

eCards

Gabriele Giacobazzi

A.A 2019/2020

ECards è un sito di e-commerce destinato ad utenti e piccoli rivenditori che vogliono comprare e/o vendere carte online facilmente.

Il progetto è stato creato principalmente mediante l'utilizzo del framework Django per la parte di back-end, mentre per la parte di front-end sono stati utilizzati HTML, Bootstrap, CSS e JS.

Vi sono quattro categorie di utenti presenti:

- Utente anonimo
- Utente registrato (acquirente)
- Utente registrato (acquirente + venditore)
- Admin

User Management:

Utente anonimo: utente che può navigare il sito ed effettuare ricerche, ma non può effettuare acquisti e/o aprire un proprio negozio.

Utente registrato (acquirente): per diventare utente acquirente bisogna registrarsi sul sito ed effettuare il login. Oltre a ciò che può fare l'utente anonimo, l'utente acquirente può acquistare prodotti e tracciarne il loro stato tramite carrello e profilo. Può lasciare recensioni dei negozi vari presenti sul sito. Inoltre può modificare varie impostazioni dal proprio profilo, quali cambio password e chiusura dell'account.

Un utente acquirente può diventare utente venditore in maniera **permanente** creando un proprio negozio.

Utente registrato(venditore): per diventare utente venditore ci sono due possibilità:

-Al momento dell'iscrizione registrazione per utente venditore

-Diventare utente venditore aprendo un negozio da utente acquirente. (Misura **permanente**)

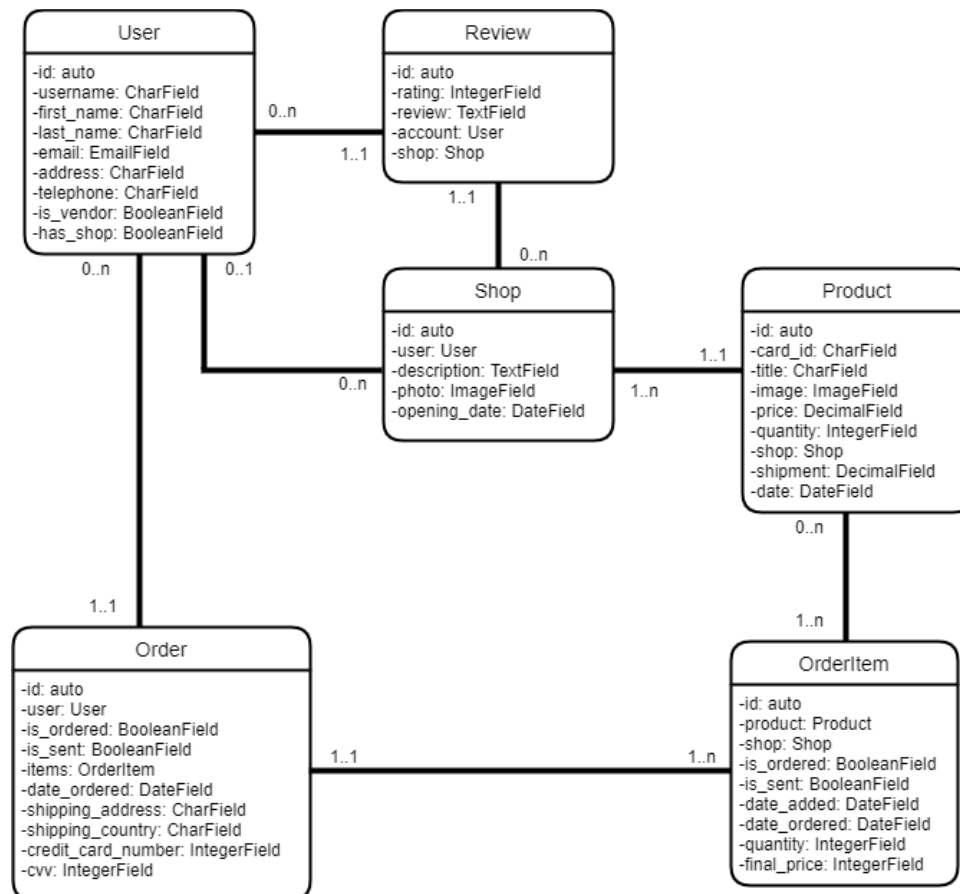
L'utente venditore può fare tutto ciò che è possibile per l'utente acquirente; inoltre l'utente venditore ha un proprio negozio nel quale può vendere i propri prodotti e ha accesso ad una tab speciale nel proprio profilo dal quale può tracciare gli ordini fatti e modificarne lo stato.

Admin: è l'admin di Django. Oltre alla view di default /admin provvista dal framework, l'utente admin ha a disposizione una view personalizzata all'interno del proprio profilo dal quale può bannare/sbannare gli utenti del sito.

Database:

Il database utilizzato dal progetto è SQLite che è quello di default del framework Django.

Di seguito il diagramma delle classi del database:



User-Shop: l'utente può avere fino ad uno shop

User-Review: l'utente può fare quante recensioni vuole

Review-Shop: ogni negozio può avere n recensioni, ma la recensione singola è per un determinato negozio

Shop-Product: ogni negozio può avere n prodotti, ma il singolo prodotto fa parte di un certo negozio

Product-OrderItem: ogni prodotto può fare parte di diversi OrderItem (oggetti che verranno acquistati)

Order-OrderItem: ogni ordine è costituito da almeno un OrderItem

User-Order: un utente può fare da 0 a n ordini

Django Apps:

All'interno del progetto sono state sviluppate le seguenti apps:

- card_ecommerce
- user_management
- product
- shop
- cart

card_ecommerce: app di default creata da Django al cui interno sono definite le pagine base e nel quale viene gestita la ricerca all'interno del sito.

user_management: app che gestisce utenti e le loro funzionalità quali registrazione, login, recupero password, cambio password e profilo personale

product: app che gestisce i prodotti, dalla loro creazione alla modifica degli stessi.

shop: app che gestisce i negozi presenti sul sito.

cart: app che gestisce un carrello, disponibile per gli utenti registrati in maniera tale da poter effettuare acquisti all'interno del sito.

Card_ecommerce features:

Ricerca:

All'interno del sito è possibile fare ricerche grazie ai comandi di SQLite molto semplici e ad una chiamata Ajax che contiene la query effettuata dall'utente.

I risultati ottenuti vengono poi filtrati utilizzando il parametro <attributo>__gt in maniera da ottenere solo i prodotti che hanno quantità diversa da 0.

```
def search_view(request):
    context = {}

    if request.method == 'POST':
        prod_query = request.POST['prod_query']
        if prod_query == '':
            products_retrieved = Product.objects.all().filter(quantity__gt=0)
            context['products'] = products_retrieved
        else:
            products_retrieved = Product.objects.filter(title__contains=prod_query, quantity__gt=0)
            context['products'] = products_retrieved

    return render(request, 'search.html', context)
```

User_management features:

Reset Password Utente:

È possibile resettare la propria password e ricevere la mail di cambio password sul terminale tramite il modulo EmailBackend di Django.

Chiusura Account Utente:

L'utente registrato (che **non** sia Admin) ha la possibilità di chiudere l'account, nello specifico disattivare, tramite le impostazioni del proprio profilo. A livello di codice questo viene effettuato trovando innanzitutto il profilo e poi settando il parametro **is_active** a False

```
def close_account(request):  
    if request.method == 'POST':  
        try:  
            user = CustomUser.objects.get(username=request.user.username)  
            user.is_active = False  
            user.save()  
            logout(request)
```

Utilizzando lo stesso meccanismo viene implementata la funzione di ban/unban per l'utente admin.

Product features:

Creazione di prodotti:

Per la creazione dei prodotti viene ottenuta la lista dei set ai quali i prodotti appartengono tramite una chiamata api esterna (nel nostro caso 'scryfall.com').

```
response = requests.get('https://api.scryfall.com/sets')  
request.session['data'] = response.json()
```

Il json ottenuto dalla risposta della chiamata contiene tutti i set che poi potranno essere utilizzati per caricare i prodotti veri e propri.

Una volta scelto il set viene ottenuta una nuova lista che presenta tutti i prodotti possibili:

```

let setData = 'https://api.scryfall.com/sets/' + data2;
fetch(setData, {
  mode: 'cors'
})
.then(response => response.json())
.then((data) => {
  console.log(data);
  let cardsData = data.search_uri;

```

Con questo nuovo json verrà poi creata una tabella contenente i prodotti possibili del set scelto in precedenza. La tabella creata nel nostro caso è una DataTable ed è un tipo speciale di tabella che permette la paginazione dei risultati e la ricerca tra gli stessi tramite una barra di ricerca posta sopra di essa.

```

// Clear old table
$('#tableData').dataTable().fnDestroy();

$('#tableData').dataTable({
  ajax: {
    url: cardsData,
    type: "GET",
    dataSrc: "data",
    processing: true,
    destroy: true,
  },
  {% load static %}
  columns: [
    {"data": "id"},
    {
      "data": "image_uris.small", "render": function (data, type, row) {
        return '';
      }
    },
    {"data": "name"},
    {"data": "oracle_text.substring(0,100)}",
    {"data": null, title: 'Action', wrap:true, "render": function(data, type, row, meta){
      var baseUrl = "<a href={% url 'product:finalize_new_product' 0 %} class='btn btn-success'>Add</button>";
      var data = baseUrl.replace('0', row.id);
      return data;
    }}
  ]

```

Per via del formato differente dei json ottenuti in base al set scelto non sempre è possibile caricare l'immagine nella tabella, ma ciò non è d'impatto sul corretto funzionamento dell'applicazione.

127.0.0.1:8000 dice

DataTables warning: table id=tableData - Requested unknown parameter 'oracle_text' for row 11, column 3. For more information about this error, please see <http://datatables.net/tn/4>

OK

Cart features:

Per la creazione del carrello sono stati definiti due modelli legati tra di loro:

-Order

-OrderItem

Order non è altro che il singolo ordine con attributo **“is_ordered = False”** del singolo utente nel quale sono salvati vari OrderItem, cioè i vari prodotti che verranno acquistati dall'utente. Il tutto viene gestito da chiamate ajax per le varie operazioni possibili (aggiunta prodotto a carrello, rimozione prodotto a carrello).

Una volta che l'utente decide di chiudere l'ordine e di fare il checkout l'attributo **“is_ordered”** verrà messo a True in maniera tale da poter effettuare altri ordini sempre da parte dello stesso utente. Quest'ultima operazione è obbligatoria, in quanto per trovare gli ordini in corso dell'utente viene utilizzato questo parametro.

```
def get_user_pending_order(request):  
    user = CustomUser.objects.get(username=request.user.username)  
    order = Order.objects.filter(user=user, is_ordered=False)
```

```

def checkout_view(request):
    order = get_user_pending_order(request)
    context = {'order': order}

    if request.method == 'POST':
        payment_form = PaymentForm(request.POST or None)
        if payment_form.is_valid():
            order_to_purchase = Order.objects.get(pk=order.id, is_ordered=False)
            order_to_purchase.shipping_address = payment_form.cleaned_data.get('shipping_address')
            order_to_purchase.shipping_country = payment_form.cleaned_data.get('shipping_country')
            order_to_purchase.credit_card_number = payment_form.cleaned_data.get('credit_card_number')
            order_to_purchase.cvv = payment_form.cleaned_data.get('cvv')
            order_to_purchase.is_ordered = True
            order_to_purchase.date_ordered = datetime.datetime.now()
            order_to_purchase.save()

            order_items = order_to_purchase.items.all()
            order_items.update(is_ordered=True, date_ordered=datetime.datetime.now())
            messages.info(request, "Thank you! Your purchase was successful!", extra_tags='alert-success '
                                                                                          'alert-dismissible')

            return redirect('user_management:profile')
        else:
            context['payment_form'] = payment_form
    else:
        context['payment_form'] = PaymentForm()
    return render(request, 'checkout.html', context)

```

Una volta confermato l'ordine, spetterà ai vari venditori la conferma della spedizione di questi tramite schermata presente nel proprio profilo che verrà salvata nell'attributo **"is_sent"** dei vari OrderItem.

Test:

All'interno del progetto sono stati creati dei test per la gestione degli utenti

```

class TestUserView(TestCase):

    def setUp(self):
        email = "test@email.com"
        username = "test8"
        first_name = "testin"
        last_name = "testina"
        password = "testtest"
        self.u = CustomUser.objects.create_user(email=email, username=username, first_name=first_name,
                                                last_name=last_name, password=password)

    def login_user(self):
        self.client.login(email='test@email.com', password="testtest")

    def test_login_redirect(self):
        self.login_user()
        response = self.client.get(reverse_lazy('user_management:login'))
        self.assertEqual(response.status_code, 200, "Authenticated user should return 200")

    def test_user_account_view_logged_in(self):
        self.login_user()
        response = self.client.get(reverse_lazy('user_management:profile'))
        self.assertEqual(response.status_code, 200, 'User is logged in successfully')

    def test_user_account_view_not_logged_in(self):
        response = self.client.get(reverse_lazy('user_management:profile'))
        self.assertEqual(response.status_code, 302, 'User not logged in should be redirected')

```

Viene creato un dummy account da poter usare durante i test e vengono testate le view di login.

Alcune schermate prese dal sito:

Market Of Cards

Website for the newly opened cards related e-commerce!



New Products

[Profile](#)[Settings](#)[Shop](#)[Orders](#)

Sales done:

Show 10 entries

Search:

Sale ID	Name	Price	Order Date	Status	Action
1	Erratic Visionary	77.11	Oct. 27, 2020, 3:42 p.m.	Sent	Send
3	Command Tower	5.00	Oct. 27, 2020, 3:04 p.m.	Sent	Send
4	Angelic Destiny	6.00	Oct. 27, 2020, 3:04 p.m.	Sent	Send
5	Seedborn Muse	14.00	Oct. 27, 2020, 2:49 p.m.	Sent	Send
8	Seedborn Muse	37.00	Oct. 27, 2020, 3:42 p.m.	Sent	Send
32	Command Tower	4.00	Oct. 27, 2020, 4:42 p.m.	Sent	Send
35	Seedborn Muse	70.00	Oct. 27, 2020, 4:42 p.m.	Sent	Send

Showing 1 to 7 of 7 entries

Previous [1](#) Next

Close your account

Are you sure you want to close your account?

This action is **irreversible!**

☐ I Agree

Close Account

Change your password

Old password*

New password*



- Your password can't be too similar to your other personal information.
- Your password must contain at least 8 characters.
- Your password can't be a commonly used password.
- Your password can't be entirely numeric.


New password confirmation*

Save changes

Search: an

Products found

ID	Photo	Name	Price	Quantity	Shop
7		Command Tower	1.00	5	Go to Shop
8		Angelic Destiny	2.00	324	Go to Shop

 admin
User Admin

[Profile](#)

[Settings](#)

[AdminBoard](#)

[Orders](#)

Users:

Show entries







Search:

Username	Email	Date Joined	Last Login	Type	Actions
	gab@gab.com	Oct. 5, 2020, 2:59 p.m.	Oct. 5, 2020, 2:59 p.m.	User	Ban User
aryel	aryel@aryel.com	Oct. 27, 2020, 2:01 p.m.	Oct. 27, 2020, 3:49 p.m.	Vendor	Ban User
emmara	emm@emm.com	Oct. 13, 2020, 3:21 p.m.	Oct. 27, 2020, 4:35 p.m.	Vendor	Ban User
gab	gac@gac.com	Oct. 6, 2020, 2:54 p.m.	Oct. 15, 2020, 1:51 p.m.	User	Ban User
gabri	gabri@gabri.com	Oct. 6, 2020, 4:19 p.m.	Oct. 27, 2020, 2 p.m.	User	Ban User
Gabriele2	gab@gabb.com	Oct. 6, 2020, 2:54 p.m.	Oct. 20, 2020, 2:43 p.m.	User	Ban User

Showing 1 to 6 of 6 entries

Previous Next

Cart

	Name	Price	Quantity	Action
	Erratic Visionary	22.00	4	
	Command Tower	1.00	1	
	Skyclave Cleric // Skyclave Basilica	1.00	1	

Order Total: \$24.00

Proceed To Checkout

Order Summary

Erratic Visionary	\$22.00
Command Tower	\$1.00
Skyclave Cleric // Skyclave Basilica	\$1.00
Order Total	\$24.00 + \$61.11 (Shipment)
	\$85.11

Checkout with a credit card

Payment Info

Shipping address*

Shipping country*

Credit card number*

Cvv*

Pay