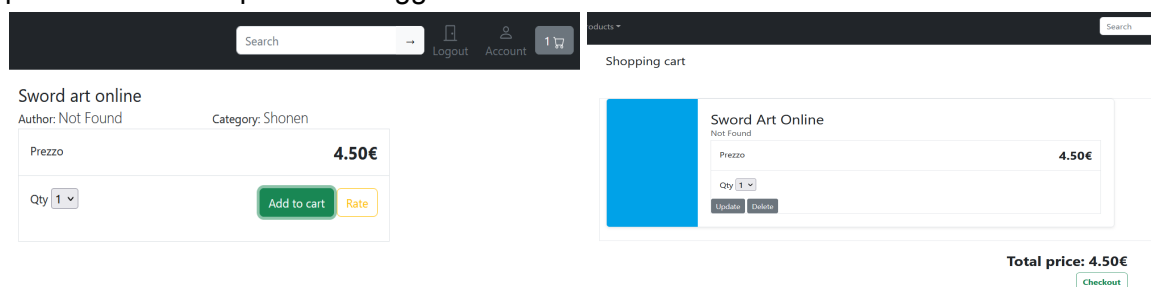
Dopodiché si è passati a modellare la visione dei singoli prodotti. Si è pensato quindi di mettere in questa pagina le recensioni fatte dagli utenti.

Viene quindi passata alla richiesta della pagina, non solo i dettagli di un singolo prodotto ma anche le recensioni per quel prodotto. Si noti anche che un utente può sia ricomprare l'oggetto sia recensirlo di nuovo. Si è deciso di renderlo possibile in quanto poteva essere anche vista come una piattaforma di recensione dei manga/anime oltre che poterne comprare per cui un utente può aver già letto/visto quel prodotto e non averlo comprato. La visione di questa pagina da parte dell'admin e del venditore saranno invece diverse: i bottoni per comperare e recensire non ci saranno.



Inoltre se admin o seller cercano comunque di compiere queste azioni tramite la scrittura manuale della pagina (ad esempio con [http://127.0.0.1:8000/rating/create_review/sword art online/](http://127.0.0.1:8000/rating/create_review/sword%20art%20online/)) essi verranno rispediti nella homepage come se non fosse successo nulla.

Per implementare la logica di aggiunta al carrello invece si è usata una funzione che prendeva la quantità che il cliente voleva e, dopo aver cliccato il pulsante andava ad aggiornare il numero di fianco all'icona col carrello. Di sotto riportato l'aggiornamento ad un prodotto subito dopo averne aggiunto uno e la vista del checkout.



Si può cambiare la quantità selezionandola tramite la tendina e infine cliccare su update per aggiornare. Questa azione farà sì che venga aggiornato il contenuto del carrello e l'icona. Se il cliente non vuole più quel prodotto può però cancellarlo con l'apposito bottone che, anche in questo caso, aggiornerà la pagina.

Dopodiché ci si è occupati della logica per il pagamento. Non volendo simulare completamente una vera transazione di soldi si è cercato di rendere la funzione il più semplice possibile. Tutto quello che avviene è che, dopo aver cliccato sul bottone di checkout, il cliente viene portato in una pagina che chiede il numero di carta (si può inserire un numero a caso), se viene inserito un numero allora il pagamento sarà avvenuto con successo e si potrà vedere il nuovo ordine nella pagina di 'ordini passati'. Se in qualche modo gli utenti admin e seller riescono ad arrivare alla pagina di pagamento essi non ne

vedranno la conferma ma apparirà una schermata dove verranno avvisati che non possono fare ordini.

Payment
Credit or debit card

Payed successfully!

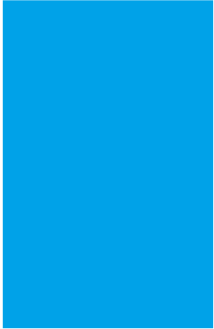
Thank you for choosing MangaStore

[See your past orders here](#)

Seller or staff member can't buy products.
Please login with another user account.
Note: after the logout the cart will be deleted.

Si noti come, dopo aver fatto un ordine, la quantità disponibile di prodotti diminuirà della quantità acquistata. Per esempio se acquistiamo l'unico prodotto chiamato Sword art online, il prodotto non sarà più disponibile, questo influisce anche sulla visione del prodotto.

Sword Art Online !



Sword art online

Author: Not Found Category: Shonen

Prezzo	4.50€
--------	--------------

This product is not available now

Si è passati quindi a modellare la visione del profilo utente e degli ordini passati. Il profilo cambia per ogni tipo di utente. Riportiamo di seguito le diverse visioni in sequenza per admin, seller e infine utente normale (ad un anonimo verrà semplicemente chiesto di loggarsi).

Admin:

Your Dashboard

Manage your personal details

Admin panel

Add new seller account

Add or edit a product

Add a new category

[Change detail](#)

[Enter](#)

[Add seller](#)

[Product](#)

[Add category](#)

Venditore:

Your Dashboard

Manage your personal details

Add or edit a product

Add a new category

[Change detail](#)

[Product](#)

[Add category](#)

Cliente:

Your Dashboard

Manage your personal details
See all your orders

[Change detail](#)
[Order history](#)

Come si può notare sono state aggiunte tutte le funzionalità che venivano elencate nella parte degli utenti in *base del progetto*.
Guardiamo quindi gli ordini passati dell'utente per vedere se è stato effettivamente salvato l'ordine di esempio.

Your orders

Aug. 26, 2022, 10:33 a.m. Total paid: **4.50€**



Sword Art Online
Qty: 1

[Leave a review](#)

Nella pagina, qui tagliata a dovere per semplificazione, è presente un solo ordine: quello fatto per una copia di Sword art online.

Il cliente, se vorrà potrà lasciare una recensione o comprarlo di nuovo andando nuovamente nella pagina del prodotto.

Facciamo allora un'altra recensione per mostrare come viene modellata la visione del form.

Rate it!

Rate

1 - Trash

1 - Trash

2 - Bad

3 - Ok

4 - Nice

5 - Perfect

[Submit](#)

Dopo aver fatto la recensione si è deciso di rimandare il cliente nella pagina del prodotto in modo che potesse vedere la propria recensione e capire quindi se è stata aggiunta con successo.

Reviews

user: 1/5
M E H
Aug. 26, 2022, 12:40 p.m.

user: 4/5
Bellerrimo
Aug. 26, 2022, 12:41 p.m.

Proseguendo con l'analisi del profilo utente guardiamo com'è vista la pagina di modifica profilo, non verranno fatte vedere quelle di tutti i tipi di utenti in quanto in questo caso sono uguali. Sono state riportate due foto poiché non ci stava tutta la pagina.

Change your profile

Account email cannot be changes

user@us.com

Username

user

Country

Italy

City name

Modena

Address

via Rossi 23

Phone number

33442312312

Postal code

41001

Save

Want to delete your account?

Delete

In realtà l'account non viene propriamente cancellato del tutto ma rimane nel database con il valore `is_active` a False, cosicché possa essere ripristinato dall'utente admin. Si noti che esso però non potrà cancellarsi. Si è presa questa decisione dato che non sembrava sensato poter cancellare l'unico amministratore.

Equivalentemente ai form per le review sono state fatte form per aggiungere/cambiare prodotti e per creare nuove categorie. Questo è esclusivamente fatto tramite i pannelli nel profilo admin e seller, di seguito le immagini.

Create a new product

Category

Enter title

Title

Author

Not found

Description

Image

Sfoggia... Nessun file selezionato.

Available

1

Price

4,5

Save

Alcuni valori come author, image, available e price sono valori di default. In ogni caso possono essere sovrascritti con dei valori diversi. Per esempio può essere aggiunta un'immagine diversa da quella unicolor azzurra. Dopo aver spinto save, viene salvato il nuovo prodotto ma se esiste già un articolo con un titolo uguale allora viene aggiornato con le credenziali appena inserite.

Create a new category

Enter name

Name

Save

Proseguendo con le funzioni disponibili per un admin troviamo l'aggiunta di un nuovo seller. Il form viene usato per la registrazione sia di nuovi utenti sia per quella di nuovi seller ma vengono poi implementate logiche diverse per creare i nuovi utenti. Questo avviene tramite la 'spunta' del valore `isSeller` a `True` che di default sarebbe `False`.

```
@login_required
def add_seller(request):
    """ Viene aggiunto un nuovo membro di venditori. Solo un membro dello staff

    if not request.user.is_staff:
        return redirect('/')

    if request.user.is_staff and request.method == 'POST':
        registerform = RegistrationForm(request.POST)

        if registerform.is_valid():
            user = registerform.save(commit=False)
            user.email = registerform.cleaned_data['email']
            user.set_password(registerform.cleaned_data['password'])

            user.country = registerform.cleaned_data['country']
            user.city = registerform.cleaned_data['city']
            user.address = registerform.cleaned_data['address']
            user.phone_num = registerform.cleaned_data['phone_num']
            user.cap_code = registerform.cleaned_data['cap_code']

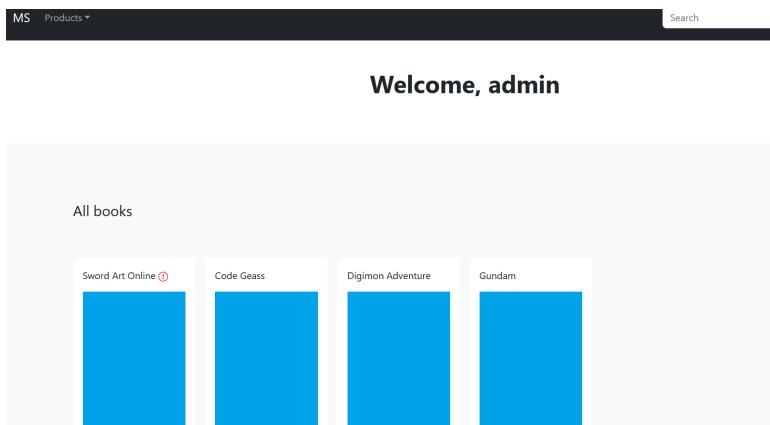
            user.is_active = True
            user.is_seller = True
            user.save()
            logout(request)

            return redirect('/account/login/')_
        else:
            registerform = RegistrationForm()

    return render(request, 'account/register.html', { 'form': registerform })
```

Si è pensato inoltre che, dopo l'aggiunta di un nuovo seller avvenuta con successo, l'admin debba provare ad inserire le nuove credenziali e per questo viene fatto il logout dell'admin che dovrà poi provare ad entrare con i nominativi del nuovo venditore.

In ogni caso, dopo ogni login, si può capire con quale utente si è entrati poiché nella homepage viene 'salutato' lo user con il suo username. Per esempio se siamo entrati con un utente di nome admin verrà scritto "Welcome, admin" e così via.



Passiamo quindi a vedere la funzione di ricerca.

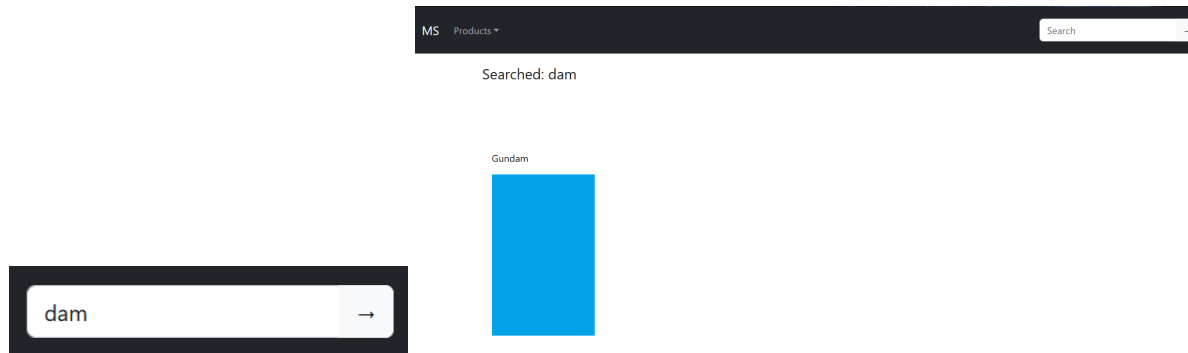
Dopo che un qualunque utente ha cliccato il pulsante → la ricerca parte.

Se non viene immesso niente allora tornerà alla pagina principale (con tutti i prodotti) in caso contrario verranno cercati i prodotti con quella stringa al loro interno. Con all'interno si

intende che si cerca nel titolo, nella descrizione e tra gli autori; dopodiché si cerca se la parola è il nome di una categoria se si procede a mettere nella risposta tutti i prodotti di quella categoria.

Si è deciso per cui si sfrutta il template usato per la visione di tutti i prodotti di una categoria e, al posto del nome di essa, come titolo ci sarà la parola cercata.

Proviamo quindi a cercare la parola 'dam' per vedere se viene trovato il manga *Gundam*.



Test effettuati

Partendo dagli account si sono testati modello e view.

Del modello vengono implementati test più semplici come la verifica dell'istanza e della funzione str.

Per le view vengono invece implementati test su funzioni quali: login, registrazione, aggiunta di un seller, visione profilo e logout. Il più interessante è indubbiamente il penultimo.

Vengono infatti testate come vengono trattati tutti i tipi di utenti che richiedono la visione del profilo.

Ad esempio se un utente anonimo fa la richiesta la risposta dovrà essere che deve prima fare il login.

Inoltre, altro aspetto interessante, viene provato come per utenti di base non può essere richiesta l'aggiunta di un utente venditore.

```
def test_addseller(self):
    """ Testa se viene richiesto il login se un utente non loggato """

    resp = self.client.post(reverse_lazy('account:add_seller'))
    self.assertEqual(resp.status_code, 302)

    # Se l'utente è normale
    u = UserBase.objects.get(username='user')
    self.client.force_login(u)
    resp = self.client.post(reverse_lazy('account:add_seller'))
    self.assertEqual(resp.status_code, 302)
    self.assertRedirects(resp, '/')

    # Se l'utente è un admin
    u = UserBase.objects.get(username='admin')
    self.client.force_login(u)
    resp = self.client.post(reverse_lazy('account:add_seller'))
    self.assertTrue(resp.context['user'].is_authenticated)
    self.assertEqual(resp.status_code, 200)
    self.assertTemplateUsed(resp, 'account/register.html')
```

Per il carrello vengono testate le funzioni di base di aggiunta, update e delete, inoltre viene controllato che venga visionato correttamente il sommario degli acquisti.

Per gli ordini si testa come vengono aggiunti nuovi ordini solo se l'utente è registrato come compratore.

```
def test_add(self):
    resp = self.client.post(reverse_lazy('payment:cartview'))
    self.assertEqual(resp.status_code, 302)

    u = UserBase.objects.get(username='user')
    self.client.force_login(u)
    resp = self.client.post(reverse_lazy('payment:cartview'))
    self.assertEqual(resp.status_code, 200)
    self.assertTemplateUsed(resp, 'payment/home.html')
    self.assertTrue(resp.context['user'].is_authenticated)

    resp = self.client.post(reverse_lazy('orders:add'),
                            {'action': 'post', 'csrfmiddlewaretoken': resp.context['csrf_token'], 'order_key': resp.context['client_secret']})

    self.assertEqual(resp.status_code, 200)

    o = Order.objects.all()
    self.assertIsNotNone(List(o))
    o = OrderItem.objects.all()
    self.assertIsNotNone(List(o))

    p = Product.objects.get(title='django advanced')
    self.assertFalse(p.in_stock)
```

Infine vediamo i test più importanti fatti per i modelli e le view che riguardano prodotti, categorie e review.

Per i tre modelli, come per gli account, viene controllato se str e l'istanza sono corrette.

Per le view invece vengono controllati alcuni dei form usati per creare delle nuove entry per questi modelli e cosa succede per ogni utente che chiamante.

Prendendo ad esempio il test_create_product si può vedere come per ognuno dei tipi di utente viene testato se possono eseguire questa azione:

se si allora si prova che viene aggiunta una nuova istanza

altrimenti viene provato che vengono usati template diversi o vengono fatte redirezioni.

```
def test_create_product(self):
    """ Testa che venga correttamente salvata l'istanza di un nuovo prodotto e che non sia accessibile a tutti gli utenti """

    # Prova con utente anonimo
    resp = self.client.post(reverse_lazy('store:create_product'))
    self.assertEqual(resp.status_code, 302)
    self.assertTemplateNotUsed(resp, '/')

    # Prova con utente cliente
    u = UserBase.objects.create_user(username='user1', email='user@us.com', password='user1')
    self.client.force_login(u)
    resp = self.client.post(reverse_lazy('store:create_product'))
    self.assertEqual(resp.status_code, 302)
    self.assertRedirects(resp, '/')

    # Prova con utente admin
    u = UserBase.objects.create_user(username='admin', email='admin@a.com', password='admin', is_staff=True)
    self.client.force_login(u)
    resp = self.client.post(reverse_lazy('store:create_product'))
    self.assertEqual(resp.status_code, 200)
    self.assertTemplateUsed(resp, 'store/products/createprod.html')
    self.assertEqual(str(resp.context['user']), 'admin')
    self.assertTrue(resp.context['user'].is_authenticated)

    # Prova con utente venditore
    u = UserBase.objects.create_user(username='seller', email='seller@s.com', password='seller', is_seller=True)
    self.client.force_login(u)
    resp = self.client.post(reverse_lazy('store:create_product'))
    self.assertEqual(resp.status_code, 200)
    self.assertTemplateUsed(resp, 'store/products/createprod.html')
    self.assertEqual(str(resp.context['user']), 'seller')
    self.assertTrue(resp.context['user'].is_authenticated)
```

Altro test interessante è quello per la ricerca `test_search` che fa due prove. Una per provare che cercando la parola 'django' ci sia effettivamente un'unica risorsa trovata e, secondo, che cercando 'django2' venga tutto renderato correttamente ma non ci sia nessuna risorsa trovata.

```
def test_search(self):
    """ Test che prova la funzione di ricerca dei prodotti, essendo un azione disponibile a tutti i tipi di utente non vi è alcun login. """

    resp = self.client.get(reverse_lazy('store:search'), {})
    self.assertEqual(resp.status_code, 302)
    self.assertRedirects(resp, '/')

    resp = self.client.get(reverse_lazy('store:search'), {'word': 'django'})
    self.assertEqual(resp.status_code, 200)
    self.assertTemplateUsed(resp, 'store/products/category.html')
    self.assertEqual(str(resp.context['category']), 'Searched: django')
    self.assertEqual(len(List(resp.context['products'])), 1)

    resp = self.client.get(reverse_lazy('store:search'), {'word': 'django2'})
    self.assertEqual(resp.status_code, 200)
    self.assertTemplateUsed(resp, 'store/products/category.html')
    self.assertEqual(str(resp.context['category']), 'Searched: django2')
    self.assertEqual(len(List(resp.context['products'])), 0)
```

Conclusione

Per finire si può dire che il programma risultato da questo lavoro non è sicuramente perfetto per andare in produzione ma è stato utile per un primo approccio alle tecnologie web.

Si può inoltre dire che nonostante le varie semplificazioni introdotte, come il pagamento falso, molte cose sono state di difficile implementazione: per esempio il recommendation system.

Altro elemento di difficoltà sono stati i test, nonostante si avesse in mente quali fare, avendo cambiato la creazione di un utente risultava difficile fare test su login o parti che ne richiedessero l'accesso.