UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Dipartimento di Scienze Fisiche, Informatiche e Matematiche

Corso di Laurea di Informatica

Tecnologie Web

Mondo NBA

Roberto Wang

Matricola 169487

Anno Accademico 2023-2024

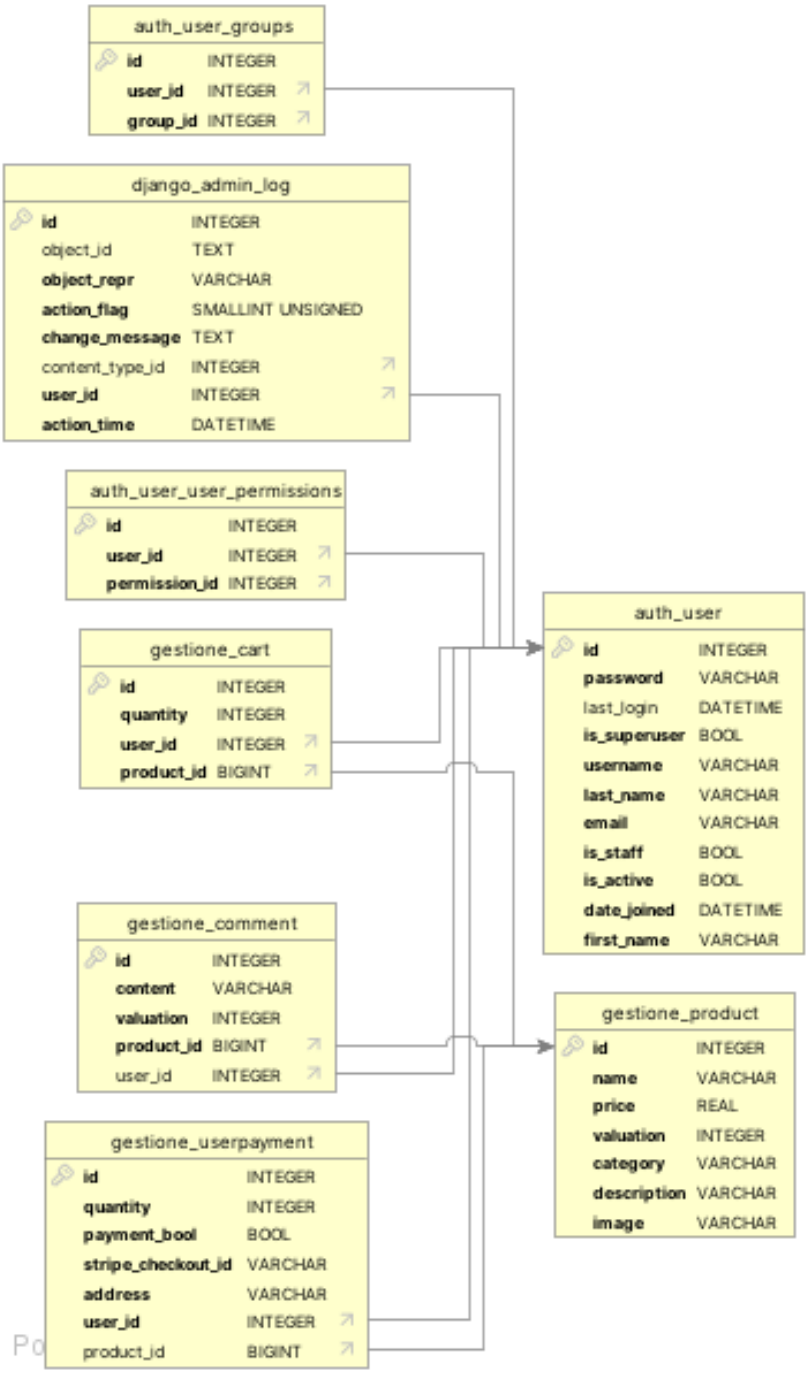
Sommario

Traccia

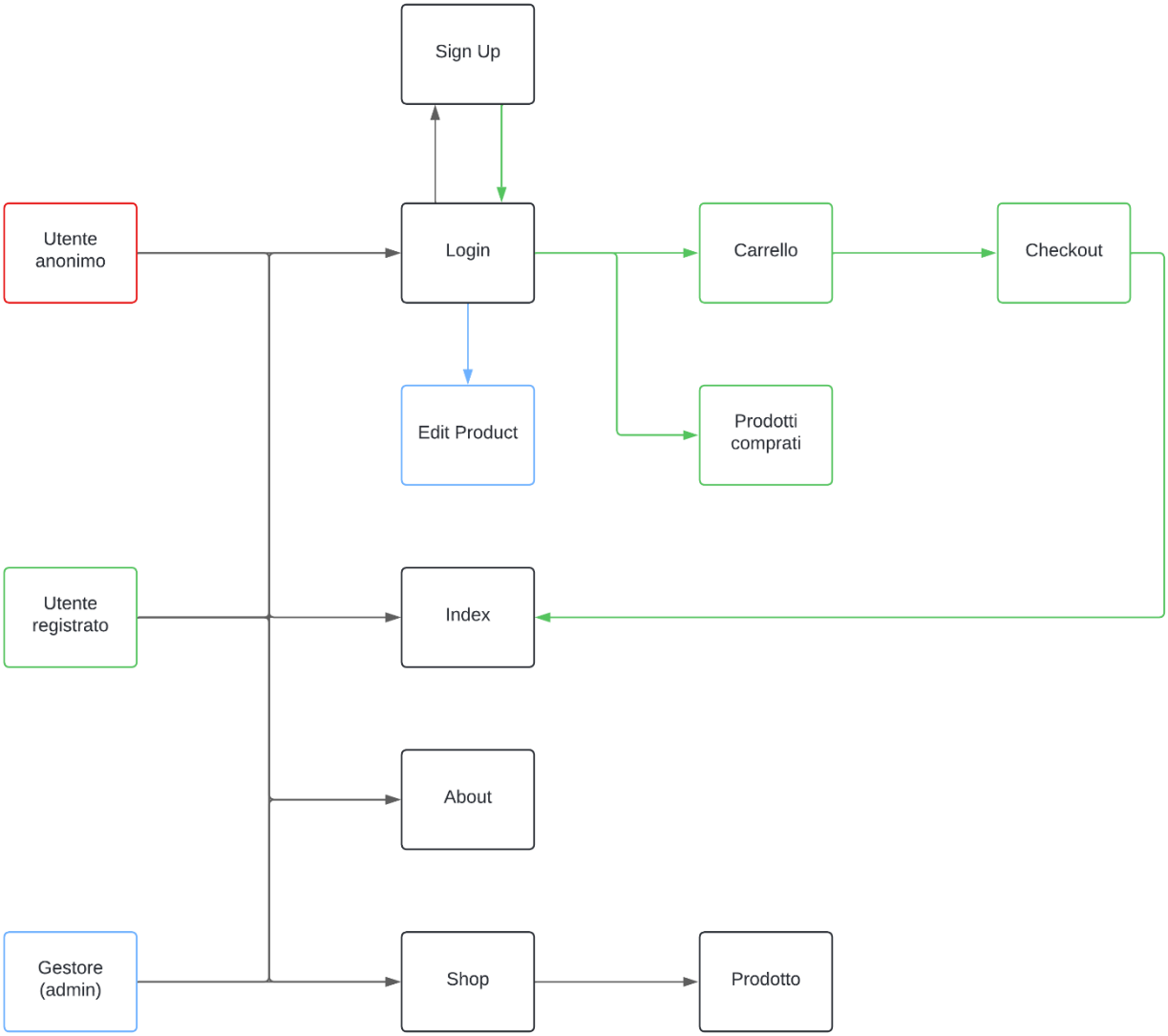
Realizzazione di uno store online attinente alla vendita di prodotti NBA, il quale prevede che:

1. Gli utenti non registrati possano:
    a. Visitare lo store, quindi vedere i prodotti disponibili con il relativo:
        a. Immagine
        b. Nome
        c. Prezzo
        d. Categoria
        e. Valutazione
        f. Recensioni degli utenti registrati
        g. Descrizione
    b. Ricercare dei prodotti in base al nome
        a. Ordinare i risultati in base alle categorie e/o al prezzo
2. Gli utenti registrati, oltre ad avere le stesse proprietà di quelli non registrati, possono:
    a. Acquistare uno o più prodotti, tramite l'utilizzo di un carrello
        i. I prodotti presenti nel carrello possono essere:
            1) Eliminati
            2) Modificati
        ii. Il pagamento avviene tramite carta di credito
    b. Lasciare una recensione sulla pagina del prodotto
3. Un unico utente gestore, il quale avrà la possibilità di:
    a. Creare un prodotto
    b. Eliminare un prodotto
    c. Modificare un prodotto
4. Un recommendation system, il quale mostrerà altri prodotti basandosi sul prezzo, nome e sugli utenti che hanno comprato lo stesso prodotto.

UML delle classi



Schema pagine



Tecnologie utilizzate

La realizzazione del progetto è stata effettuata mediante utilizzo di diverse tecnologie, ovvero:

1. Javascript

   Utilizzato principalmente per mostrare i messaggi di conferma dopo una determinata operazione dell'utente e per manipolare l'HTML DOM in modo che fosse dinamico l'interazione con l'utente.

2. Django

   Web Framework per interagire con il database, in maniera di ottenere i risultati e mostrarli all'utente attraverso l'utilizzo dei template. Quest'ultimi in base ai risultati ottenuti, possono variare il proprio contenuto. Inoltre, attraverso ai suoi strumenti è possibile ridurre la quantità di codice HTML che si ripete nelle varie pagine.

3. HTML

   Linguaggio che caratterizza le pagine che vengono mostrate agli utenti.

4. CSS

   Fogli di stili per personalizzare le pagine HTML.

5. Python

   Linguaggio necessario per l'utilizzo di Django ed i suoi strumenti.

6. SQLite

   Libreria software per l'implementazione del database, in cui risiedono i dati necessari per il corretto funzionamento delle pagine del sito.

7. Stripe

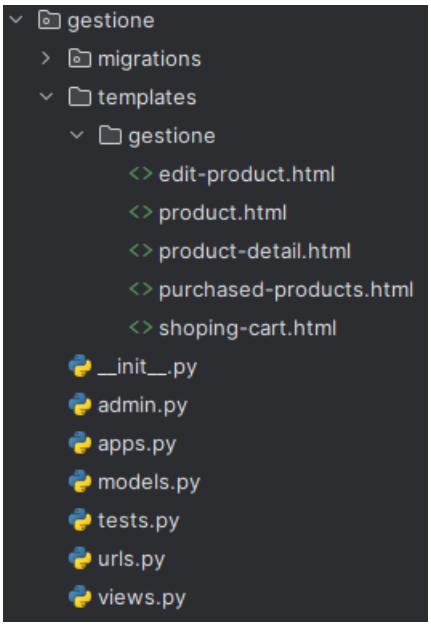
   Servizio utilizzato per quando l'utente registrato intendere ad acquistare uno o più prodotti. L'utente verrà reindirizzato ad una pagina dedicata per l'inserimento dei dati e per concludere l'acquisto.

Organizzazione del progetto

1. Suddivisione in app

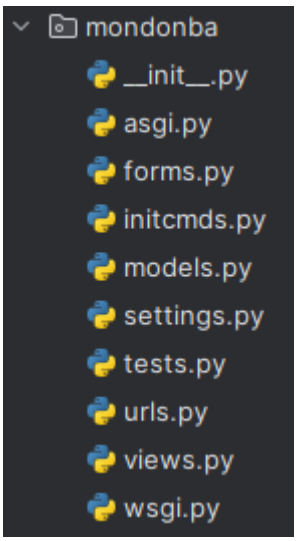
   1.1. gestione

   App per gestire i risultati che vengono mostrati nei vari templates (pagine) e per l'acquisto di uno o più prodotti (Stripe).



   1.2. mondonba

   Parte dell'applicazione che ha il compito di gestire principalmente l'autenticazione e registrazione da parte dell'utente.

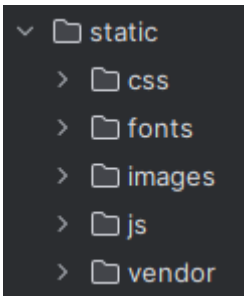


2. Templates, media e librerie

Per la personalizzazione del sito vi sono diverse cartelle:

2.1. static

Contiene tutti i file statici, i quali possono essere immagini ma anche fogli di stili (CSS) e librerie (javascript).

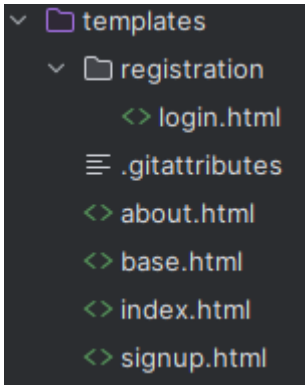


2.2. media

Risiedono le immagini inerenti ai prodotti caricate dal gestore del sito.

2.3. templates

Vi sono tutti i templates inerenti all'autenticazione e registrazione dell'utente, ma anche tutti quelli non inerenti al caricamento dei prodotti provenienti dal database.



Scelte implementative

1. In caso l'utente sia registrato, attraverso un "context_processors" personalizzato (presente nella cartella "utils"), sarà in grado di visualizzare tutti i suoi prodotti aggiunti al carrello tramite un'icona nella navbar.

```
from gestione.models import *
from django.contrib.auth.decorators import login_required
def getCart(request):
    if request.user.is_authenticated:
        cart = Cart.objects.all().filter(user=request.user)
        return {"cart": cart}
    return {}
```

2. Il sistema di raccomandazione dei prodotti è stato sviluppato in modo che vada a cercare prima i prodotti comprati degli altri utenti, i quali hanno acquistato il prodotto interessato, in caso il numero di prodotti consigliati raggiunge 8, si ferma, altrimenti continua andando a cercare i prodotti che hanno lo stesso prezzo, infine, se nuovamente la soglia dei prodotti consigliati non è stata raggiunta, va a controllare i prodotti con il nome simile.

```
# Recommendation System
listRaccomendationProducts = []

# All payment that has that product
tempUserPaymentProducts = UserPayment.objects.filter(product=tempProduct)
for userPaymentProduct in tempUserPaymentProducts:

    # All user that bought that product
    tempUser = userPaymentProduct.user
    if tempUser != self.request.user:
        tempUserPaymentUsers =
UserPayment.objects.filter(user=tempUser).exclude(product=tempProduct)

        # All other product that user bought
        for userPaymentUser in tempUserPaymentUsers:
            tempProduct2 = userPaymentUser.product
            print(tempProduct2)
            listRaccomendationProducts.append(tempProduct2)

if len(listRaccomendationProducts) < 8:

    tempProducts = Product.objects.filter(price=tempProduct.price)[:(8 -
len(listRaccomendationProducts))]

    for p in tempProducts:
        if p not in listRaccomendationProducts and p != tempProduct:
            print(p)
            listRaccomendationProducts.append(p)

listName = (tempProduct.name).split()
```

```
for name in listName:

    if len(listRaccomendationProducts) < 8:

        tempProducts = Product.objects.filter(name__icontains=name)[:(8 -
len(listRaccomendationProducts))]

        for p in tempProducts:
            if p not in listRaccomendationProducts and p != tempProduct:
                print(p)
                listRaccomendationProducts.append(p)
```

3. Durante il pagamento, attraverso il servizio “Stripe”, l’utente viene indirizzato in un’altra pagina, la quale non è presente nella struttura del sito. Per fare in modo che la pagina di pagamento accetti i dati si utilizza il decoratore “csrf_exempt”.

```
@csrf_exempt
def stripe_webhook(request):
    stripe.api_key = settings.STRIPE_SECRET_KEY_TEST
    time.sleep(10)
    payload = request.body
    signature_header = request.META['HTTP_STRIPE_SIGNATURE']
    event = None
    try:
        event = stripe.Webhook.construct_event(
            payload, signature_header, settings.STRIPE_WEBHOOK_SECRET_TEST
        )
    except ValueError as e:
        return HttpResponse(status=400)
    except stripe.error.SignatureVerificationError as e:
        return HttpResponse(status=400)

    if event['type'] == 'checkout.session.completed':
        session = event['data']['object']
        session_id = session.get('id', None)
        time.sleep(15)
        user_payment = UserPayment.objects.get(stripe_checkout_id=session_id)
        user_payment.payment_bool = True
        user_payment.save()
    return HttpResponse(status=200)
```

4. Ogni utente registrato può recensire un prodotto un’unica volta, in modo da non compromettere la valutazione di quest’ultimo.
5. Essendo che vi è solo un unico gestore, esso corrisponde al superuser del database. Nonostante ciò, dispone una pagina dedicata per interagire coi prodotti.

Test

Il progetto include i seguenti test:

1. Creazione di un prodotto e test di una sua funzione di calcolo

```
def test_product_create(self):
    name = "Test"
    price = "10.0"
    image = "product-36.jpg"
    category = "Clothing"
    description = "Description"
    valuation = "0"

    Product.objects.create(name=name, price=price, image=image,
category=category, description=description, valuation=valuation)
    product = Product.objects.get(name=name)

    self.assertIsNotNone(product)
    self.assertEqual(product.howManyEmptyStars(), range(0, 5))
```

2. Controllo raggiungibilità
   a. Controllare che il sito sia irraggiungibile per coloro non sono registrati
   b. Controllare che il sito sia reperibile per gli utenti registrati

```
def test_view_shoppingCart(self):
    username = "test"
    email = "test@gmail.com"
    password = "prova12345"

    User.objects.create_user(username=username, email=email,
password=password)

    c = Client()
    c.login(username="test", password="prova12345")

    response = c.get(reverse("gestione:shoppingCart"))
    self.assertEqual(response.status_code, 200, "Error")

    response = self.client.get(reverse("gestione:shoppingCart"))
    self.assertEqual(response.status_code, 302, "Error")
```

3. Simulazione per l'aggiunta di un prodotto nel carrello
   a. Iscrivere un utente attraverso i credenziali necessari
   b. Autenticare l'utente mediante i suoi dati
   c. Mostrare il catalogo dei prodotti all'utente
   d. Selezionare un prodotto
   e. Aggiungere il prodotto al carello
   f. Mostrare il carrello con i prodotti

```
def test_view_addCart(self):
    username = "test"
    email = "test@gmail.com"
    password = "prova12345"

    user = User.objects.create_user(username=username, email=email,
password=password)
```

```
    c = Client()
    c.login(username="test", password="prova12345")

    response = c.get(reverse("gestione:product"))
    self.assertEqual(response.status_code, 200, "Error")

    name = "Vince Carter Funko Pop"
    product = Product.objects.get(name=name)

    response = c.get(reverse("gestione:productDetail", kwargs={'pk':
product.id}))
    self.assertEqual(response.status_code, 200, "Error")

    data = {
        'num-product':1,
        'idProduct':product.id
    }

    response = c.post(reverse("gestione:addCart"), data, follow=True)
    self.assertEqual(response.status_code, 200, "Error")

    cart = Cart.objects.get(product=product, user=user)
    self.assertIsNotNone(cart, "Error")

    response = c.get(reverse("gestione:shoppingCart"))
    self.assertEqual(response.status_code, 200, "Error")
```