



UNIVERSITÀ DEGLI STUDI DI CATANIA
DIPARTIMENTO DI MATEMATICA E INFORMATICA
CORSO DI LAUREA MAGISTRALE IN INFORMATICA

Carmelo Marco Fuccio

Albums

RELAZIONE PROGETTO INGEGNERIA DEI
SISTEMI DISTRIBUITI

Prof. Emiliano Alessio Tramontana
Prof. Andrea Francesco Fornaia

Anno Accademico 20xx - 20xx

Abstract

Il progetto si basa sullo sviluppo di un sito utilizzando Django, sia nel front-end che nel back-end. L'applicazione è stata principalmente creata per i soggetti che vogliono ricevere una valutazione sui propri scatti. Viene data la possibilità di pubblicare le proprie foto che verranno poi valutate dagli utenti registrati nel sito. Sono presenti vari microservizi all'interno del sito che danno la possibilità all'utente di navigare e utilizzare il sito in sicurezza e comodità. Ognuno di quest'ultimi è gestito in modo automatico, come l'autenticazione, la pubblicazione, browsing, dashboard e altro.

Indice

1	Introduzione	4
2	Implementazione	6
2.1	Gestione Sign Up / Utenti	7
2.1.1	Implementazione Sign Up (Registrazione)	8
2.2	Gestione Log In	10
2.2.1	Implementazione Log In	10
2.3	Gestione Dashboard	12
2.3.1	Implementazione Dashboard	12
2.4	Gestione Admin	14
2.5	Gestione Item	15
2.5.1	Implementazione Items	16
2.6	Gestione Face Detection	19
2.6.1	Implementazione Face Detection	20
2.7	Gestione Detail e Rating	21
2.7.1	Implementazione Detail	22
2.7.2	Implementazione Rating	23
2.7.3	Implementazione Storing immagini	24
2.8	Gestione Sessions (Recently Viewed)	25
2.8.1	Implementazione Session	25
2.9	Gestione browsing	27
2.9.1	Implementazione browsing	28
3	Diagrammi UML	30
3.1	Gestione Core (Registrazione e Login)	30
3.1.1	Forms	30
3.1.2	Views	31
3.1.3	UML Finale	31
3.2	Gestione Items	32
3.2.1	Forms	32
3.2.2	Models	33

<i>INDICE</i>	3
3.2.3 Views	34
3.2.4 UML Finale	34
Conclusione	35
Bibliografia	36

Capitolo 1

Introduzione

Il progetto qui presentato rappresenta un connubio avvincente tra la complessità e la semplicità nell'implementazione di un sito web attraverso l'utilizzo del framework Django. Tale framework, basato sul linguaggio di programmazione Python, si è dimostrato una scelta intuitiva e accessibile per lo sviluppo del nostro progetto, permettendo una realizzazione agevole e allo stesso tempo robusta.

L'applicazione concepita si rivolge a una nicchia specifica di fotografi desiderosi di ottenere una valutazione dei propri scatti da parte di una community selezionata. L'obiettivo primario è stato creare un ambiente digitale attraente, dove gli utenti possano non solo ricevere feedback sulla propria creatività, ma anche esplorare immagini altrui per trovare fonti di ispirazione e condividere la loro passione comune.

Le funzionalità principali del sito si concentrano sulla registrazione degli utenti, l'accesso tramite login e la gestione fluida delle valutazioni e degli elementi visivi presenti. L'interfaccia utente è stata progettata con attenzione per offrire un'esperienza comoda, semplice e accogliente, creando un ambiente amichevole e invitante per la navigazione. La chiara disposizione delle sezioni facilita la fruizione delle varie funzionalità, contribuendo a rendere il sito un luogo piacevole in cui interagire.

Particolare attenzione è stata dedicata alla creazione di una community circoscritta e selezionata, che favorisce la condivisione e l'interazione tra gli utenti. Questo approccio mirato contribuisce a creare una cerchia ristretta di appassionati, consolidando la qualità delle interazioni e promuovendo un ambiente collaborativo e stimolante.

Il progetto, quindi, non si limita alla semplice creazione di uno spazio vir-

tuale per il rating fotografico, ma si propone come un punto d'incontro per gli amanti della fotografia, dove la qualità delle interazioni e la condivisione di idee e feedback contribuiscono a creare un'esperienza arricchente per tutti gli utenti coinvolti.

Capitolo 2

Implementazione

L'implementazione è stata trattata come un'unica applicazione che gestisce ogni microservizio e attività. Come pagina principale nel sito web è presente la seguente:

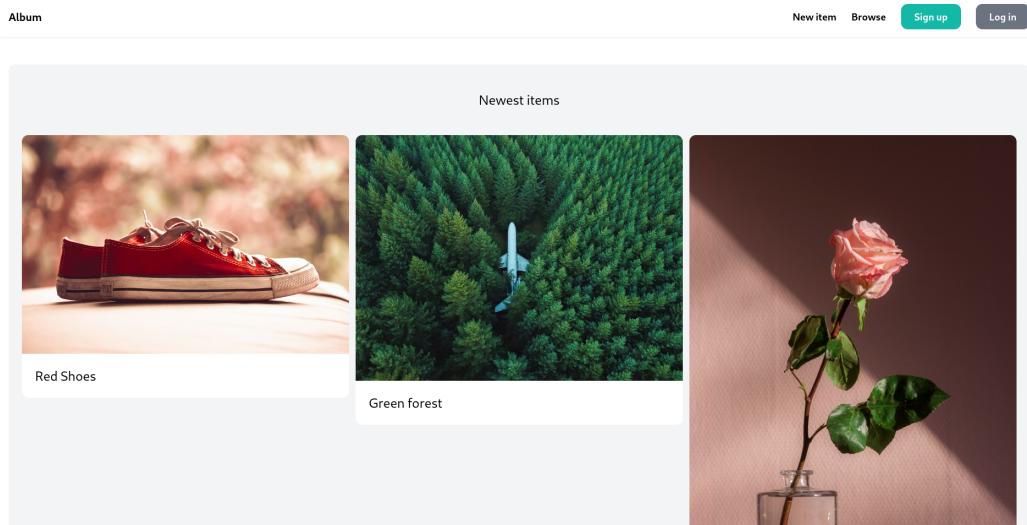


Figura 2.1: Homepage

All'interno dell'immagine è presente l'icona **Album** che dovrebbe rappresentare il logo/nome del sitoweb. Le successive sezioni verranno trattate di seguito

Nella sezione *new item* sono presenti le immagine pubblicate di recente da tutti gli utenti. Album è cliccabile, e reindirizza alla homepage, sarà presente in qualsiasi url all'interno del nostro sito per permettere una navigazione semplice e comoda.

Se l'utente non è autenticato e cercherà di navigare in una qualsiasi sezione

che non sia *Login* o *Signup* non potrà vedere il contenuto, gestito tramite un middleware.

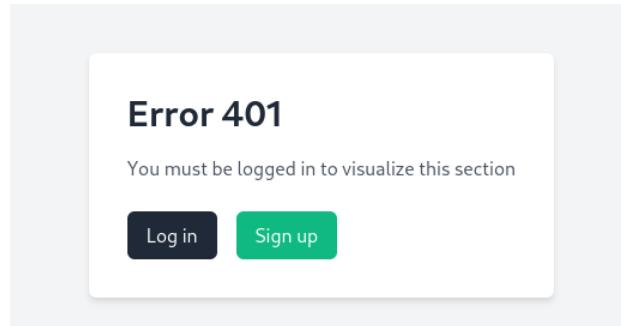


Figura 2.2: Sign Up

2.1 Gestione Sign Up / Utenti

Se l'utente non è registrato puo' loggarsi attraverso il pulsate **Sign Up** in verde. A seguito del click verrà reindirizzato alla seguente pagina:

A screenshot of a "Sign up" form. The form has a dark blue background with white text and input fields. At the top, the title "Sign up" is displayed in white. Below the title are four input fields: "Username" (placeholder "Your username"), "Email" (placeholder "Your email address"), "Password" (placeholder "Your password"), and "Repeat password" (placeholder "Repeat password"). At the bottom of the form is a large green "Submit" button.

Figura 2.3: Sign Up

2.1.1 Implementazione Sign Up (Registrazione)

Questa pagina è generata dalla funzione view, "signup" integrata. Essa gestisce la richiesta di registrazione.

```
def signup(request):
    if request.method == 'POST':
        form = SignupForm(request.POST)

        if form.is_valid():
            form.save()

            return redirect('/login/')
    else:
        form = SignupForm()

    return render(request, 'core/signup.html', {
        'form': form
    })
```

Figura 2.4: SignUp view

La classe illustrata utilizza la funzione per la creazione del form. Essa eredita la classe *UserCreationForm*” di **Django** per la gestione degli utenti.

```
class SignupForm(UserCreationForm):
    class Meta:
        model = User
        fields = ('username', 'email', 'password1', 'password2')

        username = forms.CharField(widget=forms.TextInput(attrs={
            'placeholder': 'Your username',
            'class': 'w-full py-4 px-6 rounded-xl'
        }))
        email = forms.CharField(widget=forms.EmailInput(attrs={
            'placeholder': 'Your email address',
            'class': 'w-full py-4 px-6 rounded-xl'
        }))
        password1 = forms.CharField(widget=forms.PasswordInput(attrs={
            'placeholder': 'Your password',
            'class': 'w-full py-4 px-6 rounded-xl'
        }))
        password2 = forms.CharField(widget=forms.PasswordInput(attrs={
            'placeholder': 'Repeat password',
            'class': 'w-full py-4 px-6 rounded-xl'
        }))
```

Figura 2.5: SignUp view

Crea un ”modello” per la creazione degli utenti, con i rispettivi parametri. E’ stata customizzata proprio perchè il form di Django riguarda solo *username* e *password*.

2.2 Gestione Log In

Nella fase di login l'utente accedere inserendo le credenziali utente e password (nel caso in cui l'utente sia già registrato).

The screenshot shows a 'Log in' form with a light gray background. At the top left is the text 'Log in'. Below it is a 'Username' field containing 'My Username'. Underneath is a 'Password' field with a redacted value. At the bottom is a teal-colored 'Submit' button.

Figura 2.6: SignUp view

Dopo che l'utente si è autenticato verrà reindirizzato alla homepage. Dove oltre al pulsante *Logout* saranno accessibili anche *New Item* e *Browse*.

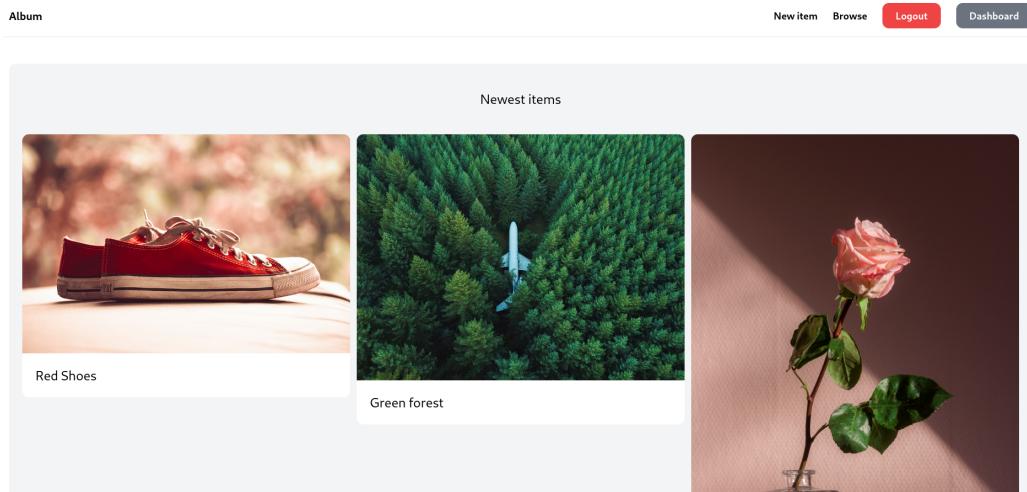


Figura 2.7: SignUp view

2.2.1 Implementazione Log In

Il login risulta leggermente più complicato per via dell'utilizzo di **jwt token**. (JSON Web Token) per consentire la persistenza dell'accesso dell'utente.

te. [1] Ci sono due principali vantaggi di un sistema di autenticazione basato su JWT: una migliore scalabilità/prestazioni e una diminuzione della complessità nel gestire sistemi distribuiti. Il token è sicuro mediante l'utilizzo del flag *httponly* che impedisce l'accesso da parte di script lato client e *secure* che consente la trasmissione solo con connessioni di tipo HTTPS. Dopo che il

```
def login_view(request):
    template_name = 'core/login.html'
    authentication_form = AuthenticationForm

    if request.method == 'GET':
        form = authentication_form(request)
        return render(request, template_name, {'form': form})

    elif request.method == 'POST':
        form = authentication_form(request, data=request.POST)

        if form.is_valid():
            login(request, form.get_user())
            user = form.get_user()
            expiration_time = datetime.utcnow() + timedelta(days=1)

            payload = {
                'username': user.username,
                'exp': expiration_time,
            }

            jwt_token = jwt.encode(payload, '259fabc6e7b379d1babad0eb3b8ed8a14c3ccfed5acf7d93c81f1add36f7626f', algorithm='HS256')
            response = redirect('core:index')
            response.set_cookie('jwt_token', jwt_token, httponly=True, secure=True)

            print(request.COOKIES.get('jwt_token'))
            return response

    return render(request, template_name, {'form': form})
```

Figura 2.8: Login view

form di autenticazione è valido, viene creato un payload JWT con all'interno informazioni come *username* e *expiration time*. Il payload viene codificato e successivamente la risposta viene aggiunta come un cookie.

Infine è stata estesa la classe form.

```
class LoginForm(AuthenticationForm):
    username = forms.CharField(widget=forms.TextInput(attrs={
        'placeholder': 'Your username',
        'class': 'w-full py-4 px-6 rounded-xl'
    }))
    password = forms.CharField(widget=forms.PasswordInput(attrs={
        'placeholder': 'Your password',
        'class': 'w-full py-4 px-6 rounded-xl'
    }))
```

Figura 2.9: Login view

2.3 Gestione Dashboard

All'interno della sezione *Dashboard* sarà presente una sorta di "Profilo" per l'utente loggato.

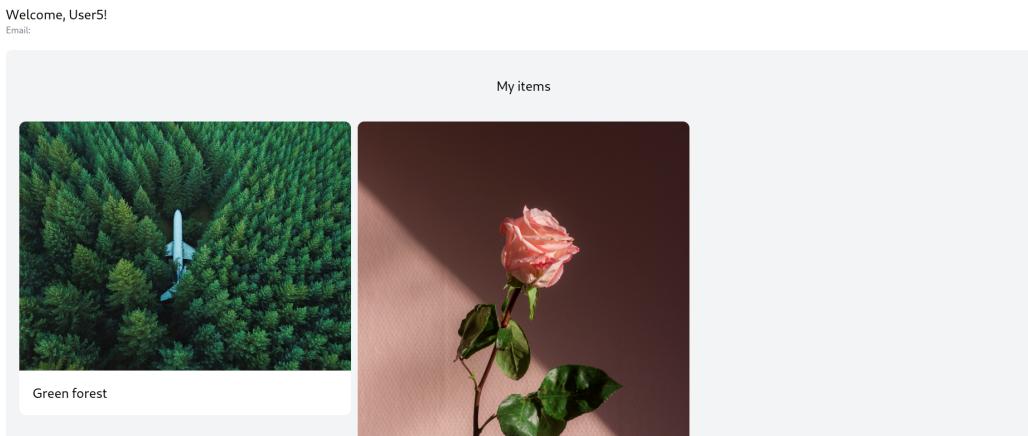


Figura 2.10: Login view

All'interno di essa saranno mostrati solo il nome dell'account e la propria e-mail i propri scatti pubblicati.

2.3.1 Implementazione Dashboard

Niente di complicato, vengono presi gli item correlati all'utente e mostrati in html.

```
@login_required
def index(request):
    items = Item.objects.filter(created_by=request.user)

    return render(request, 'visuals/index.html', {
        'items': items,
    })
```

Figura 2.11: Dashboard view

Il codice html utilizzato per la mostra dei propri oggetti, accessibile anche ai dettagli ad ognuna di esse. Vedremo successivamente nel dettaglio in cosa consiste.

```
{% extends 'core/base.html' %}

{% block title %}Dashboard{% endblock %}

{% block content %}


<h2 class="text-2xl">Welcome, {{ user.username }}!</h2>
    <p class="text-gray-500">Email: {{ user.email }}</p>


<div class="mt-6 px-6 py-12 bg-gray-100 rounded-xl">
    <h2 class="mb-12 text-2xl text-center">My items</h2>

    <div class="grid grid-cols-3 gap-3">
        {% for item in items %}
            <div>
                <a href="{% url 'item:detail' item.id %}">
                    <div>
                        
                    </div>

                    <div class="p-6 bg-white rounded-b-xl">
                        <h2 class="text-2xl">{{ item.name }}</h2>
                    </div>
                </a>
            </div>
        {% endfor %}
    </div>
</div>
{% endblock %}
```

Figura 2.12: Dashboard html

2.4 Gestione Admin

La gestione dell'admin è implementata da Django che permette di accedere al database e poter controllare e modificare qualsiasi tabella. Quando viene creato il progetto **Django** da console vi è la possibilità di creare un *superuser* inserendo nickname, email(facoltativo), e password.

Andando all'URL *localhost:8000/admin* accederemo alla pagina di login.

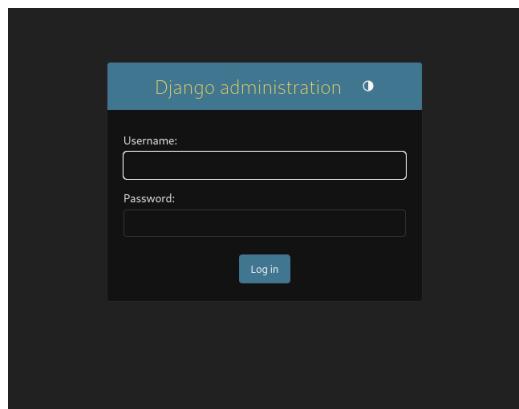


Figura 2.13: Enter Caption

Nella figura 2.14 sono presenti le varie tabelle all'interno del database.

- Categories
- Items
- Ratings

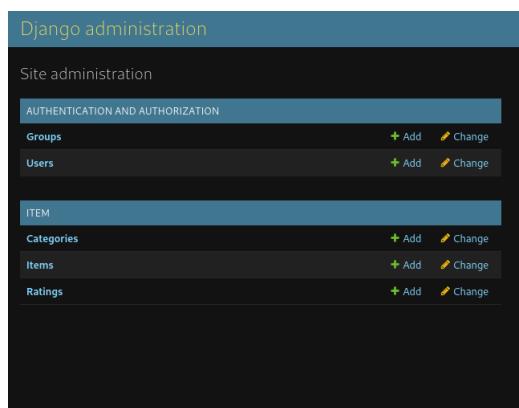


Figura 2.14: Enter Caption

2.5 Gestione Item

Come mostrato in precedenza gli item (gli scatti) vengono mostrati nella vetrina in homepage, esponendo principalmente gli ultimi 6 oggetti creati all'interno del sito.

Per effettuare una pubblicazione, andremo nella sezione *New Item*, presente nella nostra *Navigation Bar*.

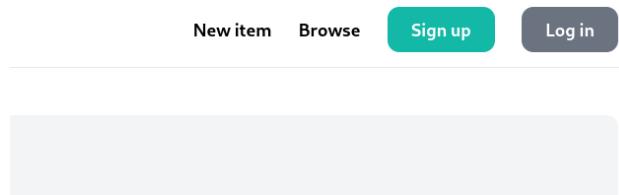


Figura 2.15: Navigation bar

Quando cliccheremo sul pulsante precedentemente citato, verremo reindirizzati nella pagina di "pubblicazione" della nostra immagine.

 A screenshot of an "Item form" titled "New item". The form contains several input fields: "Category" (a dropdown menu), "Name" (a text input field), "Description" (a text area), and "Image" (a file upload field with "Choose File" and "No file chosen" labels). At the bottom is a "Submit" button.

Figura 2.16: Item form

All'interno di questa pagina avremo i seguenti campi.

- **Category:** Assegnare una categoria associata (come meglio crediamo) all'immagine da pubblicare.
- **Name:** Il nome/titolo della foto pubblicata
- **Description:** La possibilità (facoltativa) di aggiungere una descrizione inerente allo scatto.

- **Image:** Inserire un file immagine (Apparte un’eccezione che sveleremo dopo).

Quando l’oggetto verrà creato l’utente sarà reindirizzato alla pagina dell’oggetto che conterrà i dettagli.

Infine quando si crea si pubblica un oggetto oltre a mostrare la pagina con tutti i dettagli sarà possibile eventualmente di rimuoverlo dal pulsante *Delete*.

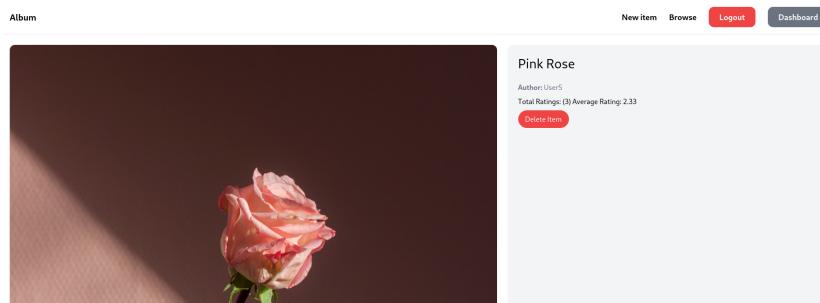


Figura 2.17: Item form

2.5.1 Implementazione Items

Nel nostro sistema, abbiamo definito tre modelli chiave per gestire la struttura dei dati: **Category**, **Item**, e **Rating**.

```
class Category(models.Model):
    name = models.CharField(max_length=255)

    class Meta:
        ordering = ('name',)
        verbose_name_plural = 'Categories'

    def __str__(self):
        return self.name

class Item(models.Model):
    category = models.ForeignKey(Category, related_name='items', on_delete=models.CASCADE)
    name = models.CharField(max_length=255)
    description = models.TextField(blank=True, null=True)
    image = models.ImageField(upload_to='item_images', blank=True, null=True)
    created_by = models.ForeignKey(User, related_name='items', on_delete=models.CASCADE)
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.name

class Rating(models.Model):
    item = models.ForeignKey(Item, on_delete = models.CASCADE)
    value = models.IntegerField()
    rated_by = models.ForeignKey(User, on_delete=models.CASCADE)
```

Figura 2.18: Enter Caption

La classe **Category** rappresenta le diverse categorie di elementi disponibili, garantendo un’organizzazione chiara e ordinata.

Il modello Item è responsabile della rappresentazione di ogni singolo elemento, inclusi dettagli come nome, descrizione e immagine.

Inoltre, il modello Rating è stato implementato per gestire le valutazioni degli utenti sugli elementi.

Nella figura 2.16 vediamo la view che serve per la visualizzazione delle varie immagini all'interno del nostro sito web.

```
def items(request):
    query = request.GET.get('query', '')
    category_id = request.GET.get('category', 0)
    categories = Category.objects.all()
    items = Item.objects.filter()

    if category_id:
        items = items.filter(category_id=category_id)

    if query:
        items = items.filter(Q(name__icontains=query) | Q(description__icontains=query))

    return render(request, 'item/items.html', {
        'items': items,
        'query': query,
        'categories': categories,
        'category_id': int(category_id)
    })
```

Figura 2.19: New item Form

Le immagini verranno visualizzate utilizzando la seguente view nella figura 2.17, dove vengono fatti ulteriori controlli che analizzeremo dopo.

```
def new(request):
    if request.method == 'POST':
        form = NewItemForm(request.POST, request.FILES)

        if form.is_valid():
            item = form.save(commit=False)

            # Check if the uploaded image contains a face
            image = request.FILES['image'].read()
            numpy_array = np.frombuffer(image, np.uint8)
            cv_image = cv2.imdecode(numpy_array, cv2.IMREAD_COLOR)
            gray_image = cv2.cvtColor(cv_image, cv2.COLOR_BGR2GRAY)

            face_classifier = cv2.CascadeClassifier(
                cv2.data.haarcascades + "haarcascade_frontalface_default.xml"
            )
            faces = face_classifier.detectMultiScale(gray_image, scaleFactor=1.1, minNeighbors=5, minSize=(48, 48))

            if len(faces) > 0:
                message = 'Face detected in the uploaded image. You cannot upload images with faces.'
                return render(request, 'item/face_detection_warning.html', {
                    'warning_message': message,
                })
            else:
                # No faces detected, save the item
                item.created_by = request.user
                item.save()
                return redirect('item:detail', pk=item.id)
        else:
            form = NewItemForm()

    return render(request, 'item/form.html', {
        'form': form,
        'title': 'New item',
    })
```

Figura 2.20: Enter Caption

Per la compilazione viene utilizzato il form illustrato nella figura 2.18

```
@login_required
def delete(request, pk):
    item = get_object_or_404(Item, pk=pk, created_by=request.user)
    item.delete()
    return redirect('visuals:index')
```

Figura 2.22: Enter Caption

```
INPUT_CLASSES = 'w-full py-4 px-6 rounded-xl border'

class NewItemForm(forms.ModelForm):
    class Meta:
        model = Item
        fields = ('category', 'name', 'description', 'image')
        widgets = {
            'category': forms.Select(attrs={
                'class': INPUT_CLASSES
            }),
            'name': forms.TextInput(attrs={
                'class': INPUT_CLASSES
            }),
            'description': forms.Textarea(attrs={
                'class': INPUT_CLASSES
            }),
            'image': forms.FileInput(attrs={
                'class': INPUT_CLASSES
            })
        }
```

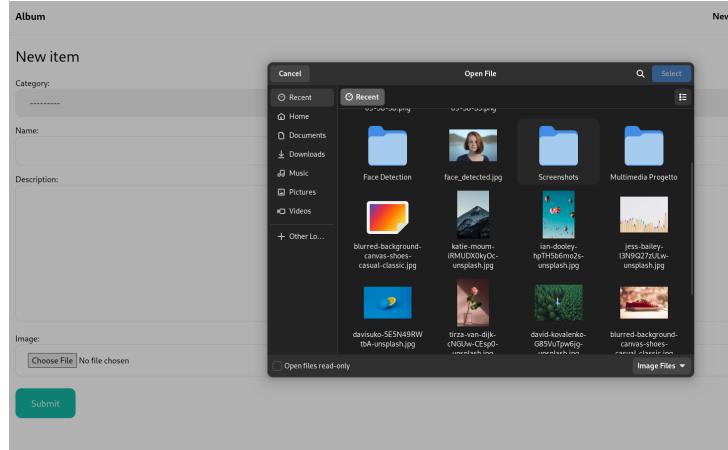
Figura 2.21: Enter Caption

Infine abbiamo la view per rimuovere l'oggetto.

2.6 Gestione Face Detection

[2] Per la psicologia, almeno dagli anni '50 e '60, i primi studi sul riconoscimento facciale possono essere rintracciati nella letteratura ingegneristica. Alcuni dei primi risultati includono esperimenti sulle teorie di Darwin riguardanti le espressioni facciali. Con la piattaforma open source chiamata OpenCV di Intel

E' stato creato un controllo per evitare l'utilizzo di volti come immagini attraverso un algoritmo di machine learning pretrainato.

**Figura 2.23:** Face detect

Come mostrato nella figura 2.19 effettueremo un caricamento di un volto, successivamente appena cliccheremo il pulsante "Submit" verremo reindirizzati alla successiva pagina (Figura 2.20). L'immagine non verrà né salvata in locale, ne pubblicata.

Warning

Face detected in the uploaded image. You cannot upload images with faces.

[Go back](#)

Figura 2.24: Sign Up

2.6.1 Implementazione Face Detection

In questo frammento di codice, viene implementato un algoritmo di machine learning per il rilevamento del viso utilizzando OpenCV. L'immagine caricata viene processata convertendola in scala di grigi e utilizzando un classificatore Haar Cascade pre-addestrato per rilevare i volti. Se vengono rilevati volti, viene visualizzato un messaggio di avvertimento, impedendo il salvataggio dell'immagine.

```

if form.is_valid():
    item = form.save(commit=False)

    # Check if the uploaded image contains a face
    image = request.FILES['image'].read()
    numpy_array = np.frombuffer(image, np.uint8)
    cv_image = cv2.imdecode(numpy_array, cv2.IMREAD_COLOR)
    gray_image = cv2.cvtColor(cv_image, cv2.COLOR_BGR2GRAY)

    Face_classifier = cv2.CascadeClassifier(
        cv2.data.haarcascades + "haarcascade_frontalface_default.xml"
    )
    faces = Face_classifier.detectMultiScale(gray_image, scaleFactor=1.1, minNeighbors=5, minSize=(40, 40))

    if len(faces) > 0:
        message = 'Face detected in the uploaded image. You cannot upload images with faces.'
        return render(request, 'item/Face_detection_warning.html', {
            'warning_message': message,
        })

```

Figura 2.25: Enter Caption

2.7 Gestione Detail e Rating

Questa sezione è dedicata ai dettagli per la visualizzazione dell’immagine. Vengono mostrate i seguenti campi :

- **Author:** L’autore dell’immagine.
- **Total Ratings:** Il numero totale di voti ricevuti.
- **Average Ratings:** La media dei voti ricevuti.
- **Rating form:** Il form per inviare la propria valutazione all’immagine.

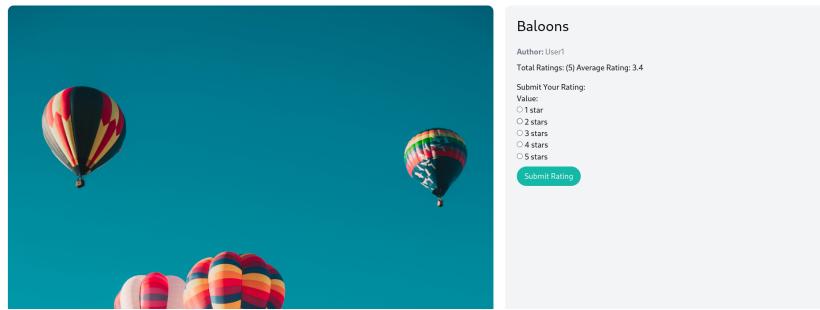


Figura 2.26: Item Form

Se l’utente non è l’autore dell’immagine potrà effettuare un rating da 1 a 5 ”stelle”, altrimenti non potrà autovotare la propria immagine.

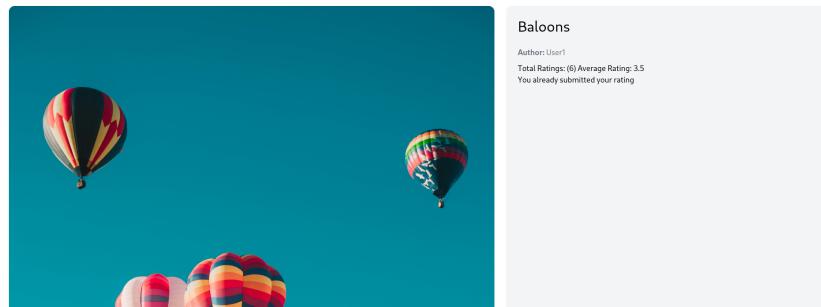


Figura 2.27: Item Form

Dopo che l'utente ha votato la pagina si aggiornerà mostrando il messaggio *"You already submitted your rating"* e aggiornerà anche il valore del voto complessivo.

2.7.1 Implementazione Detail

Nella view detail è presente molto materiale anche perchè sono presenti alcune variabili, soprattutto per la gestione del rating. Quando andiamo a cliccare su un'immagine sono presenti il *rating*, gli *item correlati*, il *form*, l'*average* del rating e altro.

```

def detail(request, pk):
    item = get_object_or_404(Item, pk=pk)
    related_items = Item.objects.filter(category=item.category).exclude(pk=pk)[:3]
    ratings = Rating.objects.filter(item=item)

    recently_viewed(request, pk)
    if ratings:
        average_rating = round(sum(rating.value for rating in ratings) / len(ratings), 2)
    else:
        average_rating = None

    user_rating = Rating.objects.filter(item=item, rated_by=request.user).first()

    if request.method == 'POST':
        print('Entered in the POST method')
        rating_form=RatingForm(request.POST)

        if rating_form.is_valid():
            rating = rating_form.save(commit=False)
            rating.item = item
            rating.rated_by = request.user
            form_submitted = True
            rating.save()
        else:
            rating_form = RatingForm()
            form_submitted = False

    return render(request, 'item/detail.html', {
        'item': item,
        'related_items': related_items,
        'ratings': ratings,
        'average_rating': average_rating,
        'rating_form': rating_form,
        'user_rating': user_rating,
        'form_submitted': form_submitted,
    })

```

Figura 2.28: Enter Caption

2.7.2 Implementazione Rating

L'apposito form per l'invio dei vari rating utente.

```

class RatingForm(forms.ModelForm):
    class Meta:
        model = Rating
        fields = ['value']
        widgets = {
            'value': forms.RadioSelect(choices=[(1, '1 star'), (2, '2 stars'), (3, '3 stars'), (4, '4 stars'), (5, '5 stars')]),,
        }

```

Figura 2.29: New item Form

All'interno della funzione *detail* è presente il controllo e l'invio del form *rating*.

```
if request.method == 'POST':
    print('Entered in the POST method')
    rating_form=RatingForm(request.POST)

    if rating_form.is_valid():
        rating = rating_form.save(commit=False)
        rating.item = item
        rating.rated_by = request.user
        form_submitted = True
        rating.save()
    else:
        rating_form = RatingForm()
        form_submitted = False
```

Figura 2.30: Enter Caption

2.7.3 Implementazione Storing immagini

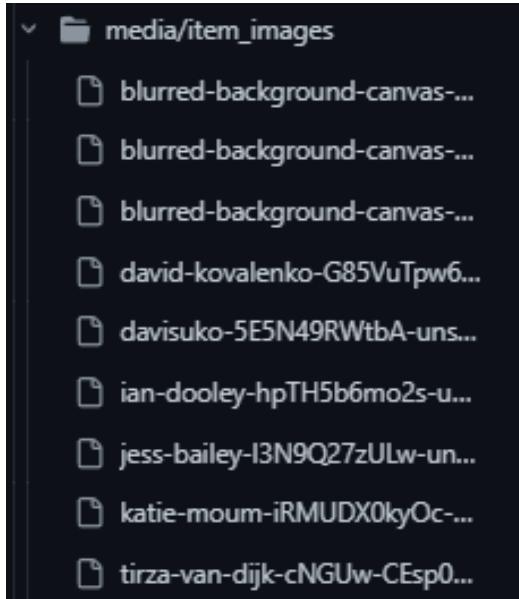


Figura 2.31: Enter Caption

Le immagini vengono immagazzinate in locale, in un'apposita directory nella nostra macchina, impostazione aggiunta in *settings* di **Django**.

```
MEDIA_URL = 'media/'  
MEDIA_ROOT = BASE_DIR / 'media'
```

Figura 2.32: Enter Caption

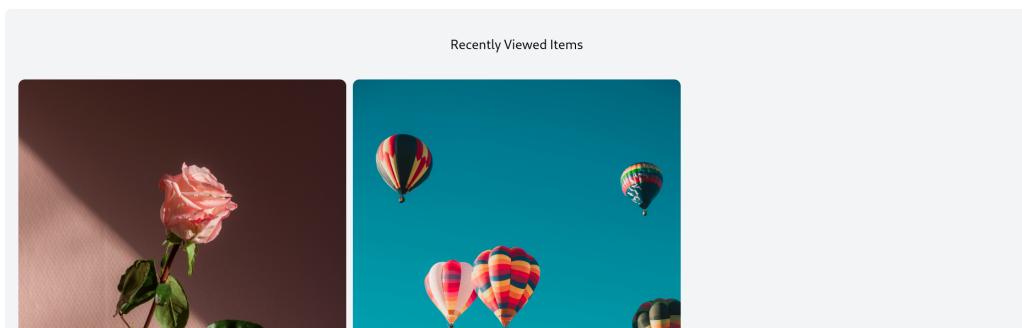
Nel form per la creazione di un nuovo oggetto viene utilizzato `forms.FileInput`, esso si traduce in un campo di input HTML di tipo "file". Questo consente agli utenti di selezionare e caricare file dal proprio dispositivo.

```
'image': forms.FileInput(attrs={  
    'class': INPUT_CLASSES  
})  
}
```

Figura 2.33: Enter Caption

2.8 Gestione Sessions (Recently Viewed)

Quando un utente mette le credenziali e accede al sito, esso avrà un ulteriore sezione nella home chiamata **Recently Viewed** dove saranno presenti tutte le immagini/scatti che ha guardato di recente fino ad un massimo di 3.

**Figura 2.34:** Enter Caption

Se si esce dal sito, se si naviga in altre pagine la *Recently Viewed* rimane intatta nella sessione.

2.8.1 Implementazione Session

Quando viene aperta la view `detail` viene richiamata la funzione `recently viewed` che farà l'append utilizzando la primary key alla sessione.

```
% if request.user.is_authenticated %}
<div class="mt-6 px-6 py-12 bg-gray-100 rounded-xl">
  <h2 class="mb-12 text-2xl text-center">Recently Viewed Items</h2>
  <div class="grid grid-cols-3 gap-3">
    {% for item in recently_viewed %}
      <div>
        <a href="{% url 'item:detail' item.id %}">
          <div>
            
          </div>
        <div class="p-6 bg-white rounded-b-xl">
          <h2 class="text-2xl">{{ item.name }}</h2>
        </div>
      </div>
    {% endfor %}
  </div>
{% endif %}
{% endblock %}
```

Figura 2.37: Enter Caption

```
def recently_viewed( request, pk ):
    print('Entered in the recently_viewed')

    if not "recently_viewed" in request.session:
        request.session["recently_viewed"] = []
        request.session["recently_viewed"].append(pk)
    else:
        if pk in request.session["recently_viewed"]:
            request.session["recently_viewed"].remove(pk)
            request.session["recently_viewed"].insert(0, pk)
        if len(request.session["recently_viewed"]) > 5:
            request.session["recently_viewed"].pop()
    request.session.modified = True

def detail(request, pk):
    item = get_object_or_404(Item, pk=pk)
    related_items = Item.objects.filter(category=item.category).exclude(pk=pk)[:3]
    ratings = Rating.objects.filter(item=item)

    recently_viewed(request,pk)
```

Figura 2.35: Enter Caption

Successivamente si controlla all'interno dell'app *core*, dove viene gestita principalmente la homepage e le view di registrazione, autenticazione, login e logout, se sono presenti degli oggetti all'interno della sessione.

```
def index(request):
    items = Item.objects.all()[:6]
    categories = Category.objects.all()

    recently_viewed_qs = Item.objects.filter(pk__in=request.session.get("recently_viewed", []))[:3]

    return render(request, 'core/index.html', {
        'categories': categories,
        'items': items,
        'recently_viewed': recently_viewed_qs
    })
```

Figura 2.36: Enter Caption

Verranno recuperati dalla tabella degli oggetti Item utilizzando i loro identificatori primari.

2.9 Gestione browsing

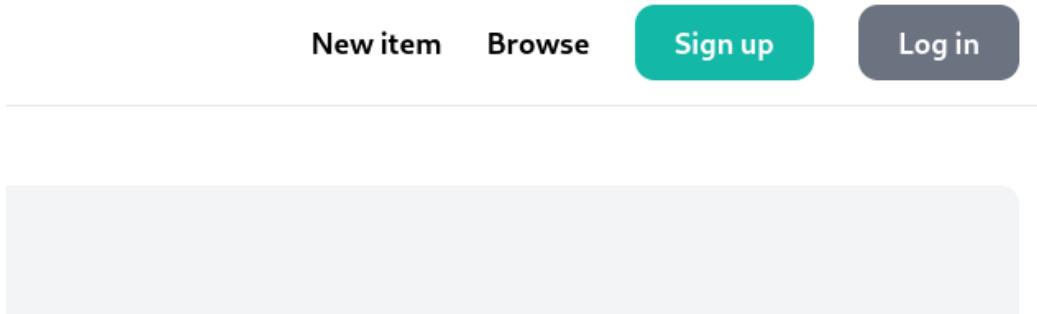


Figura 2.38: Enter Caption

Nella figura 2.38 è presente il pulsante **Browse** che ci reindirizzerà alla pagina di ricerca

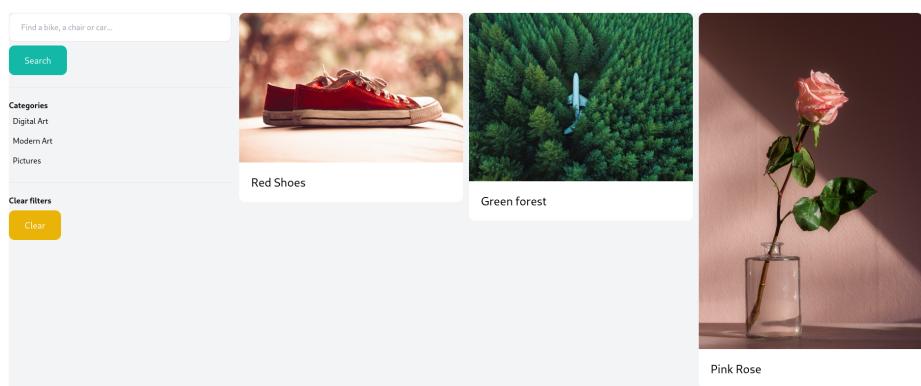


Figura 2.39: Enter Caption

L’interfaccia che ci si presenta mostrerà tutti gli oggetti presenti nel sito utilizzando lo scorrimento verso il basso. Alla destra è presente la barra di ricerca che permette di cercare, attraverso l’utilizzo dei filtri e di una *barra di ricerca*, le foto che più ci interessano per categorie o attraverso la ricerca del titolo dell’immagine.

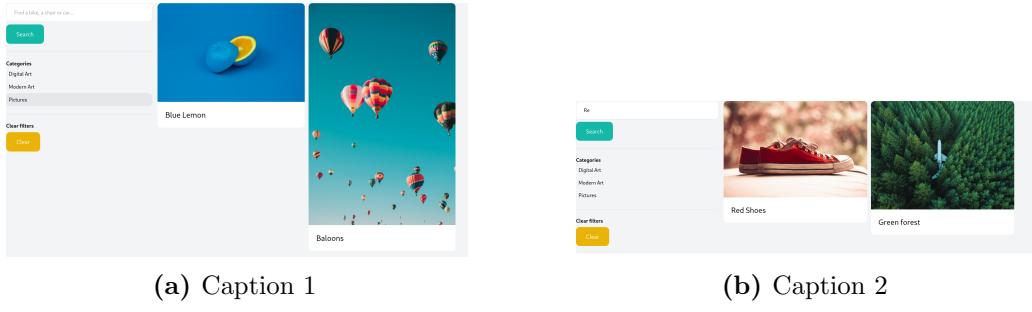


Figura 2.40: Main caption

2.9.1 Implementazione browsing

Il principale sviluppo del *browsing* è stato fatto in html. [3]Django fornisce molto più supporto per le applicazioni Web rispetto alle vecchie librerie CGI. Attraverso un sistema di templating, si possono separare la logica dell'applicazione dall'interfaccia utente implementata in HTML. Gli utenti possono filtrare gli articoli per categoria e cancellare i filtri applicati. Ogni articolo è visualizzato con un'immagine e un link alla pagina dettagliata.

```

<div class="pb-6 grid grid-cols-4 gap-4 bg-gray-100">
  <div class="col-span-1">
    <form method="get" action="{% url 'item:items' %}">
      <input name="query" class="w-full py-4 px-6 border rounded-xl" type="text" value="{{ query }}" placeholder="Find a bike, a chair or car...">
      <button class="mt-2 py-4 px-8 text-lg bg-teal-500 text-white rounded-xl">Search</button>
    </form>

    <hr class="my-6">

    <p class="font-semibold">Categories</p>

    <ul>
      {% for category in categories %}
        <li class="py-2 px-2 rounded-xl{% if category.id == category_id %} bg-gray-200{% endif %}">
          <a href="{% url 'item:items' %}?query={{ query }}&category={{ category.id }}">{{ category.name }}{{ item.name }}
            </div>
          </a>
        </div>
      {% endfor %}
    </div>
  </div>

```

Figura 2.41: Enter Caption

Questa parte del codice HTML rappresenta un ciclo che itera attraverso una lista di categorie. Ogni categoria viene presentata come un elemento di lista ($\langle li \rangle$) all'interno di un elenco non ordinato.

Per ogni categoria, viene verificato se l'identificatore della categoria corrente corrisponde all'identificatore della categoria selezionata. Se corrisponde, viene applicata una formattazione visiva specifica alla categoria per evidenziarla.

All'interno di ciascun elemento di lista, viene incluso un link ($\langle a \rangle$) che reindirizza gli utenti a una pagina che mostra gli articoli correlati a quella categoria. Il testo del link corrisponde al nome della categoria.

Capitolo 3

Diagrammi UML

In questa sezione verranno mostrati gli UML in merito alle classi utilizzate per le due "applicazioni" utilizzate in **Django**.

- Core
- Item

3.1 Gestione Core (Registrazione e Login)

Nell'applicazione *Core* sono state implementate le classi per il Login, Logout e Registrazione.

3.1.1 Forms

Questi sono i forms utilizzati per l'autenticazione e per il Login.

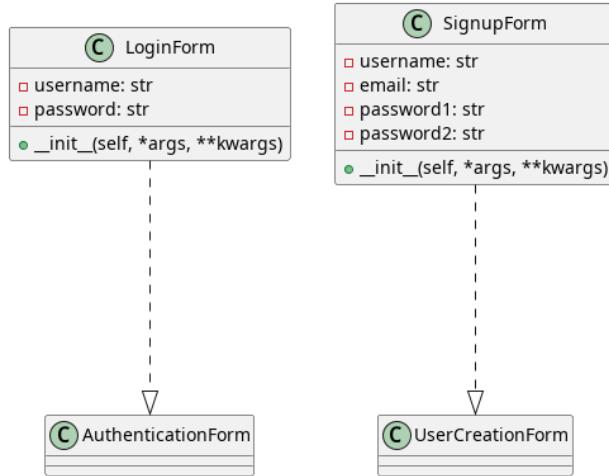


Figura 3.1

3.1.2 Views

I vari metodi di comunicazione per la visualizzazione e la gestione dell'app.

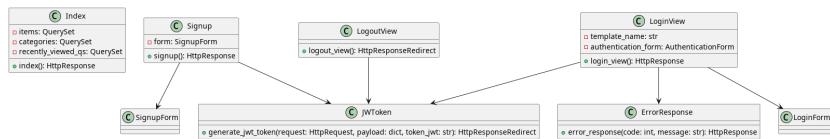


Figura 3.2

3.1.3 UML Finale

L'intera comunicazione dell'applicazione *Core*.

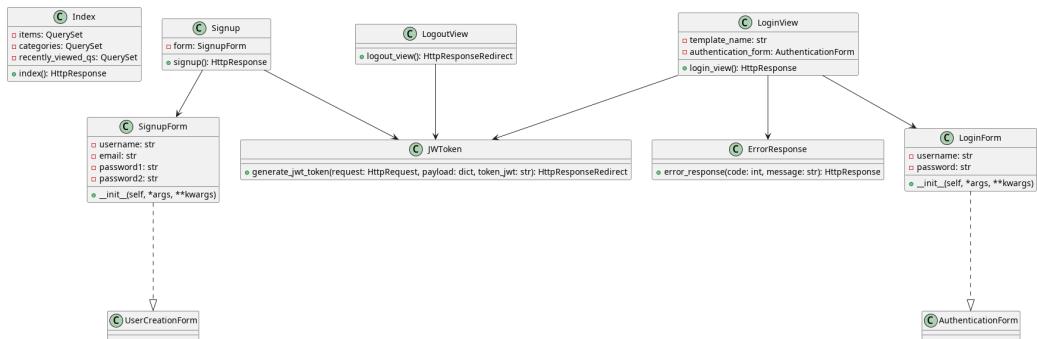


Figura 3.3

3.2 Gestione Items

La gestione degli item all'interno della nostra applicazione in tutti gli aspetti.

3.2.1 Forms

Sono stati utilizzati dei form particolari per la creazione e per l'invio del rating.

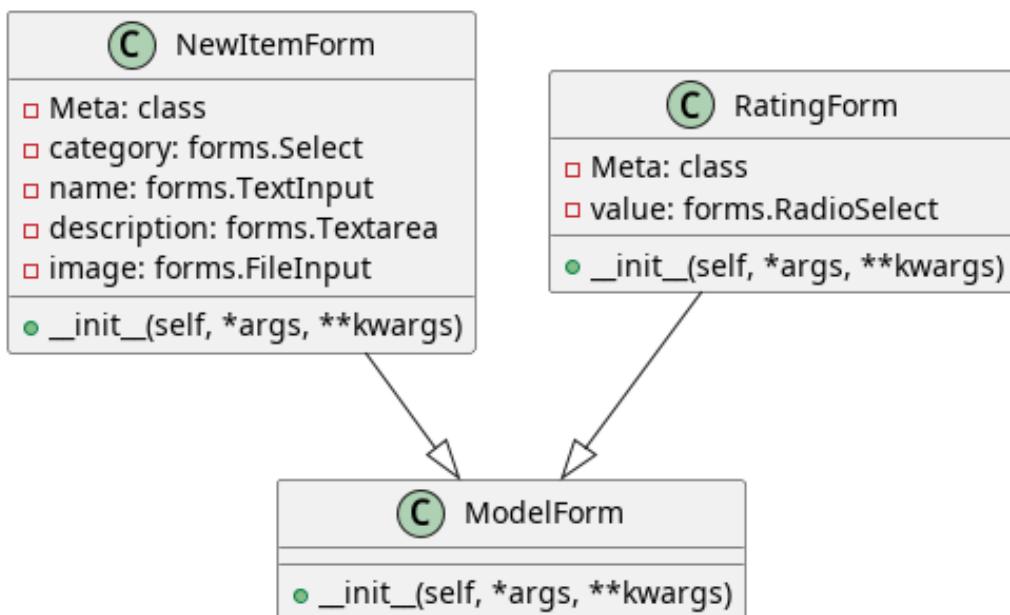


Figura 3.4

3.2.2 Models

I modelli di base utilizzati per la creazione sia nel database che per i form.

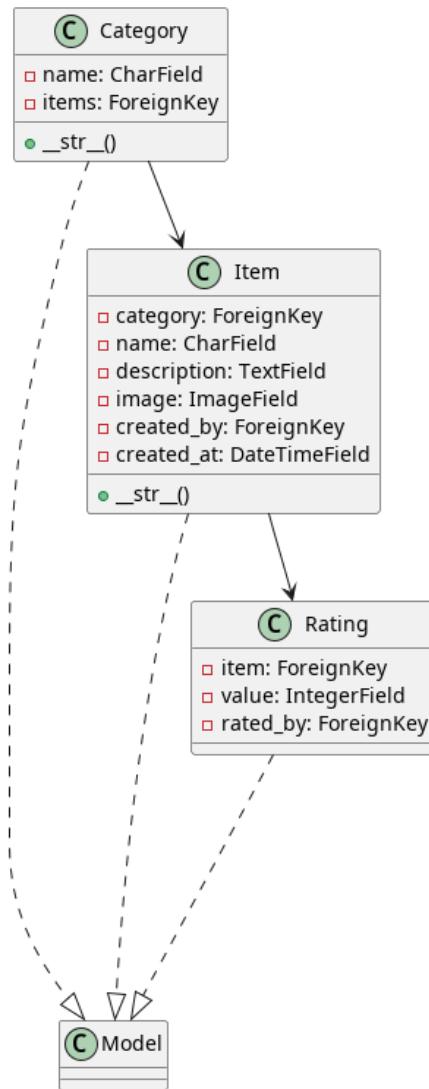


Figura 3.5

3.2.3 Views

La parte di visualizzazione e gestione del sito attraverso le funzioni.

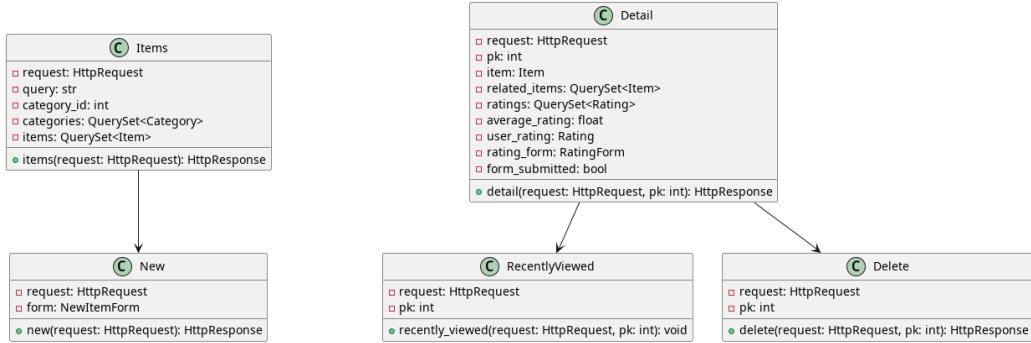


Figura 3.6

3.2.4 UML Finale

L'intera comunicazione nell'applicazione *Items*.

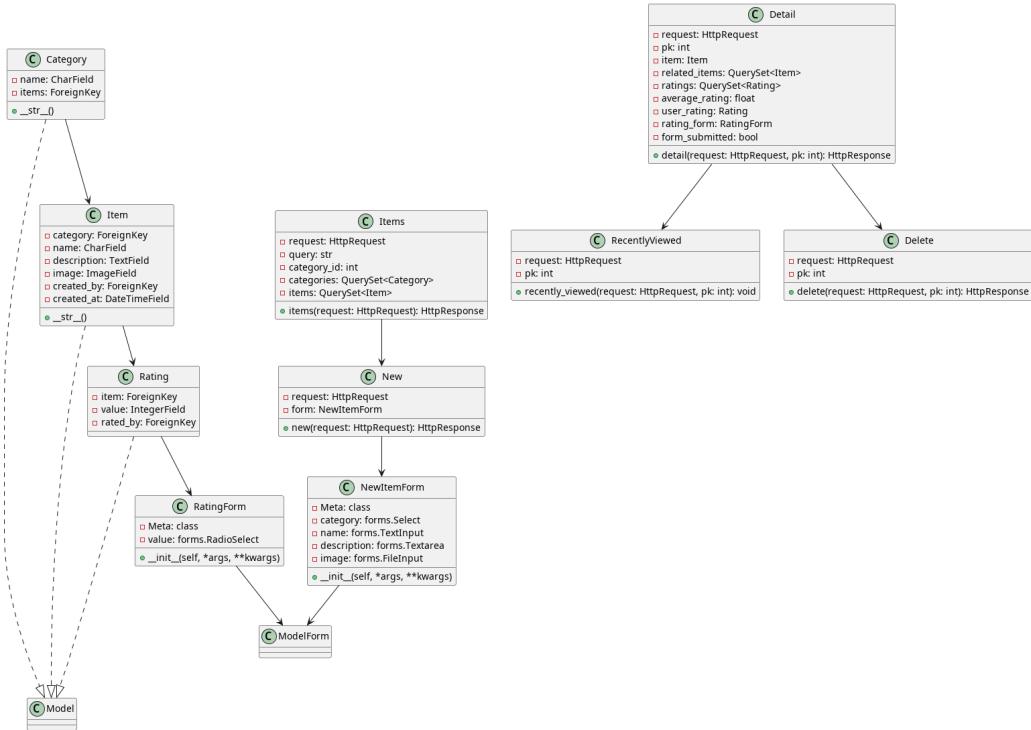


Figura 3.7

Conclusione

In conclusione, il progetto sviluppato rappresenta un'eccitante fusione di complessità e semplicità nell'implementazione di un sito web utilizzando il framework Django. Attraverso la scelta intuitiva e accessibile di questo framework, basato su Python, siamo stati in grado di realizzare un ambiente digitale che offre agli appassionati di fotografia una piattaforma dinamica e accogliente per condividere la propria creatività e interagire con una community selezionata.

Sono stati utilizzati vari microservizi per fornire un ambiente semplice per gli utenti, grazie alla facilità di utilizzo del framework **Django**, che ha dato la possibilità di creare un sistema semplice, pulito e leggibile, sia per chi implementa il codice, sia per chi lo analizza.

In definitiva, questo progetto non solo fornisce uno spazio per il rating fotografico, ma funge anche da punto d'incontro per gli amanti della fotografia, arricchendo l'esperienza di tutti gli utenti coinvolti attraverso la qualità delle interazioni e la condivisione di idee e feedback.

Bibliografia

- [1] László Viktor Jánoky, János Levendovszky, and Péter Ekler. An analysis on the revoking mechanisms for json web tokens. *International Journal of Distributed Sensor Networks*, 14(9):1550147718801535, 2018.
- [2] Maliha Khan, Sudeshna Chakraborty, Rani Astya, and Shaveta Khepra. Face detection and recognition using opencv. In *2019 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)*, pages 116–119, 2019.
- [3] Carl Burch. Django, a web framework using python: tutorial presentation. *J. Comput. Sci. Coll.*, 25(5):154–155, may 2010.