

UCommerce

Yuri Gardinazzi

Progetto di Tecnologie Web

A.A. 2019/2020



UCommerce è un e-commerce in cui gli utenti possono acquistare i prodotti ma anche vendere i propri, in particolare la tipologia dei prodotti può spaziare dall'usato a veri e propri prodotti creati dall'utente.

Il progetto è basato su Django, HTML, JS, CSS, AJAX, jQuery, python, Bootstrap, sqlite

Le tipologie di utenti sono

- Anonimo
- Utente registrato
- Utente venditore
- Admin

L'utente anonimo può accedere alla pagina senza effettuare alcuna registrazione, ed inoltre può effettuare ricerche, ma non avrà suggerimenti dati dal recommendation system.

L'utente registrato per essere tale deve effettuare la registrazione e successivamente il login. Un utente registrato può effettuare tutte le operazioni dell'utente anonimo ma con funzionalità aggiuntive. Può comprare i prodotti e controllarne lo stato dell'acquisto, inoltre può effettuare recensioni dei prodotti.

Se lo desidera può diventare un utente venditore **permanentemente** ed inoltre avrà dei prodotti suggeriti dal recommendation system nei risultati delle sue ricerche.

L'utente venditore per essere tale deve essere un utente che si è registrato come venditore durante la registrazione oppure un normale utente registrato che ha spuntato "is vendor" nelle impostazioni dell'account. Non può ritornare ad essere un utente registrato normale.

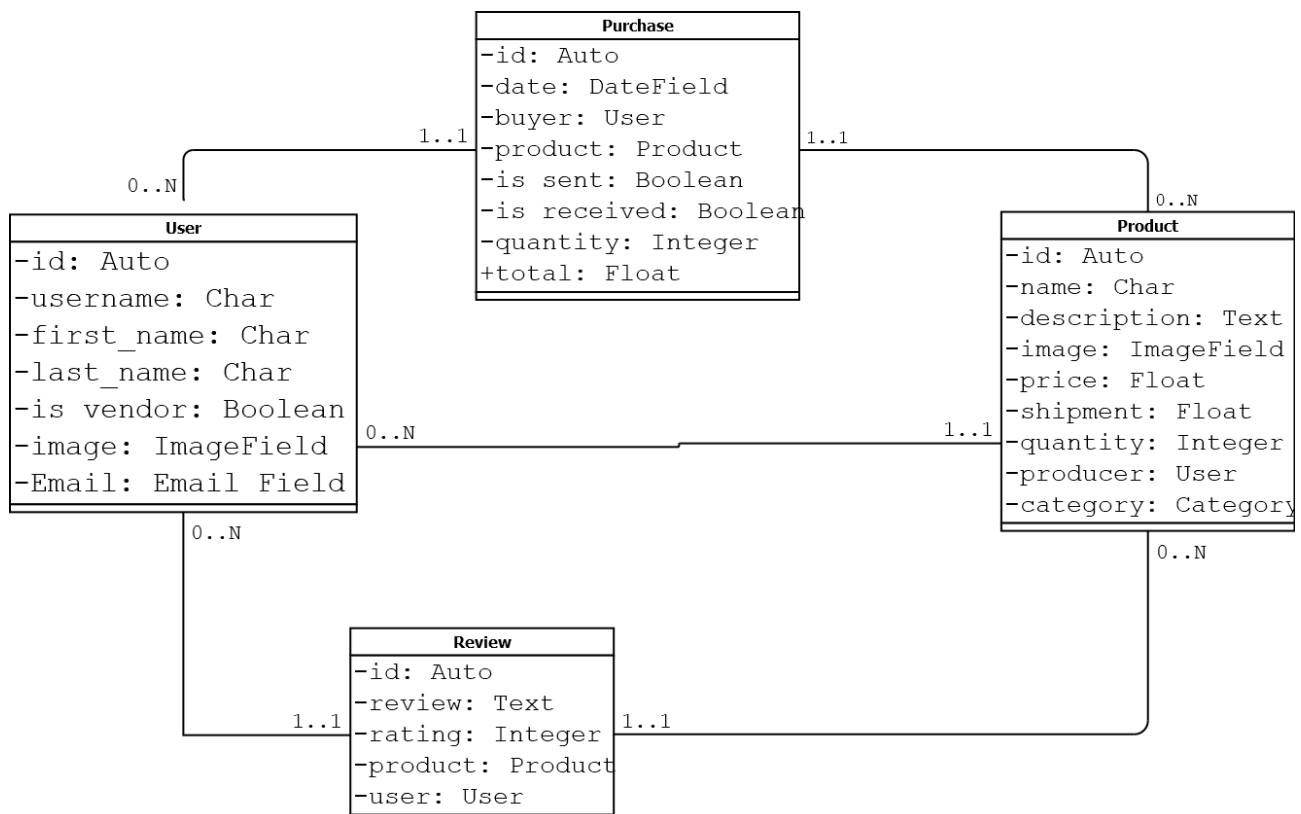
Può effettuare tutte le operazioni dell'utente registrato, ma a differenza dell'utente registrato può vendere i propri prodotti, modificarli e gestire lo stato degli ordini.

L'admin è l'utente admin di django, il cosiddetto "superuser". Può inserire, modificare ed eliminare ogni oggetto del database. Per accedere all'interfaccia di admin si utilizza l'url relativo standard di django "/admin". Infine è l'unico utente

che può aggiungere le categorie che saranno poi rese disponibili come opzioni per i prodotti e le ricerche.

Il database implementato è quello di default di Django, cioè SQLite.

Diagramma delle classi del Database



User – Review: L'utente può decidere se fare o meno delle recensioni ai prodotti, e ne può fare quante ne vuole.

Review – Product: Ogni recensione riguarda un solo prodotto ed è composta dal voto e il commento dell'utente.

User – Product: Quando un utente è abilitato può creare quanto prodotti vuole, e ciascun prodotto è associato ad un solo utente. **Sarà il backend a gestire la creazione dei prodotti per gli utenti abilitati**

User – Purchase: Un utente può effettuare da 0 a N acquisti.

DJANGO – APP

Essendo Django la tecnologia core di questa piattaforma è stato necessario per una migliore leggibilità del codice dividere l'applicazione in più “django app”, che essenzialmente sono codice e pezzi del database scritti e divisi in cartelle diverse. Le django app sviluppate per questo processo sono

- **user_management**
- **product_management**
- **sales**
- **ecommerce**

ecommerce è quella generata col nome del progetto (che in origine era solo ecommerce) e si occupa di gestire la home la pagina di ricerca

product_management come suggerisce il nome gestisce i prodotti, quindi l'aggiunta e modifica dei prodotti e la visualizzazione della dashboard per l'utente venditore per permettergli di fare tali operazioni.

Infine implementa anche la pagina di “product detail” cioè la visualizzazione del prodotto per ogni tipo di utente, e la possibilità di aggiungere recensione da parte degli utenti registrati.

user_management è l'app per la gestione degli utenti e si occupa di: login, registrazione, profilo utente e sistema di cambio e reset password con conferma via mail.

sales si occupa invece della gestione degli acquisti, fornendo quindi all'utente la lista degli acquisti da lui effettuati con relativo stato “spedito, ricevuto” nella pagina Orders. Si occupa anche del lato venditore fornendo la pagina Sales dove il venditore può vedere la

BACKEND – alcuni dettagli

Le immagini dei prodotti e degli utenti sono salvate in una cartella definita dalla variabile `MEDIA_ROOT`

```
MEDIA_ROOT = os.path.join(BASE_DIR, 'user_management/media')
```

In particolare vengono salvate all'interno dell'app **user_management**

Per la gestione del reset della password tramite mail è stato utilizzato il backend_email di Django che permette di inviare mail nel terminale invece che ad un server mail vero e proprio.

```
EMAIL_BACKEND = 'django.core.mail.backends.console.EmailBackend'
```

Inoltre per personalizzare le pagine di *cambio password e reset* sono state utilizzate le views da `django.contrib.auth` ma con un template personalizzato descritto nel parametro `template_name`

esempio

```
path('password_reset/', auth_views.PasswordResetView.as_view(success_url=reverse_lazy('user_management:password_reset_done'),
                                                            template_name='registration/password_reset_form_user.html',
                                                            email_template_name='registration/password_reset_email_user.html'),
```

Da notare i nomi che finiscono per “_user” perché sennò si chiamerebbero come i template default di django per le autenticazioni e verrebbero usati quelli.

Per quanto riguarda le form ove possibile sono state utilizzate le **django-crispy-form** cioè Form class-based che vengono generate Client-side.

Esempio di form:

```
class ProductCrispyForm(SubmitButtonMixin):
    category = CategoryChoiceField(queryset=MyCategory.objects.all())

    helper = FormHelper()
    helper.form_id = 'product-crispy-form'
    helper.form_method = 'POST'
    helper.add_input(Submit('submit', 'Save'))

class Meta:
    model = Product
    exclude = ('producer',)
```

AJAX & jQuery

AJAX (Asynchronous Javascript and XML) è una tecnologia che permette di fare richieste asincrone al server, quindi quando la pagina è stata già caricata.

È stata utilizzata in modo per tutto ciò che riguarda i prodotti.

Dalla Dashboard del venditore permettendo di eliminare prodotti e ricreare la tabella, all'acquisto dei prodotti all'inserimento di recensioni.

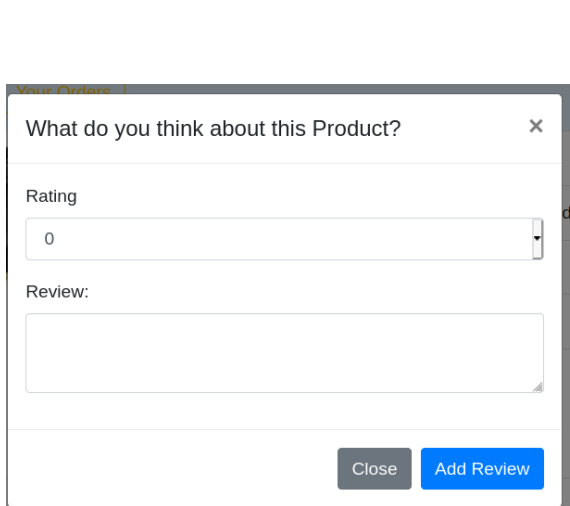
Esempio di recensione effettuata con l'ausilio di AJAX

Path urls.py

```
path('ajax/add_review', views.add_review, name='add_review'),
```

Form html

views.py



```
def add_review(request):
    data = {'success': False}
    if request.method == 'POST':
        review = Review()
        review.product_id = request.POST.get('product_id')
        review.review = request.POST.get('review')
        review.rating = request.POST.get('rating')

        #rating not valid
        if(int(review.rating) < 0 or int(review.rating) > 5):
            data['success'] = False
            response = JsonResponse(data)
            response.status_code=403
            return response

        review.user_id = request.POST.get('user_id')
        review.save()
        data['success'] = True
    return JsonResponse(data)
```

```
$.ajax({
    url : '{% url "product_management:add_review" %}',
    method: 'POST',
    dataType: 'json',
    data: {
        'product_id' : {{ product.id }},
        'user_id' : {{ user.id }},
        'rating' : $('#rating').val(),
        'review' : $('#review').val(),
        'csrfmiddlewaretoken' : '{{ csrf_token }}',
    },
    success : function(){
        $('#ReviewModal').modal('toggle');
        location.reload();
    },
    error : function(err){
        $('#ReviewModal').modal('toggle');
        console.log(err);
        alert('Something went wrong');
    }
});
```

Chiamata ajax, all'interno del campo *data* vengono inviati i dati necessari alla creazione della Recensione.

In *success* ed *error* vengono descritte le funzioni da eseguire in base al messaggio della risposta del server.

TEST

In questo progetto per una manutenzione più efficace del codice sono stati effettuati degli Unittest che in Django si possono implementare in modo estremamente semplice tramite i file *test.py*

In particolare son stati sviluppati test per le Review e per gli utenti (User) e si possono eseguire con i comandi *python manage.py test user_management* e *python manage.py test product_management*.

I test particolarmente interessanti da analizzare sono quelli delle Review che vengono implementate tramite delle AJAX.

```
def test_add_review_negative_rating(self):
    self.login_user()
    data = {
        'type': 'POST',
        'product_id': self.p.id,
        'review': 'blablabla',
        'rating': -10,
        'user_id': self.u.id
    }
    response = self.client.post(reverse_lazy('product_management:add_review'), data)
    self.assertEqual(response.status_code, 403, 'Show 403 if rating is < 0')

def test_add_review_with_too_high_rating(self):
    self.login_user()
    data = {
        'type': 'POST',
        'product_id': self.p.id,
        'review': 'blablabla',
        'rating': 6,
        'user_id': self.u.id
    }
    response = self.client.post(reverse_lazy('product_management:add_review'), data)
    self.assertEqual(response.status_code, 403, 'Show 403 if rating is > 5')
```

Come si può notare son state testate le review con dei valori al di fuori di [0-5], per fare ciò si è sfruttato lo status code 403 che abbiamo inserito nella funzione lato server.

RECOMMENDATION SYSTEM

Per la creazione del recommendation system l'obiettivo è stato quello di consigliare prodotti agli utenti sulla base degli acquisti di utenti "simili" all'utente loggato. Dove per "utente simile" si intende un utente che ha acquistato bene o male gli stessi prodotti del nostro utente target.

Vi sono svariati metodi per fare ciò. In questo progetto è stato utilizzato il Coefficiente di similarità di Jaccard che permette di ottenere un grado di similarità tra vettori binari.

Il fatto che funzioni su vettori binari, cioè con valori 0 o 1 è comodo nel nostro caso d'uso perché i vettori che sono stati utilizzati sono quelli della matrice Utenti x Prodotti dove M_{ij} vale 1 se l'utente U_i ha acquistato il prodotto P_j , 0 altrimenti.

User\products	1	2	3	4	5
1	0	0	1	0	0
2	1	0	1	1	0
3	1	0	1	0	0

Una volta calcolato e preso l'utente con il coefficiente più alto si suggeriscono i prodotti da lui acquistati non presenti nell'utente che stiamo analizzando

RICERCA

La ricerca è stata effettuata tramite una **ajax** prendendo la **categoria** e la **query** dell'utente. Sono stati analizzati i vari casi in cui categoria o la query sono presenti o meno, e tramite il parametro **<attributo>__contains** è stato possibile controllare se la query fosse presente nel nome dell'oggetto. Infine sono stati aggiunti i prodotti suggeriti nel caso l'utente sia autenticato.

```
def get_search(request):
    context = {}
    if request.method == 'POST':
        query = request.POST['query']
        category = request.POST['category']
        if category == 'all' and query == '':
            products = Product.objects.all()
            context['products'] = products
        elif category != 'all' and query == '':
            products = Product.objects.filter(category_id=category)
            context['products'] = products
        elif category == 'all':
            print(" category all - query presente")
            products = Product.objects.filter(name__contains=query)
            context['products'] = products
        else:
            products = Product.objects.filter(name__contains=query, category_id=category)
            context['products'] = products
    items = []
    if(request.user.is_authenticated):
        suggested_items = get_similary_items(request.user.id)
        if(suggested_items):
            for element in suggested_items:
                items.append(Product.objects.get(id=element))

    context['suggested_items'] = items
    return render(request, 'search.html', context)
```

HOME

Ucommerce

Login

Register

All categories

Search

Search

GET RID OF THINGS YOU DON'T NEED ANYMORE

penna rossa
Bic rossa per correggere le verifiche

RICERCA

Ucommerce

Login

Register

All categories

Search




Search

Research for cuscino Category: all

	Name	Description	Price	Availability
	touch: cuscino	Cuscino molto bello e open source	15.0	3
	cuscino microsoft	Cuscino non open source e un po' buggato	15.0	10

PRODOTTI DEL VENDITORE

DASHBOARD OF pippo

Add Product					
	Name	Quantity	Price	Category	Actions
	touch: cuscino	3	15€	Home	Info Delete Edit
	Bicchieri fatto a mano	110	13€	Home	Info Delete Edit
	cuscino microsoft	10	15€	Home	Info Delete Edit

PRODOTTO E REVIEW



touch: cuscino

Description: Cuscino molto bello e open source

Price: 15.0 €

Ratings 3.5/5


[BUY](#) Available: 3

ADD REVIEW

Rating: 3	User: gatto91	Comodissimo
Rating: 4	User: ominide	Mi piace un sacco

PRODOTTI SUGGERITI

Research for Category: all

Suggested Items	Name	Description	Price	Availability
	Algebra Lineare	Libro di analisi lineare	60.0	2

GESTIONE ORDINI

Prodotto appena ordinato (venditore):

Orders

Order ID	Product ID	Buyer	Quantity	Total	Date	Send	Received
17	18	5	2	36€	2020-09-25	Send product	Not sent

Prodotti appena inviato (venditore):

Orders

Order ID	Product ID	Buyer	Quantity	Total	Date	Send	Received
17	18	5	2	36€	2020-09-25	Sent	Sent but not received

Schermata utente:

17	18	2	36€	2020-09-25	Sent	Click here if it has arrived
----	----	---	-----	------------	------	--

Prodotto confermato (utente):

17	18	2	36€	2020-09-25	Sent	Received
----	----	---	-----	------------	------	----------

Prodotto inviato e ricevuto:

Orders

Order ID	Product ID	Buyer	Quantity	Total	Date	Send	Received
17	18	5	2	36€	2020-09-25	Sent	Received