Sommario

INTRODUZIONE *2*

1 PANORAMICA DI DJANGO E L'ARCHITETTURA MVC *3*

1.1 Struttura di base di un progetto Django 3

2 SPECIFICA DEI REQUISITI *5*

2.2 Use Case diagram 5

2.3 Class Diagram 8

3 AJAX *11*

4 SCREENSHOTS *13*

5 TEST *15*

6 CONCLUSIONI FINALI *16*

INTRODUZIONE

Il presente documento descrive lo sviluppo di un sito web a tema social network utilizzando il framework Django. Il sito web è stato sviluppato per l'esame di Tecnologie Web erogato dall'università di Modena e Reggio Emilia, tenuto dalla professoressa Claudia Canali e dal professore Nicola Capodieci. Il sito è puramente a scopo didattico per la verifica delle conoscenze acquisite durante il corso sopra citato.

Nel seguente documento, verranno presentati i modelli dati utilizzati, le view e il codice JavaScript AJAX implementato per migliorare l'esperienza utente. Saranno anche fornite informazioni sulle funzionalità del sito, i casi d'uso e un diagramma delle classi.

1 PANORAMICA DI DJANGO E L'ARCHITETTURA MVC

Il sito web è stato sviluppato utilizzando Django, un framework web Python che facilita la creazione di applicazioni web complesse.

Django offre diversi vantaggi per lo sviluppo di siti web. Prima di tutto, fornisce un'architettura ben definita che promuove la separazione delle responsabilità tra i componenti del sito web. Più in dettaglio, Django adotta un'architettura MVC, che suddivide il codice in tre componenti principali: il Model, che rappresenta i dati e la logica sottostante, la View, che gestisce la presentazione dei dati agli utenti, e il Controller, che gestisce le interazioni degli utenti con il sistema. L'architettura MVC consente una separazione chiara delle responsabilità e promuove una migliore organizzazione e manutenibilità del codice. Questa separazione consente una migliore organizzazione del codice e una manutenibilità più semplice. Inoltre, Django offre una serie di funzionalità integrate come l'autenticazione degli utenti, la gestione delle sessioni e l'accesso al database, che semplificano la gestione delle operazioni comuni.

1.1 Struttura di base di un progetto Django

Un progetto Django è organizzato in una struttura gerarchica ben definita, che permette una gestione ordinata e modulare del codice. La cartella principale del progetto contiene i file di configurazione principali e funge da punto di partenza per lo sviluppo dell'applicazione.

All'interno della directory del progetto, sono presenti diversi file e cartelle che svolgono ruoli specifici nel contesto dell'applicazione Django. Alcuni dei file di configurazione principali sono:

1. File di configurazione del percorso (urls.py): Questo file definisce i percorsi dell'applicazione, associando le URL alle view corrispondenti. I percorsi specificano quale view deve essere chiamata quando un determinato URL viene richiesto dal client.

2. File di configurazione del database (settings.py): Questo file contiene le impostazioni di configurazione del database utilizzato dall'applicazione Django. Qui è possibile specificare il tipo di database da utilizzare, le credenziali di accesso e altre opzioni di configurazione.

Oltre ai file di configurazione, all'interno della directory del progetto sono presenti anche le cosiddette "applicazioni". Le applicazioni sono moduli autonomi che gestiscono specifiche funzionalità dell'applicazione globale. Ogni applicazione ha una propria cartella, che contiene i file necessari per il suo funzionamento. Alcuni dei file più importanti all'interno di una cartella di un'applicazione sono:

1. Modelli (models.py): Questo file definisce i modelli di dati utilizzati dall'applicazione. I modelli sono classi Python che rappresentano le tabelle del database e contengono i campi e le relazioni tra i dati.

2. View (views.py): Questo file contiene le view dell'applicazione, ovvero le funzioni o le classi che gestiscono le richieste del client e producono le risposte corrispondenti. Le view sono responsabili di elaborare i dati e interagire con i modelli per ottenere le informazioni necessarie da mostrare all'utente.

3. Template: La cartella dei template contiene i file HTML che definiscono la struttura e il layout delle pagine dell'applicazione. I template permettono di separare la logica di presentazione dai dati, consentendo una maggiore flessibilità e facilità di manutenzione.

4. File statici: La cartella dei file statici contiene risorse statiche come fogli di stile CSS, file JavaScript e immagini. Questi file vengono serviti direttamente dal server web e vengono utilizzati per personalizzare l'aspetto e il comportamento dell'applicazione.

La struttura modulare del progetto Django permette di organizzare il codice in modo chiaro e di suddividere le responsabilità in base alle funzionalità specifiche. Ogni applicazione può essere sviluppata e testata separatamente, rendendo il progetto più gestibile e scalabile.

2 SPECIFICA DEI REQUISITI

Il social network offre una serie di funzionalità per gli utenti. Questi possono registrarsi al sito fornendo un nome utente, un indirizzo e-mail e una password. Una volta autenticati, gli utenti possono creare nuovi post contenenti testo e immagini per condividerli con gli altri utenti. È possibile commentare i post di altri utenti per esprimere opinioni o porre domande. Gli utenti possono anche seguire o smettere di seguire altri utenti, visualizzando così i post dei profili seguiti nella propria pagina principale. È possibile salvare i post interessanti per una visione successiva. Il profilo degli utenti può essere personalizzato con informazioni aggiuntive come il nome, la biografia e le immagini di profilo e di copertina. Gli utenti possono cercare altri utenti e post utilizzando parole chiave specifiche. Inoltre, è disponibile la funzionalità di chat privata tra gli utenti, consentendo loro di inviare messaggi e comunicare direttamente. Infine, il sito web implementa un meccanismo di suggerimenti che fornisce all'utente in sessione una lista di altri utenti consigliati per essere seguiti.

2.2 Use Case diagram

Di seguito vengono riportati i requisiti funzionali del sistema tramite Use-case diagram **(Figura 2.1)** e tramite descrizione a parole:

1. Registrazione:
    - L'utente fornisce un nome utente, un indirizzo e-mail e una password per registrarsi al social network.
    - Il sistema verifica la validità dei dati e crea un nuovo account per l'utente.

2. Autenticazione:
    - L'utente inserisce le credenziali di accesso (nome utente e password) per effettuare l'autenticazione.
    - Il sistema verifica le credenziali e autentica l'utente, consentendogli di accedere alle funzionalità del social network.
3. Creazione di nuovi post:
    - L'utente autenticato può creare nuovi post contenenti testo e immagini.
    - L'utente inserisce il contenuto del post e lo invia.
    - Il sistema salva il post nel database e lo rende visibile agli altri utenti.
4. Commento ai post:
    - L'utente può commentare i post di altri utenti per esprimere opinioni o porre domande.
    - L'utente seleziona un post e inserisce il commento desiderato.
    - Il sistema salva il commento associato al post corrispondente.
5. Follow/Unfollow di altri utenti:
    - L'utente può seguire altri utenti per visualizzare i loro post nella propria pagina principale.
    - L'utente seleziona un altro utente da seguire o smette di seguirlo.
    - Il sistema aggiorna la lista dei profili seguiti dall'utente corrente.
6. Salvataggio di post:
    - L'utente può salvare i post interessanti per una visione successiva.
    - L'utente seleziona il post e lo salva nella propria lista di post salvati.

7. Personalizzazione del profilo:

    - L'utente può personalizzare il proprio profilo aggiungendo informazioni come il nome, la biografia e le immagini di profilo e di copertina.
    - L'utente modifica le informazioni del proprio profilo e le salva.
    - Il sistema aggiorna il profilo dell'utente con le informazioni modificate.

8. Ricerca di altri utenti e post:

    - L'utente può cercare altri utenti e post utilizzando parole chiave specifiche.
    - L'utente inserisce le parole chiave nella barra di ricerca.
    - Il sistema restituisce i risultati corrispondenti alle parole chiave inserite.

9. Chat privata:

    - L'utente può comunicare direttamente con altri utenti tramite chat privata.
    - L'utente seleziona un altro utente con cui desidera chattare.
    - Il sistema crea una chat privata tra i due utenti e permette loro di inviare messaggi.

10. Suggerimenti di follow:

    - Il sistema fornisce all'utente una lista di altri utenti consigliati per essere seguiti, basandosi su algoritmi di raccomandazione.
    - L'utente visualizza la lista di suggerimenti di follow proposti dal sistema.

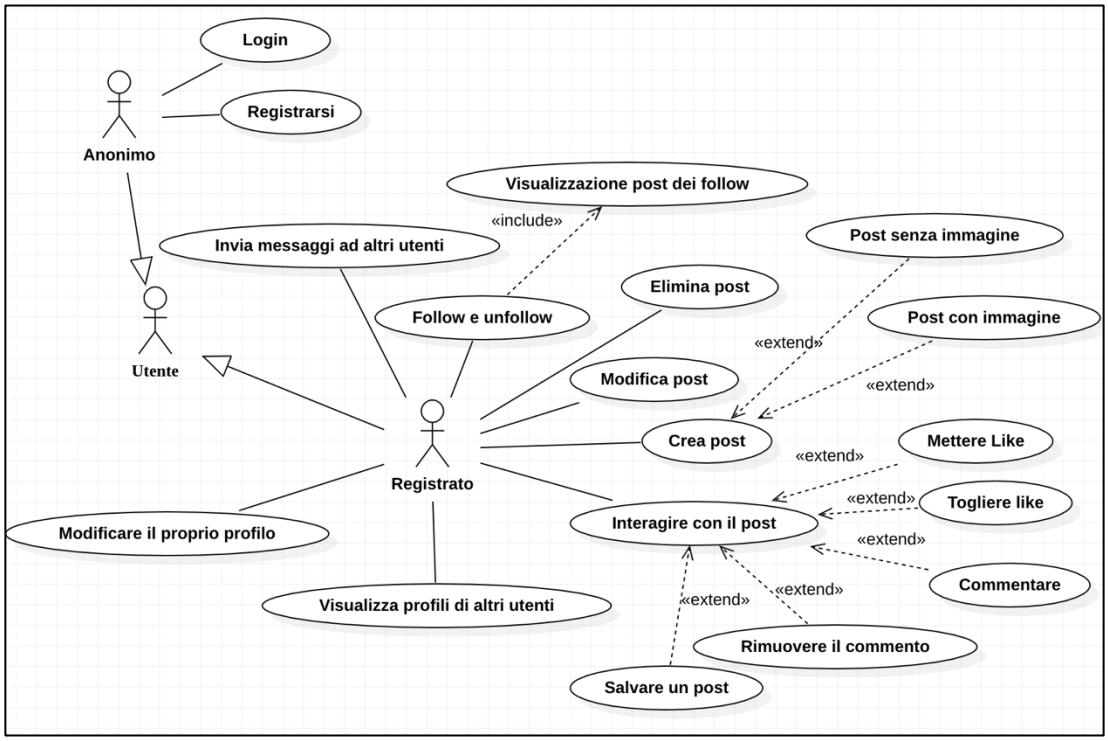

Figura 2.1 Use-case diagram.

2.3 Class Diagram

Di seguito viene riportato il class diagram **(Figura 2.2)** reso disponibile tramite Pycharm e alcuni screenshot **(Figura 2.3 e Figura 2.4)** dell'implementazione delle classi:

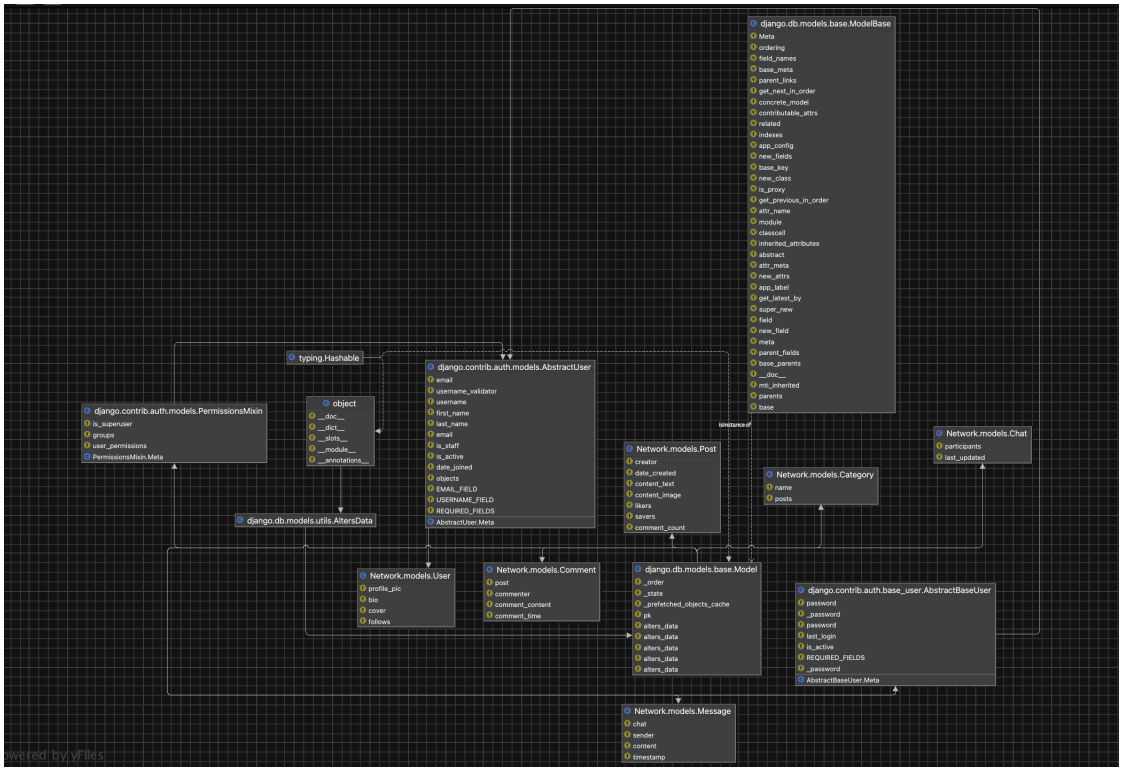

Figura 2.2 Class diagram, Pycharm.

```python
Ayoub Imad
class User(AbstractUser):
    profile_pic = models.ImageField(upload_to='profile_pic/')
    bio = models.TextField(max_length=160, blank=True, null=True)
    cover = models.ImageField(upload_to='covers/', blank=True)
    follows = models.ManyToManyField("self", related_name="followed_by", symmetrical=False, blank=True)

    Ayoub Imad
    def __str__(self):
        return self.username

    Ayoub Imad
    def serialize(self):
        return {
            'id': self.id,
            "username": self.username,
            "profile_pic": self.profile_pic.url,
            "first_name": self.first_name,
            "last_name": self.last_name
        }

    Ayoub Imad
    def get_suggested_followers(self):
        followed_users = self.follows.all()
        suggested_users = User.objects.exclude(id=self.id).exclude(id__in=followed_users).filter(
            followed_by__in=followed_users)
        suggested_users = suggested_users.annotate(num_my_followers=Count('followed_by__follows')).order_by(
            '-num_my_followers')[:5]
        return suggested_users
```

Figura 2.3 User class, file models.py in Network application.

```python
Ayoub Imad
class Post(models.Model):
    creator = models.ForeignKey(User, on_delete=models.CASCADE, related_name='posts')
    date_created = models.DateTimeField(default=timezone.now)
    content_text = models.TextField(max_length=140, blank=True)
    content_image = models.ImageField(upload_to='posts/', blank=True)
    likers = models.ManyToManyField(User, blank=True, related_name='likes')
    savers = models.ManyToManyField(User, blank=True, related_name='saved')
    comment_count = models.IntegerField(default=0)

    Ayoub Imad
    def __str__(self):
        return f"Post ID: {self.id} (creator: {self.creator})"
```

Figura 2.4 Post class, file models.py in Network application.

3 AJAX

AJAX (Asynchronous JavaScript And XML) è una tecnologia di sviluppo web che consente di aggiornare le pagine web in modo asincrono, senza dover ricaricare l'intera pagina. AJAX utilizza una combinazione di JavaScript e XML (o altri formati dati come JSON) per inviare richieste al server in background e ricevere i dati richiesti senza interrompere l'interazione dell'utente con la pagina. Ciò consente di creare un'esperienza utente più fluida e reattiva, in cui le modifiche e gli aggiornamenti possono essere applicati dinamicamente senza dover ricaricare completamente la pagina.

Nel contesto del social network sviluppato con Django, AJAX è stato utilizzato per migliorare l'interattività e l'efficienza dell'applicazione. Ad esempio, le richieste AJAX sono state utilizzate per consentire agli utenti di inviare commenti, salvare i post o eseguire follow/unfollow ad altri utenti senza dover ricaricare l'intera pagina. In questo modo, è possibile ottenere risposte immediate e visualizzare le modifiche apportate senza interruzioni o ritardi nell'esperienza utente.

Di seguito **(Figura 3.1 e Figura 3.2)** viene riportata una interazione AJAX tra client e server:

```
function save_post(element) {
    let id = element.dataset.post_id;
    fetch( input: '/post/' + parseInt(id) + '/save', init: {
        method: 'PUT'
    })
        .then(response => {
            if (response.ok) {
                let count = element.querySelector('.savers_count');
                let value = count.innerHTML;
                value++;
                count.innerHTML = value;
                element.querySelector('.svg-span').innerHTML = `<svg xmlns="http://www.w3.org/2000/svg" width="16"...>`;
                element.setAttribute( qualifiedName: 'onclick', value: 'unsave_post(this)');
            } else {
                console.log('Failed to save post.');
            }
        })
        .catch(error => {
            console.log('Error occurred while saving post:', error);
        });
}
```

Figura 3.1 funzione save_post(), in script.js file.

Figura 3.2 funzione save_post(), views.py in Network application.

4 SCREENSHOTS

Di seguito vengono riportati alcuni screenshot dell'applicazione, in particolare della homepage **(Figura 4.1)**, del profilo utente **(Figura 4.2),** di una chat **(Figura 4.3)** e infine di una creazione di un post **(Figura 4.4)**.

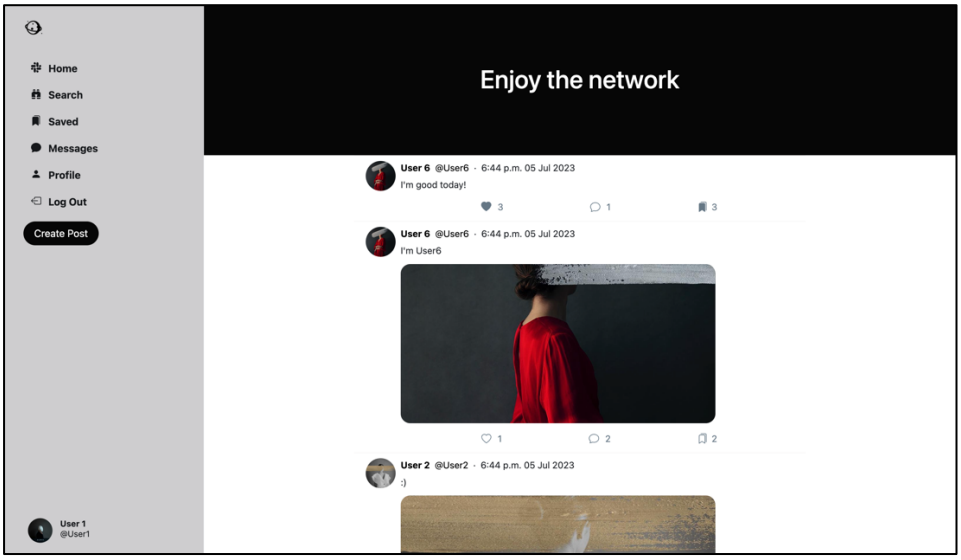

Figura 4.1 home page.

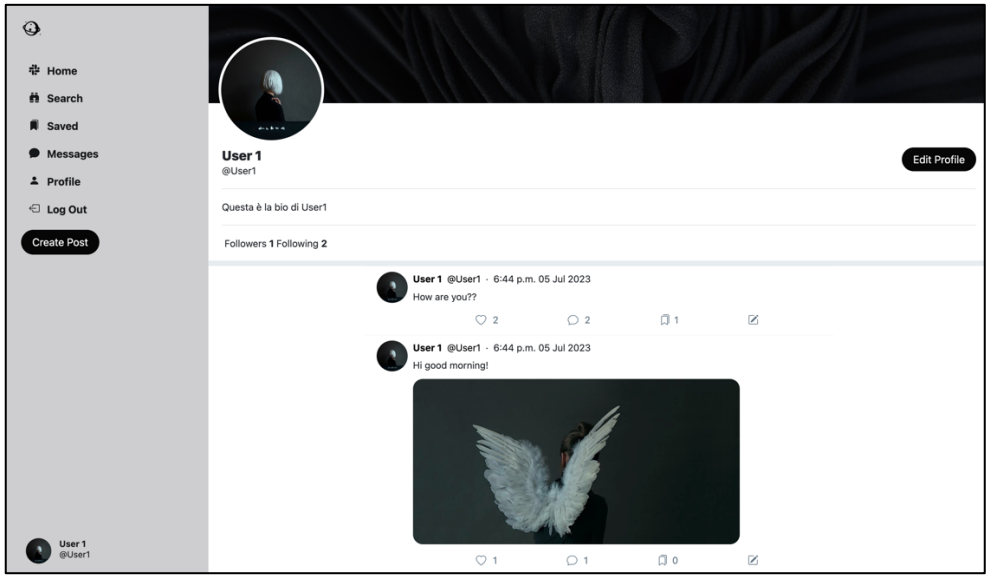

Figura 4.2 profilo utente.

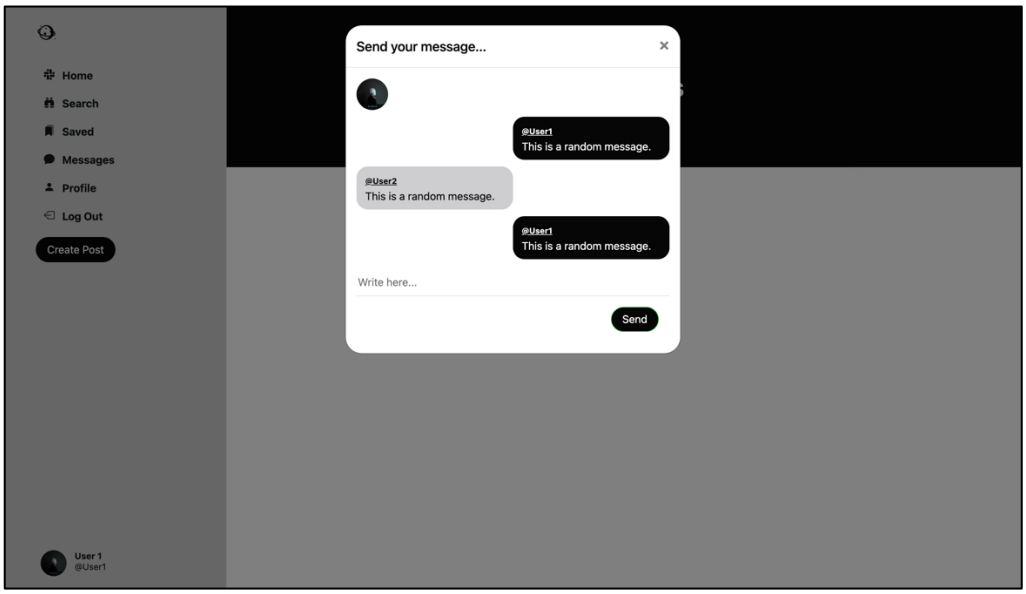

Figura 4.3 Interazione tra utenti tramite messaggi.

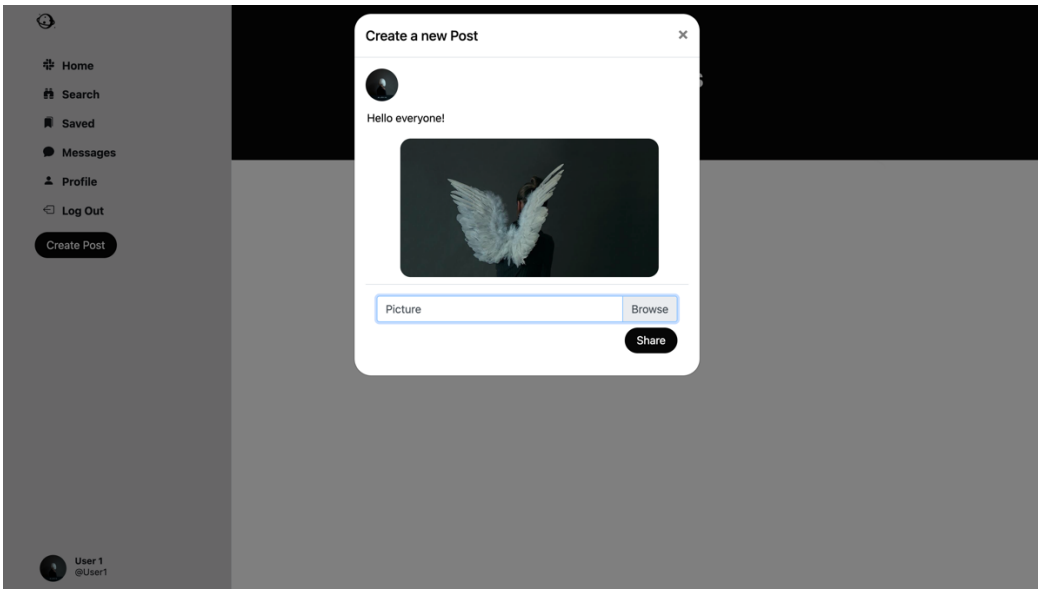

Figura 4.4 creazione di un post.

5 TEST

Di seguito vengono riportati due test effettuati, uno per il corretto funzionamento del meccanismo di suggerimento utenti **(Figura 5.1)** e uno per la view che gestite il like e unlike di un post **(Figura 5.2)** :

```python
Ayoub Imad *
class UserTestCase(TestCase):
    Ayoub Imad *
    def setUp(self):
        self.user1 = User.objects.create(username='user1')
        self.user2 = User.objects.create(username='user2')
        self.user3 = User.objects.create(username='user3')
        self.user4 = User.objects.create(username='user4')
        self.user5 = User.objects.create(username='user5')
        self.user6 = User.objects.create(username='user6')
        self.user1.follows.add(self.user2, self.user3, self.user4)
        self.user2.follows.add(self.user5)
        self.user3.follows.add(self.user5, self.user6)
        self.user4.follows.add(self.user1, self.user2)
        self.user5.follows.add(self.user1, self.user3)

    Ayoub Imad
    def test_get_suggested_followers(self):
        suggested_followers = self.user1.get_suggested_followers()
        self.assertEqual(len(suggested_followers), 2)
        self.assertEqual(suggested_followers[0], self.user5)
        self.assertEqual(suggested_followers[1], self.user6)
        self.assertNotIn(self.user1, suggested_followers)
```

Figura 5.1 test case per meccanismo di suggerimento utenti.

```python
Ayoub Imad *
class LikeUnlikePostViewTestCase(TestCase):
    Ayoub Imad *
    def setUp(self):
        self.user = User.objects.create_user(username='testuser', password='testpassword')
        self.post = Post.objects.create(creator=self.user)

    Ayoub Imad *
    def test_like_post(self):
        self.client.login(username='testuser', password='testpassword')
        response = self.client.put(reverse('like_post', args=[self.post.pk]))
        self.assertEqual(response.status_code, 200)
        self.assertIn(self.user, self.post.likers.all())

    Ayoub Imad *
    def test_unlike_post(self):
        self.client.login(username='testuser', password='testpassword')
        self.post.likers.add(self.user)
        response = self.client.put(reverse('unlike_post', args=[self.post.pk]))
        self.assertEqual(response.status_code, 200)
        self.assertNotIn(self.user, self.post.likers.all())
```

Figura 5.2 test case view like e unlike.

6 CONCLUSIONI FINALI

Per maggiori dettagli, il repository GitHub del progetto è recuperabile al seguente link: https://github.com/Ayoubimad/SocialWebSite , con istruzioni per l'installazione se si volesse provare il sito.