



**MANIPAL INSTITUTE OF TECHNOLOGY**  
**MANIPAL**  
*(A constituent unit of MAHE, Manipal)*

# **IT Lab Mini Project Report on**

## **Social Media App**

**SUBMITTED  
BY**

1. Prahlad Menon-210962057-30
2. Mutyala Jayasuryan-210962009-5

April 2024

### **Abstract**

This paper explores the conceptualization, design, and implementation of a dynamic social media platform developed using Django, HTML, CSS, and JavaScript. The application offers users a comprehensive suite of features, including account creation, profile customization, and interactive engagement with other users. Central to its functionality is the ability for users to upload profile pictures, write bios, and specify their locations, fostering a sense of identity and community within the platform. Moreover, users can seamlessly navigate through the app to discover and connect with other users, enabling interactions such as following, viewing posts, and sharing content. By leveraging Django's robust framework and incorporating responsive front-end design techniques, the application provides an intuitive and engaging user experience across various devices and screen sizes. This report delves into the technical architecture of the app, discussing key components such as how the templates were created, what role the database plays, etc. Overall, this project underscores the versatility and potential impact of utilizing web development technologies to create innovative and user-centric social media platforms.

# **Table of Contents**

## **Abstract**

## **Table of Contents**

### **1. Introduction**

- 1.1 Project Overview
- 1.2 Problem Statement and Objectives

### **2. Technologies Used**

- 2.1 Frontend Technologies
  - 2.1.1 HTML
  - 2.1.2 Bootstrap
  - 2.1.3 Tailwind CSS
  - 2.1.4 JavaScript
- 2.2 Backend Technologies
  - 2.2.1 Django
  - 2.2.2 SQLite3

### **3. Functionality**

- 3.1 Login
- 3.2 Register
- 3.3 Account Settings
- 3.4 Homepage
  - 3.4.1 Uploading Pictures
  - 3.4.2 Suggestions for following
  - 3.4.3 Search
- 3.5 Viewing profiles

### **4. Models**

- 4.1 Overview of the Main Models
- 4.2 Description of the Profile Model
- 4.3 Description of the Post Model
- 4.4 Description of the LikePost Model
- 4.5 Description of the FollowersCount Model

## **5. Views**

- 5.1 Overview of the Views Used in the Application
- 5.2 Description of the index View
- 5.3 Description of the settings View
- 5.4 Description of the upload View
- 5.5 Description of the search View
- 5.6 Description of the like\_post View
- 5.7 Description of the profile View
- 5.8 Description of the follow View
- 5.9 Description of the signup View
- 5.10 Description of the signin View
- 5.11 Description of the logout View

## **6. Conclusion**

- 6.1 Summary of Key Points
- 6.2 Future Prospects and Improvements

# **1. Introduction**

## **1.1 Project Overview**

The social media application offers users a versatile platform for creating, customizing, and engaging with content and other users. Key features include user account management, profile setup with options for profile picture upload, bio creation, and location specification. Users can explore the platform to discover and connect with other users, follow their activity, view posts, and share their own content. The project focuses on delivering a seamless user experience through responsive design principles and robust back-end functionality.

Building a small-scale social media app for college fosters community and connection among students. It promotes sharing of academic resources, organizing events, and facilitating communication. Such a platform can enhance student engagement and provide a tailored space to address specific college needs and interests.

## **1.2 Problem Statement and Objectives**

Problem Statement:

The proliferation of social media platforms has transformed the way individuals communicate, share information, and build communities online. However, existing platforms often lack certain features or fail to provide a seamless user experience, leading to user dissatisfaction and limited engagement. Additionally, concerns regarding data privacy and security continue to challenge the trustworthiness of many social media platforms. In response to these shortcomings, there is a growing demand for innovative social media solutions that prioritize user experience, functionality, and security.

Motivation:

We designed the social media app for promoting college community events and promote student engagement. It promotes sharing of organizing events and recommending new users to follow. This can enhance student engagement and provide a tailored space to address specific college needs and interests.

## Objectives:

- 1) Develop a user-friendly social media application using Django, HTML, CSS, and JavaScript, focusing on core features such as account creation, profile customization, and post interactions.
- 2) Enable users to create accounts, personalize profiles with profile pictures, bios, and locations through the user settings, and also, they can engage with other users.
- 3) Implement secure user authentication and login mechanisms to ensure the protection of user accounts and sensitive information.
- 4) Implement a user-friendly interface with responsive design principles to ensure optimal user experience across different devices and screen sizes.
- 5) Facilitate meaningful interactions and community-building within the platform by providing discoverability features, and content-sharing capabilities.
- 6) Enable users to follow and unfollow other users within the platform, facilitating social connections and personalized content discovery

## 2. Technologies Used

Our application leverages a combination of powerful frontend and backend technologies to deliver a seamless user experience and robust functionality.

### 2.1 Frontend Technologies

The front end is responsible for the visual elements and user interactions within the web application. Here's a breakdown of the critical frontend technologies used:

#### 2.1.1 HTML (Hypertext Markup Language)

HTML serves as the foundation of our front end. It defines the structure and content of the web pages, including elements like headings, paragraphs, forms, and buttons. HTML provides the basic building blocks styled and manipulated using other technologies.

- **HTML** is the standard markup language used to create the structure and content of web pages.
- HTML documents are interpreted by web browsers to render the visual representation of a web page for users.
- It provides a set of elements or tags that define the different parts of a web page, such as headings, paragraphs, links, images, forms, and more.

#### 2.1.2 Bootstrap

Bootstrap is a popular CSS framework that simplifies the creation of responsive and visually appealing user interfaces. Our application utilizes Bootstrap to achieve the following:

- **Pre-built Components:** Bootstrap offers a rich library of pre-built components like buttons, navigation bars, and forms. These components streamline the development process and promote consistency in the application's look and feel.
- **Grid System:** Bootstrap's grid system provides a flexible layout structure, allowing developers to arrange elements on the page in a responsive and visually balanced manner.
- **JavaScript Plugins:** It also includes JavaScript plugins for adding interactive features like carousels, modals, tooltips, and dropdown menus, further enhancing the functionality of web applications.

### 2.1.3 Tailwind CSS

- Tailwind CSS is a utility-first CSS framework that provides a set of pre-built utility classes for styling HTML elements.
- Unlike traditional CSS frameworks that rely on pre-defined components and styles, Tailwind CSS allows developers to apply styles directly to HTML elements using utility classes.
- It offers a highly customizable and flexible approach to styling, enabling developers to quickly build and iterate on designs without writing custom CSS.

### 2.1.4 JavaScript

- JavaScript is a versatile programming language commonly used for building interactive and dynamic web applications.
- It runs on the client-side within web browsers and allows developers to manipulate HTML content, respond to user interactions, and dynamically update web page elements.

## 2.2 Backend Technologies

The backend of our app handles user authentication, database interactions, etc to facilitate user interactions and data management within the social media platform.

### 2.2.1 Django

Django is a high-level Python web framework renowned for its rapid development capabilities and clean, pragmatic design emphasis. Our web application leverages Django's features to achieve:

- **Model-View-Template (MVT) Architecture:** By employing the MVT architecture, Django promotes a clean separation of concerns. This structure keeps the business logic (models), user interface (templates), and request handling (views) distinct, leading to maintainable and scalable code.
- **Object-Relational Mapper (ORM):** Django's ORM simplifies database interaction by providing an abstraction layer. Developers can work with data using Python objects, and Django handles the underlying database communication.
- **User Authentication and Authorization:** Django offers built-in mechanisms for user authentication and authorization, allowing for secure user management within the application.
- **Admin Interface:** Django provides a user-friendly interface that enables administrators to manage the application's content, users, and other aspects.

### 2.2.2 SQLite3

SQLite3 is a lightweight, embeddable relational database management system. Here's how SQLite3 contributes to our application:

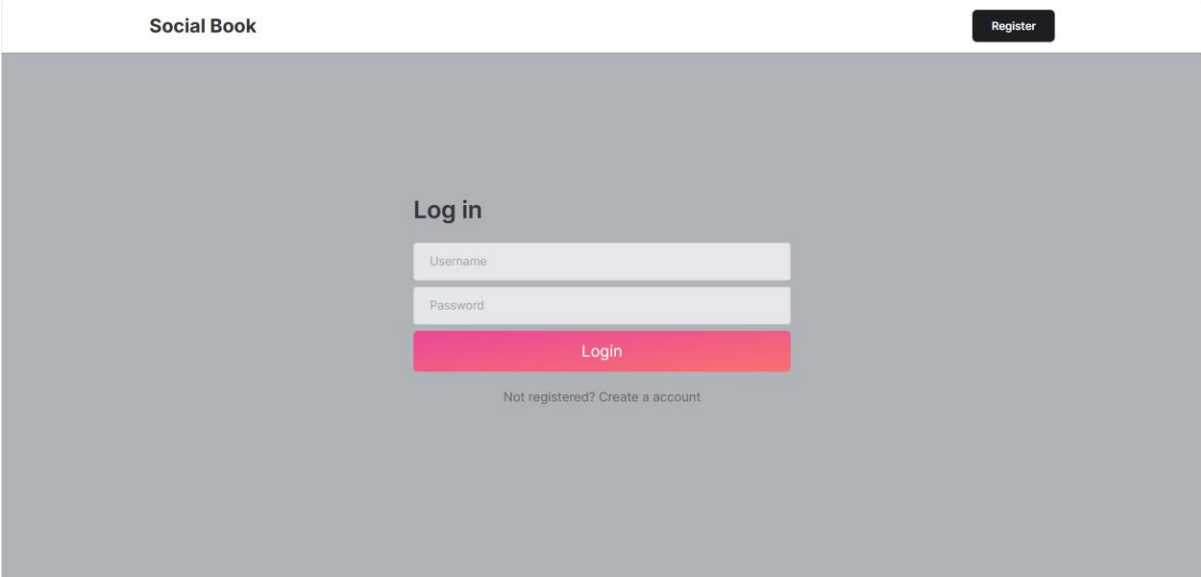
- **Database Storage:** SQLite3 is the storage engine for our application's data. It manages the application's creation, storage, and retrieval of information.
- **Lightweight and Efficient:** SQLite3 is known for its small footprint and efficient operation. These characteristics make it ideal for web applications where performance and scalability are important considerations.
- **Embedded Database:** Unlike traditional client-server database models, SQLite3 can be embedded directly within the application. This eliminates the need for a separate database server, simplifying deployment and maintenance.



## 3. Functionality

### 3.1 Login

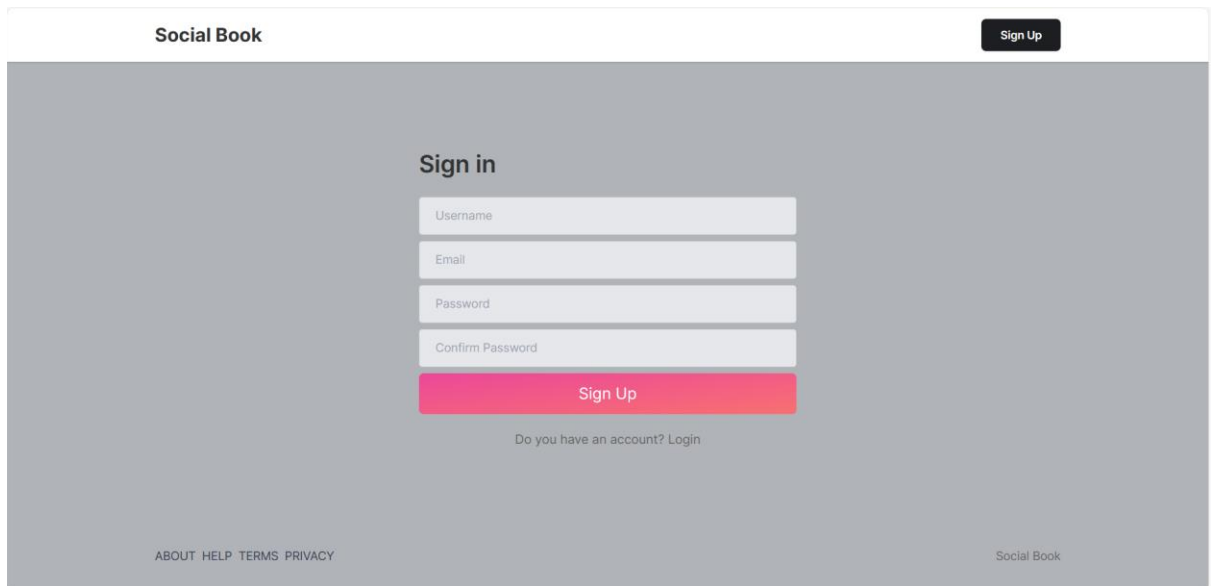
- Existing users can access the platform by providing their registered username and password through a login form.
- The form submits data to a Django view function that is responsible for user authentication. This process involves validating the username and password against stored user credentials in the database.
- Upon successful login, users are led to a personalized homepage within the application.



The screenshot displays the 'Social Book' login interface. At the top left, the text 'Social Book' is visible. At the top right, there is a dark button labeled 'Register'. The main content area has a light gray background and features a 'Log in' heading. Below the heading are two input fields: 'Username' and 'Password'. A prominent pink 'Login' button is positioned below these fields. At the bottom of the form, a link reads 'Not registered? Create a account'.

### 3.2 Register

- New users can create accounts by providing their username, email address, and password through a registration form.
- The form submits data to a Django view function that is responsible for user authentication. The view ensures the entered information adheres to specific formats and prevents potential security vulnerabilities.
- After successful registration, users are redirected to their home page.



The image shows a web page for 'Social Book' with a 'Sign in' form. The form includes input fields for Username, Email, Password, and Confirm Password, followed by a pink 'Sign Up' button. Below the button is a link: 'Do you have an account? Login'. The page has a header with 'Social Book' and a 'Sign Up' button, and a footer with 'ABOUT HELP TERMS PRIVACY' and 'Social Book'.

Social Book

Sign Up

### Sign in

Username

Email

Password

Confirm Password

Sign Up

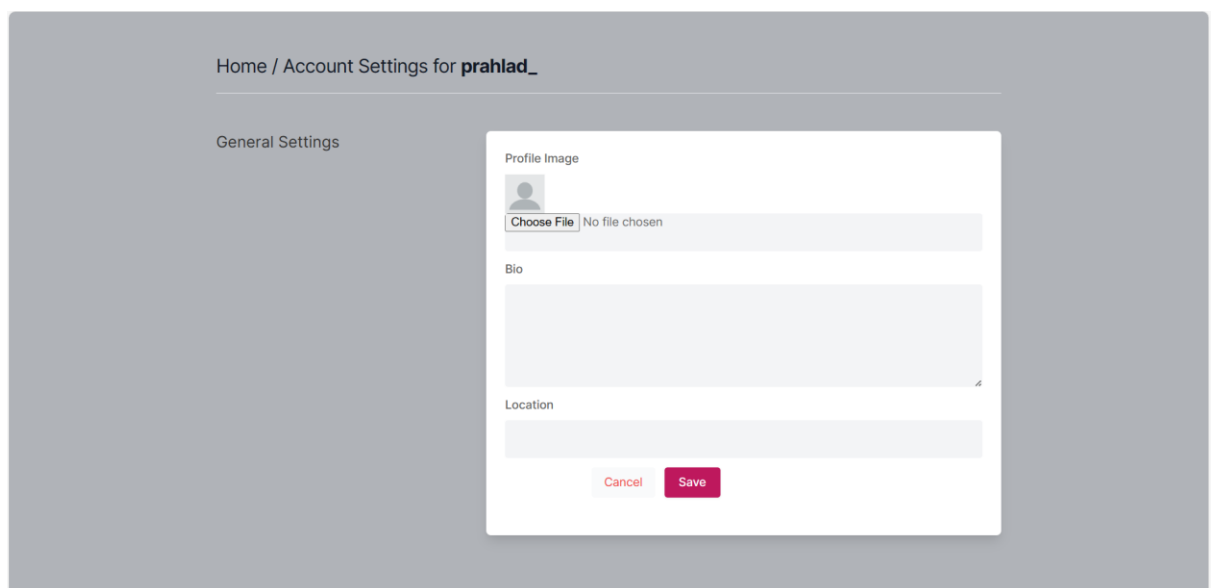
Do you have an account? Login

ABOUT HELP TERMS PRIVACY

Social Book

### 3.3 Account Settings

- Allows updating Profile Image, Bio, and Location. This page is also displayed after immediately creating a new account.



The image shows the 'Account Settings' page for a user named 'prahlad\_'. The page has a breadcrumb 'Home / Account Settings for prahlad\_'. Under 'General Settings', there is a modal form for updating the profile. The modal includes a 'Profile Image' section with a 'Choose File' button and 'No file chosen' text, a 'Bio' text area, and a 'Location' text input. At the bottom of the modal are 'Cancel' and 'Save' buttons.

Home / Account Settings for prahlad\_

General Settings

Profile Image

Choose File No file chosen

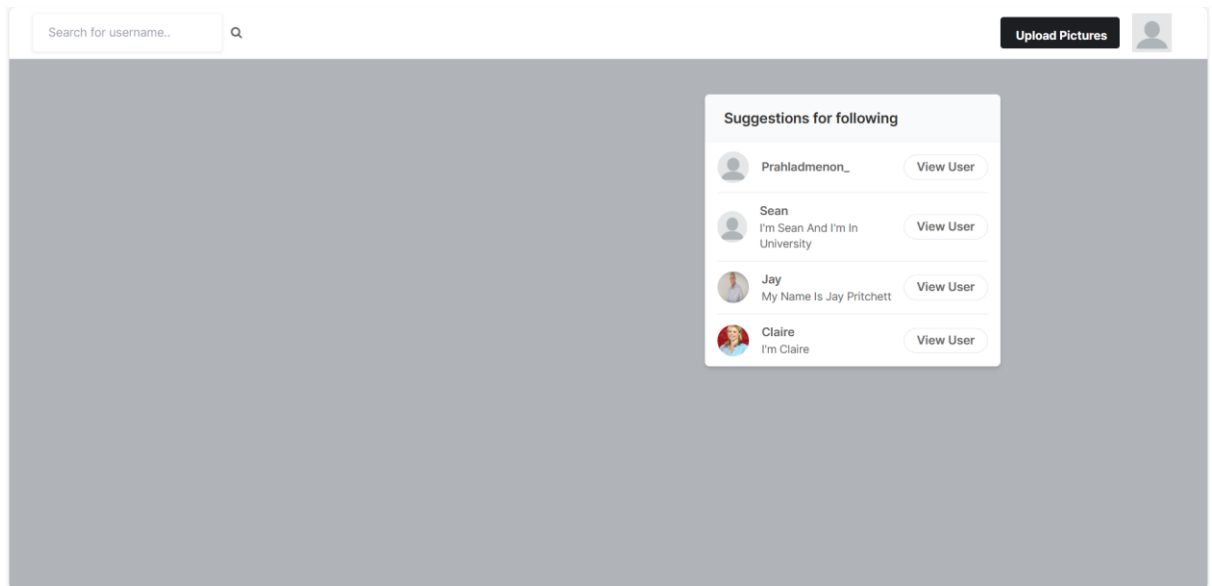
Bio

Location

Cancel Save

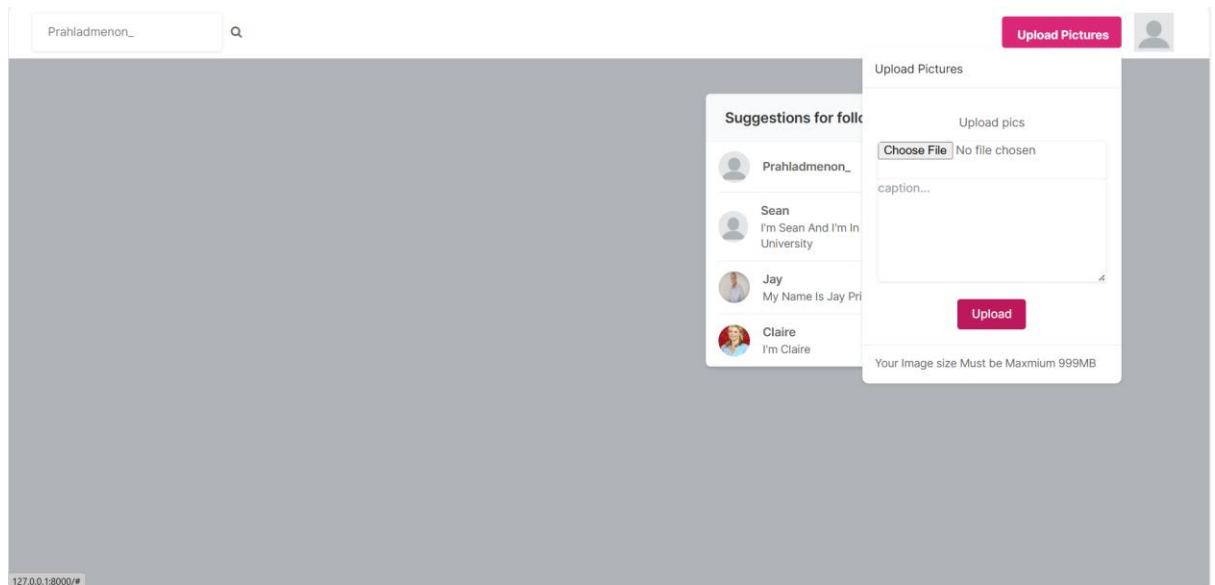
### 3.4 Homepage

- Once an account is created, or you have logged in to an existing account, the user is taken to the home page.
- From this page, many other functionalities such as uploading pictures, logging out, searching for other users, viewing other users' profile and posts can be accessed.



### 3.4.1 Uploading pictures

- From the homepage, you have an option to upload pictures and hence create posts as shown below.

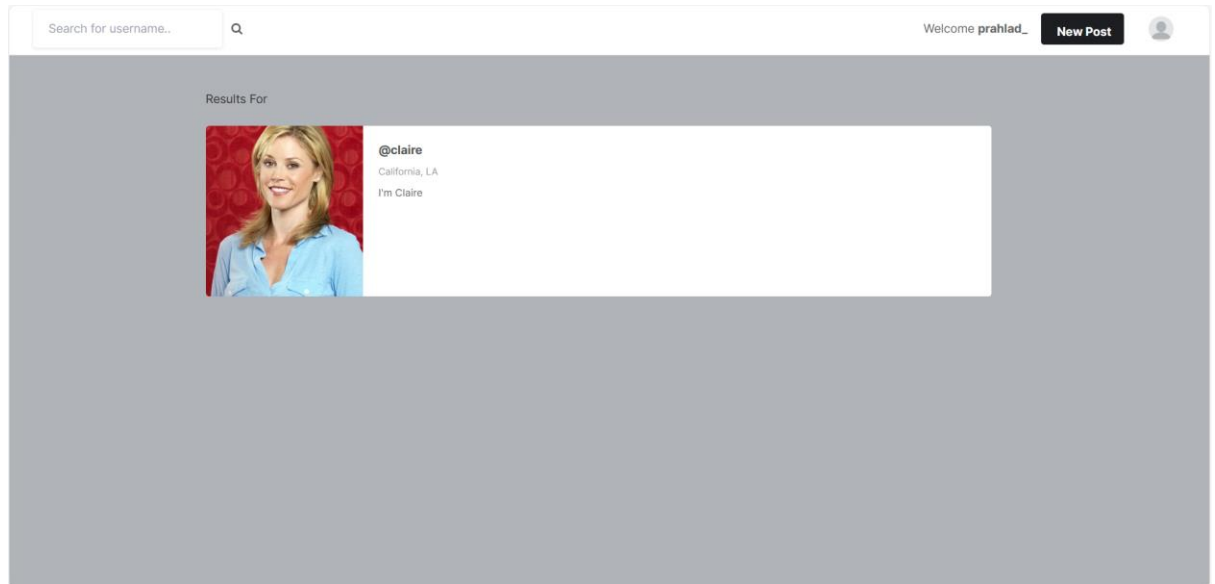


### 3.4.2 Suggestions for following

- There are a few users shown on the right side of the homepage. These are users whom you may want to follow.
- You can view their profile by clicking on the "View User" button which is right next to their username and profile picture.


### 3.4.3 Search

- There is a search bar at the top left of the homepage. Here, you can search for users by their username. Once you type in a username, you can view the search results and from there on, there is an option to further view the profile of the user that appears through the search results. An example search result is shown below.



### 3.5 Viewing Profiles

- Once you click the “view user” button or click the user profile of the user that is displayed in the search results, you are directed to the profile of that user.
- This profile page contains various details about that user such as their username, bio, number of posts, their followers and following. You can also view the pictures they have posted/uploaded.
- Additionally, there is also a button which allows you to follow that user.
- We can also view a particular post and like and download that post as well which is shown in the second pic below.



[Home](#)

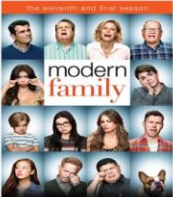
2 Posts1 follower2 following

Follow


@jay

My name is Jay Pritchett

the seventh and final season




modern family




Search for username...


Q

Upload Pictures




@Jay






Liked by 1 person



jay Singapore airport


@Jay

the seventh and final season




modern family

Suggestions for following



Clair  
I'm Clair

View User



Sam  
I'm Sam

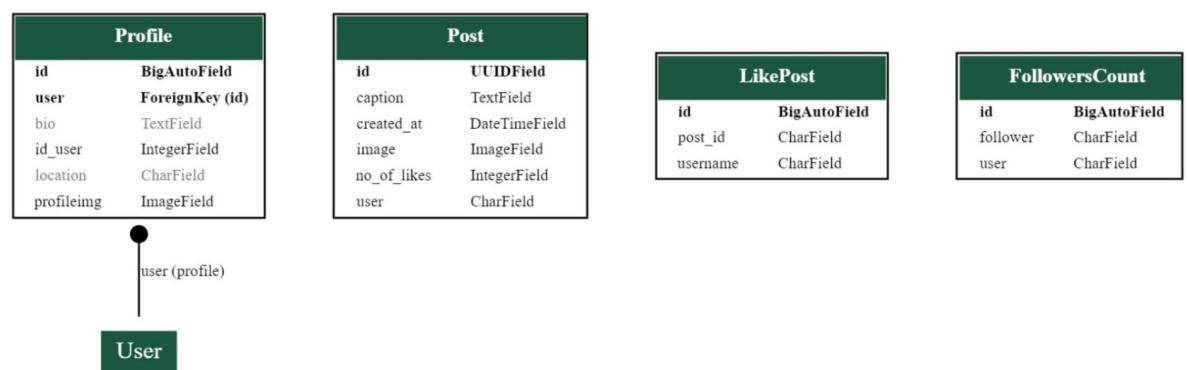
View User

## 4. Models

### 4.1 Overview of the Main Models

The application utilizes four main models which define the core entities and relationships in our Django application, facilitating user profile management, post creation, liking, and follower tracking:

- Profile
- Post
- LikePost
- FollowersCount



ER diagram for the schema

### 4.2 Description of the Profile Model

- This model represents user profiles within your application.
- It is linked to the built-in User model using a ForeignKey relationship, ensuring each profile is associated with a user.
- Attributes include:
  - **user**: ForeignKey to the User model, establishing a one-to-one relationship between users and profiles.
  - **id\_user**: An integer field representing a unique identifier for the user.
  - **bio**: A TextField allowing users to provide a brief biography.
  - **profileimg**: An ImageField for uploading profile pictures.
  - **location**: A CharField for specifying user location.

### 4.3 Description of the Post Model

- This model represents user posts within your application.
- It contains information about each post, such as the user who created it, the image associated with the post, the caption, creation timestamp, and the number of likes.
- Attributes include:
  - **id**: A UUIDField serving as the primary key for the post.
  - **user**: A CharField storing the username of the user who created the post.
  - **image**: An ImageField for uploading images associated with the post.
  - **caption**: A TextField for adding captions to posts.
  - **created\_at**: A DateTimeField indicating the timestamp when the post was created.
  - **no\_of\_likes**: An IntegerField storing the number of likes received by the post.

### 4.4 Description of the LikePost Model

- This model represents the likes given to posts by users.
- It associates a post ID with the username of the user who liked the post.
- Attributes include:
  - **post\_id**: A CharField storing the ID of the post being liked.
  - **username**: A CharField storing the username of the user who liked the post.

### 4.5 Description of the FollowersCount Model

- This model represents the count of followers for each user.
- It associates a follower with the user being followed.
- Attributes include:
  - **follower**: A CharField storing the username of the follower.
  - **user**: A CharField storing the username of the user being followed.

## 5. Views

Views in Django applications handle user requests and define the application's logic. They retrieve data from models, process user input from forms, and render the appropriate HTML templates in response. This section explores the different views used in our application.

### 5.1 Overview of the Views Used in the Application

These views collectively provide the functionality required for user authentication, profile management, post management, search, and social interactions (following, liking). Each

view corresponds to specific URLs and HTTP methods, serving different purposes within the application:

- **index:** This view is responsible for rendering the main index page of the application. It retrieves the user's profile, followed users' posts (feed), and suggestions for new users to follow.
- **settings:** This view handles user settings, allowing users to update their profile information such as bio, location, and profile picture.
- **upload:** This view is used for uploading new posts. It saves the posted image and caption to the database as a new Post object.
- **search:** This view handles user search functionality. It retrieves user profiles based on the provided search query.
- **like\_post:** This view is responsible for handling post likes. It adds or removes a like for a specific post and updates the number of likes accordingly.
- **profile:** This view displays user profiles, including user information, posts, follower/following counts, and a follow/unfollow button.
- **follow:** This view handles the follow/unfollow functionality. It allows users to follow or unfollow other users.
- **signup:** This view manages user registration. It creates a new user account, logs in the user, creates a profile for the user, and redirects to the settings page upon successful registration.
- **signin:** This view handles user authentication and login. It checks the provided credentials and logs in the user if authentication is successful.
- **logout:** This view logs out the current user and redirects to the sign-in page.

## 5.2 Description of the index View



```

def index(request):
    user_object = User.objects.get(username=request.user.username)
    user_profile = Profile.objects.get(user=user_object)

    user_following_list = []
    feed = []

    user_following = FollowersCount.objects.filter(follower=request.user.username)

    for users in user_following:
        user_following_list.append(users.user)

    for usernames in user_following_list:
        feed_lists = Post.objects.filter(user=usernames)
        feed.append(feed_lists)

    feed_list = list(chain(*feed))

    # user suggestion starts
    all_users = User.objects.all()
    user_following_all = []

    for user in user_following:
        user_list = User.objects.get(username=user.user)
        user_following_all.append(user_list)

    new_suggestions_list = [x for x in list(all_users) if (x not in list(user_following_all))]
    current_user = User.objects.filter(username=request.user.username)
    final_suggestions_list = [x for x in list(new_suggestions_list) if (x not in list(current_user))]
    random.shuffle(final_suggestions_list)

    username_profile = []
    username_profile_list = []

    for users in final_suggestions_list:
        username_profile.append(users.id)

    for ids in username_profile:
        profile_lists = Profile.objects.filter(id_user=ids)
        username_profile_list.append(profile_lists)

    suggestions_username_profile_list = list(chain(*username_profile_list))

    return render(request, 'index.html', {'user_profile': user_profile, 'posts': feed_list, 'suggestions_username_profile_list': suggestions_username_profile_list[:4]})

```

### 5.3 Description of the settings View

```

def settings(request):
    user_profile, created = Profile.objects.get_or_create(user=request.user)

    if request.method == 'POST':
        user_profile.bio = request.POST.get('bio', user_profile.bio)
        user_profile.location = request.POST.get('location', user_profile.location)
        if 'image' in request.FILES:
            user_profile.profileimg = request.FILES['image']
            user_profile.save()
        return redirect('settings')

    return render(request, 'settings.html', {'user_profile': user_profile})

```

### 5.4 Description of the uploads View

```

def upload(request):
    if request.method == 'POST':
        user = request.user.username
        image = request.FILES.get('image_upload')
        caption = request.POST.get('caption')

        new_post = Post.objects.create(user=user, image=image, caption=caption)
        new_post.save()

        return redirect('/')
    else:
        return redirect('/')

```

## 5.5 Description of the search View

```
def search(request):
    user_object = User.objects.get(username=request.user.username)
    user_profile = Profile.objects.get(user=user_object)

    if request.method == 'POST':
        username = request.POST.get('username')
        username_object = User.objects.filter(username__icontains=username)

        username_profile = []
        username_profile_list = []

        for users in username_object:
            username_profile.append(users.id)

        for ids in username_profile:
            profile_lists = Profile.objects.filter(id_user = ids)
            username_profile_list.append(profile_lists)

        username_profile_list = list(chain(*username_profile_list))

    return render(request, 'search.html', {'user_profile': user_profile, 'username_profile_list': username_profile_list})
```

## 5.6 Description of the like\_post View

```
def like_post(request):
    username = request.user.username
    post_id = request.GET.get('post_id')
    post = Post.objects.get(id=post_id)
    like_filter = LikePost.objects.filter(post_id=post_id, username=username).first()

    if like_filter == None:
        new_like = LikePost.objects.create(post_id=post_id, username=username)
        new_like.save()
        post.no_of_likes += 1
        post.save()
        return redirect('/')

    else:
        like_filter.delete()
        post.no_of_likes -= 1
        post.save()
        return redirect('/')
```

## 5.7 Description of the profile View

```

def profile(request, pk):
    user_object = get_object_or_404(User, username=pk)
    user_profile = get_object_or_404(Profile, user=user_object)
    user_posts = Post.objects.filter(user=pk)
    user_post_length = len(user_posts)

    follower = request.user.username
    user = pk

    # Initialize these variables outside the if-else block
    user_followers = len(FollowersCount.objects.filter(user=pk))
    user_following = len(FollowersCount.objects.filter(follower=pk))

    if FollowersCount.objects.filter(follower=follower, user=user).first():
        button_text = 'Unfollow'
    else:
        button_text = 'Follow'

    context = {
        'user_object': user_object,
        'user_profile': user_profile,
        'user_posts': user_posts,
        'user_post_length': user_post_length,
        'button_text': button_text,
        'user_followers': user_followers,
        'user_following': user_following,
    }
    return render(request, 'profile.html', context)

```

## 5.8 Description of the follow View

```

def follow(request):
    if request.method == 'POST':
        follower = request.POST.get('follower')
        user = request.POST.get('user')

        if FollowersCount.objects.filter(follower=follower, user=user).first():
            delete_follower = FollowersCount.objects.get(follower=follower, user=user)
            delete_follower.delete()
            return redirect('/profile/' + user)
        else:
            new_follower = FollowersCount.objects.create(follower=follower, user=user)
            new_follower.save()
            return redirect('/profile/' + user)
    else:
        return redirect('/')

```

## 5.9 Description of the signup View

```

def signup(request):
    if request.method == "POST":
        username = request.POST.get('username')
        email = request.POST.get('email')
        password = request.POST.get('password')
        password2 = request.POST.get('password2')

        if password != password2:
            messages.info(request, "Passwords are not matching")
            return redirect('signup')

        if User.objects.filter(email=email).exists():
            messages.info(request, 'Email Taken')
            return redirect('signup')

        if User.objects.filter(username=username).exists():
            messages.info(request, "Username Taken")
            return redirect('signup')

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

        #log in user and redirect to the settings page
        user_login = auth.authenticate(username=username,password=password)
        auth.login(request,user_login)

        #Create a profile for the user
        user_model = User.objects.get(username=username)
        new_profile = Profile.objects.create(user=user_model,id_user=user_model.id)
        new_profile.save()
        # You might want to redirect to a login page or to the homepage with a success message
        messages.success(request, "User created successfully")
        return redirect('settings') # Assuming you have a 'login' route defined

    else:
        return render(request, 'signup.html')

```

## 5.10 Description of the signin View

```

def signin(request):
    if request.method == 'POST':
        username = request.POST.get('username')
        password = request.POST.get('password')
        user = auth.authenticate(username=username, password=password)

        if user is not None:
            auth.login(request, user)
            return redirect('/')
        else:
            # If authentication fails, inform the user and render the same signin page
            messages.info(request, "Invalid Credentials")
            return redirect(signin) # Ensure this return here to handle failed authentication

    else:
        # This handles GET requests and any other methods that might be used to access the page
        return render(request, 'signin.html')

```

## 5.11 Description of the logout View

```
def logout(request):  
    auth.logout(request)  
    return redirect('signin')
```

## 7. Conclusion

### 7.1 Summary of Project

The Django-based social media app provides users with a platform to create accounts, customize profiles, share posts, and engage with other users through features like following and liking posts. With a focus on user interaction and content discovery, the app facilitates seamless social networking while offering a visually appealing and responsive interface. This report has documented the key functionalities and technical aspects of the application.

#### Key Functionalities:

##### 1. User Management:

- Account creation and authentication.
- User profile customization (bio, location, profile picture).

##### 2. Content Sharing:

- Posting images.

##### 3. Social Interactions:

- Following and unfollowing other users.
- Viewing profiles of other users.

##### 4. Search Functionality:

- Searching for users based on usernames.

##### 5. Profile Management:

- **Editing profile information such as bio and location.**
- **Uploading profile pictures.**

## 6. Responsive Design:

- **Utilizing HTML, CSS (Tailwind CSS, Bootstrap), and JavaScript to ensure a visually appealing and user-friendly interface.**

## Technical Stack:

- **Frontend:** HTML, Bootstrap, Tailwind CSS, JavaScript
- **Backend:** Django (Python web framework) with SQLite3 database

## 7.2 Future Prospects and Improvements

Our application has a strong foundation and potential for further development and feature enhancements. Here are some future considerations:

- **Community Building Tools:** Introducing features like groups, events, and forums to foster community engagement and facilitate discussions among users with shared interests.
- **Scalability and Performance Optimization:** Optimizing database queries, caching strategies, and server-side processing to handle increased traffic and ensure smooth performance as the user base grows.
- **Accessibility and Internationalization:** Improving accessibility features and supporting multiple languages to make the app more inclusive and accessible to a diverse user base.
- **Analytics and Insights:** Providing users and administrators with analytics dashboards and insights to track engagement metrics, audience demographics, and post-performance.
- **Content Moderation:** Introducing content moderation tools and user reporting mechanisms to maintain a safe and inclusive community environment.

Implementing these prospects and improvements will help transform our application into a more engaging and robust social media platform. These new features will expand the app's functionality and revenue potential, ultimately fostering a thriving and vibrant community.