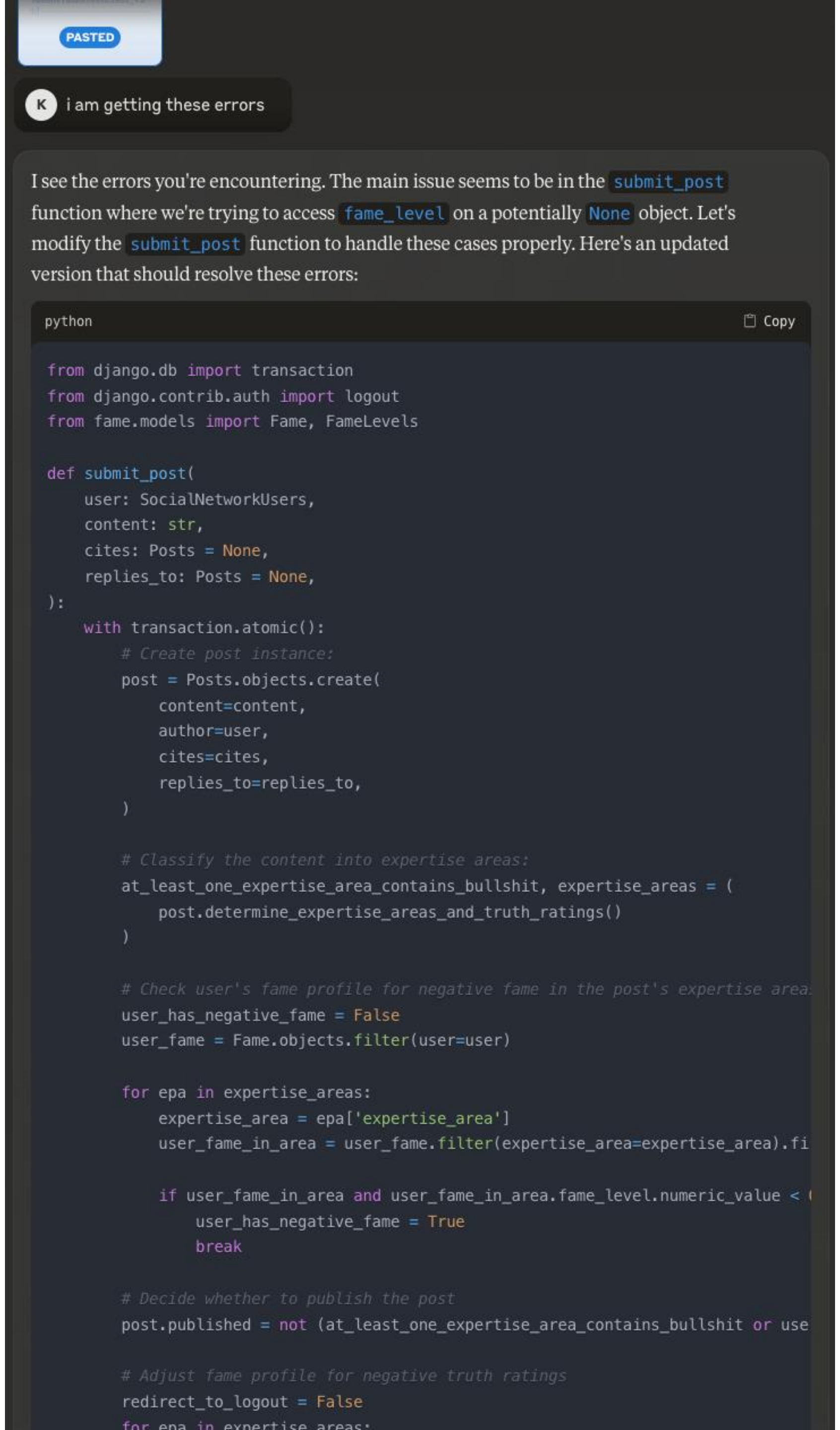
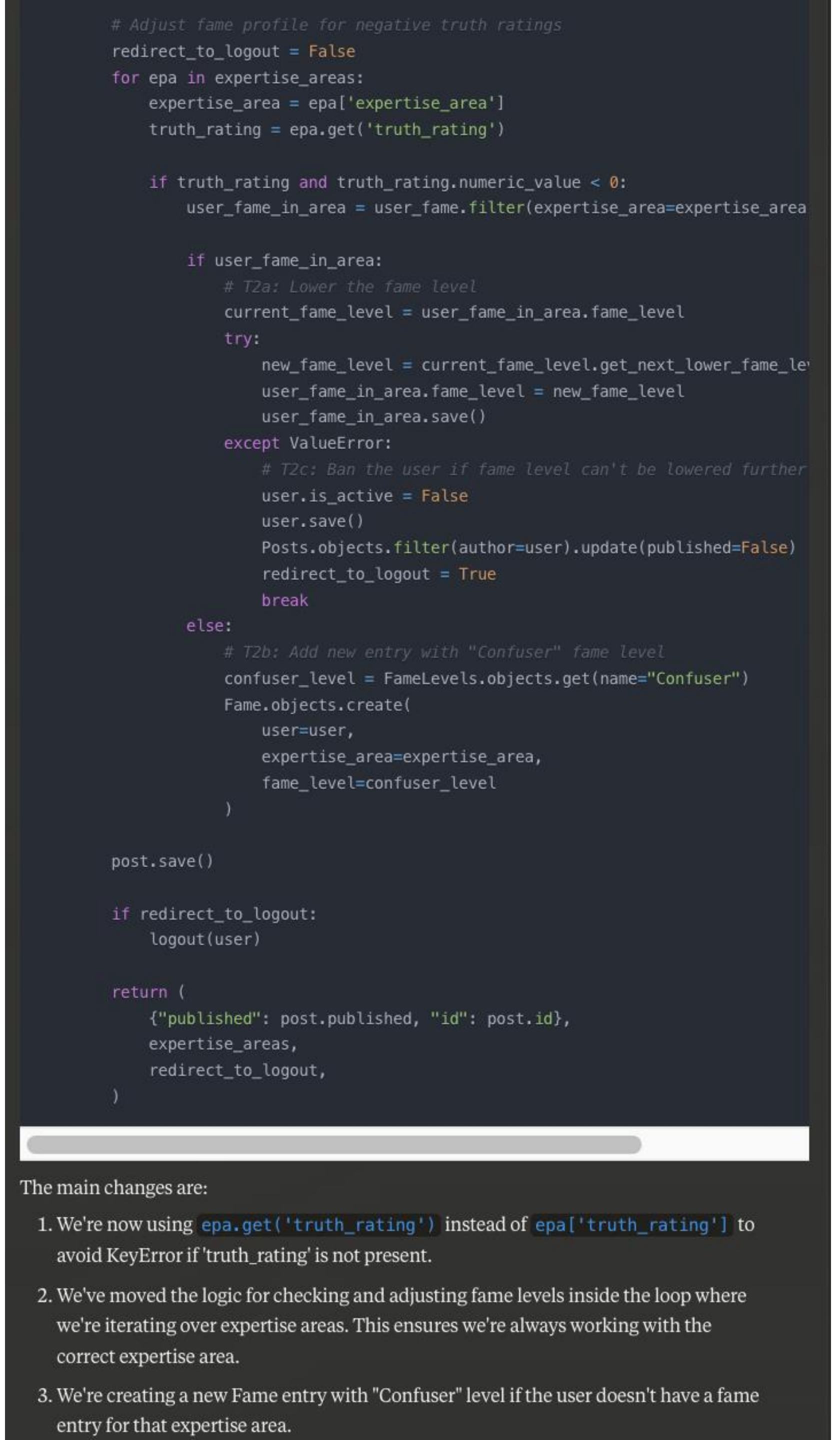


i am getting these errors

I see the errors you're encountering. The main issue seems to be in the `submit_post` function where we're trying to access `fame_level` on a potentially `None` object. Let's modify the `submit_post` function to handle these cases properly. Here's an updated version that should resolve these errors:

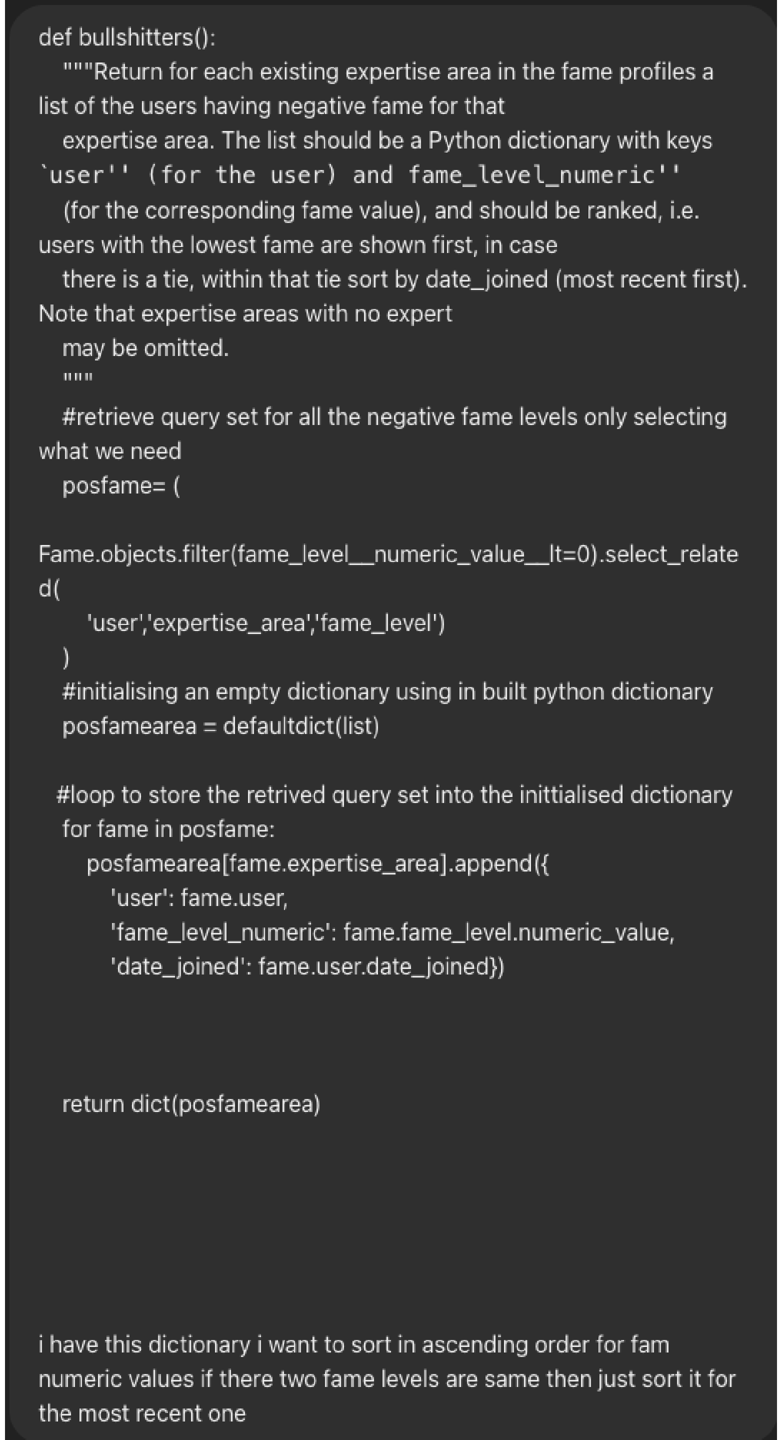
```python
from django.db import transaction
from django.contrib.auth import logout
from fame.models import Fame, FameLevels

def submit_post(
    user: SocialNetworkUsers,
    content: str,
    cites: Posts = None,
    replies_to: Posts = None,
):
    with transaction.atomic():
        # Create post instance:
        post = Posts.objects.create(
            content=content,
            author=user,
            cites=cites,
            replies_to=replies_to,
        )

        # Classify the content into expertise areas:
        at_least_one_expertise_area_contains_bullshit, expertise_areas = (
            post.determine_expertise_areas_and_truth_ratings()
        )

        # Check user's fame profile for negative fame in the post's expertise area
        user_has_negative_fame = False
        user_fame = Fame.objects.filter(user=user)

        for epa in expertise_areas:
            expertise_area = epa['expertise_area']
            user_fame_in_area = user_fame.filter(expertise_area=expertise_area).fi

            if user_fame_in_area and user_fame_in_area.fame_level.numeric_value <
                user_has_negative_fame = True
                break

        # Decide whether to publish the post
        post.published = not (at_least_one_expertise_area_contains_bullshit or use

        # Adjust fame profile for negative truth ratings
        redirect_to_logout = False
        for epa in expertise_areas:
```



```python
        # Adjust fame profile for negative truth ratings
        redirect_to_logout = False
        for epa in expertise_areas:
            expertise_area = epa['expertise_area']
            truth_rating = epa.get('truth_rating')

            if truth_rating and truth_rating.numeric_value < 0:
                user_fame_in_area = user_fame.filter(expertise_area=expertise_area

                if user_fame_in_area:
                    # T2a: Lower the fame level
                    current_fame_level = user_fame_in_area.fame_level
                    try:
                        new_fame_level = current_fame_level.get_next_lower_fame_le
                        user_fame_in_area.fame_level = new_fame_level
                        user_fame_in_area.save()
                    except ValueError:
                        # T2c: Ban the user if fame level can't be lowered further
                        user.is_active = False
                        user.save()
                        Posts.objects.filter(author=user).update(published=False)
                        redirect_to_logout = True
                        break
                else:
                    # T2b: Add new entry with "Confuser" fame level
                    confuser_level = FameLevels.objects.get(name="Confuser")
                    Fame.objects.create(
                        user=user,
                        expertise_area=expertise_area,
                        fame_level=confuser_level
                    )

        post.save()

        if redirect_to_logout:
            logout(user)

        return (
            {"published": post.published, "id": post.id},
            expertise_areas,
            redirect_to_logout,
        )
```

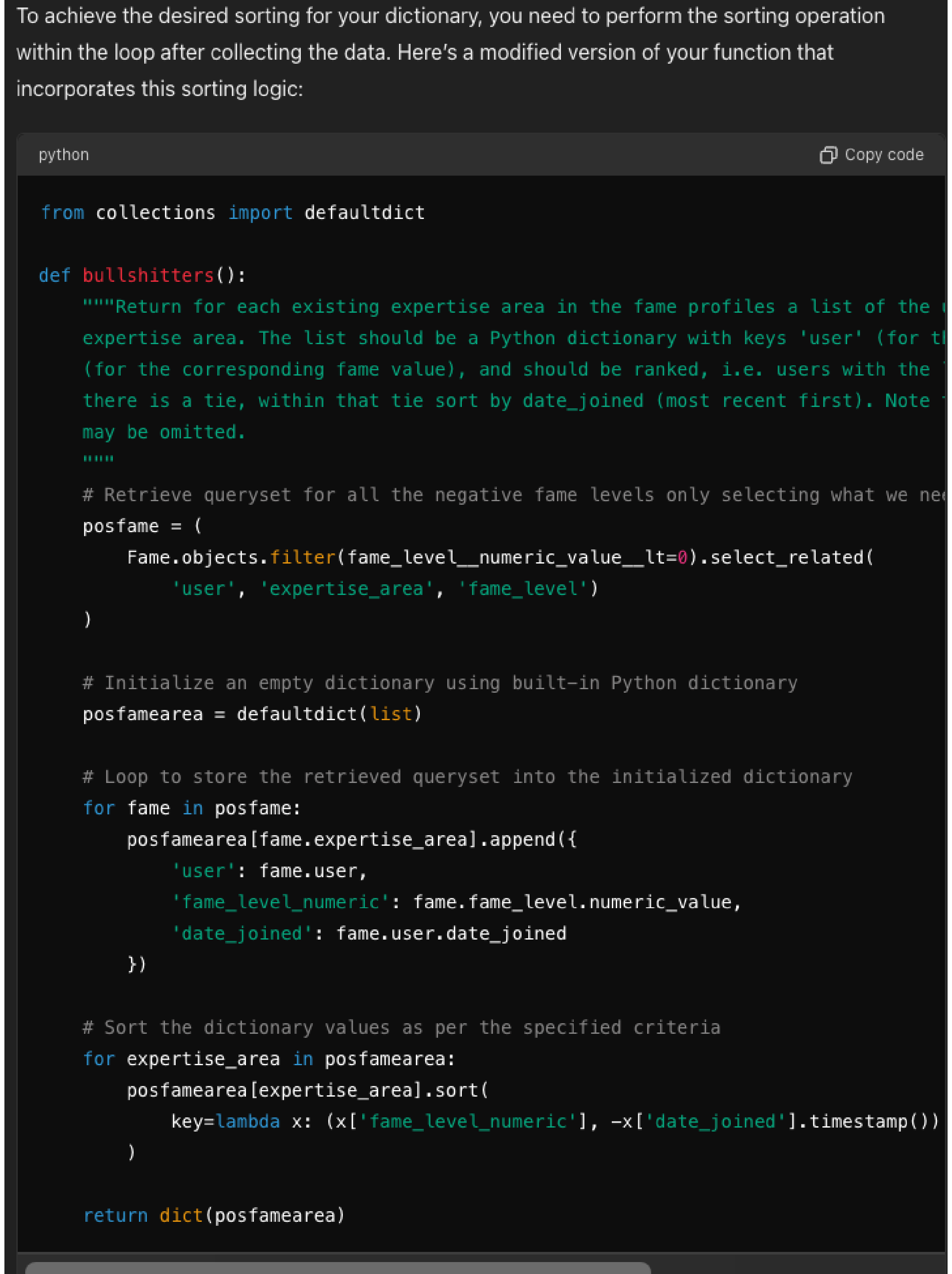
The main changes are:

1. We're now using `epa.get('truth_rating')` instead of `epa['truth_rating']` to avoid KeyError if 'truth_rating' is not present.
2. We've moved the logic for checking and adjusting fame levels inside the loop where we're iterating over expertise areas. This ensures we're always working with the correct expertise area.
3. We're creating a new Fame entry with "Confuser" level if the user doesn't have a fame entry for that expertise area.



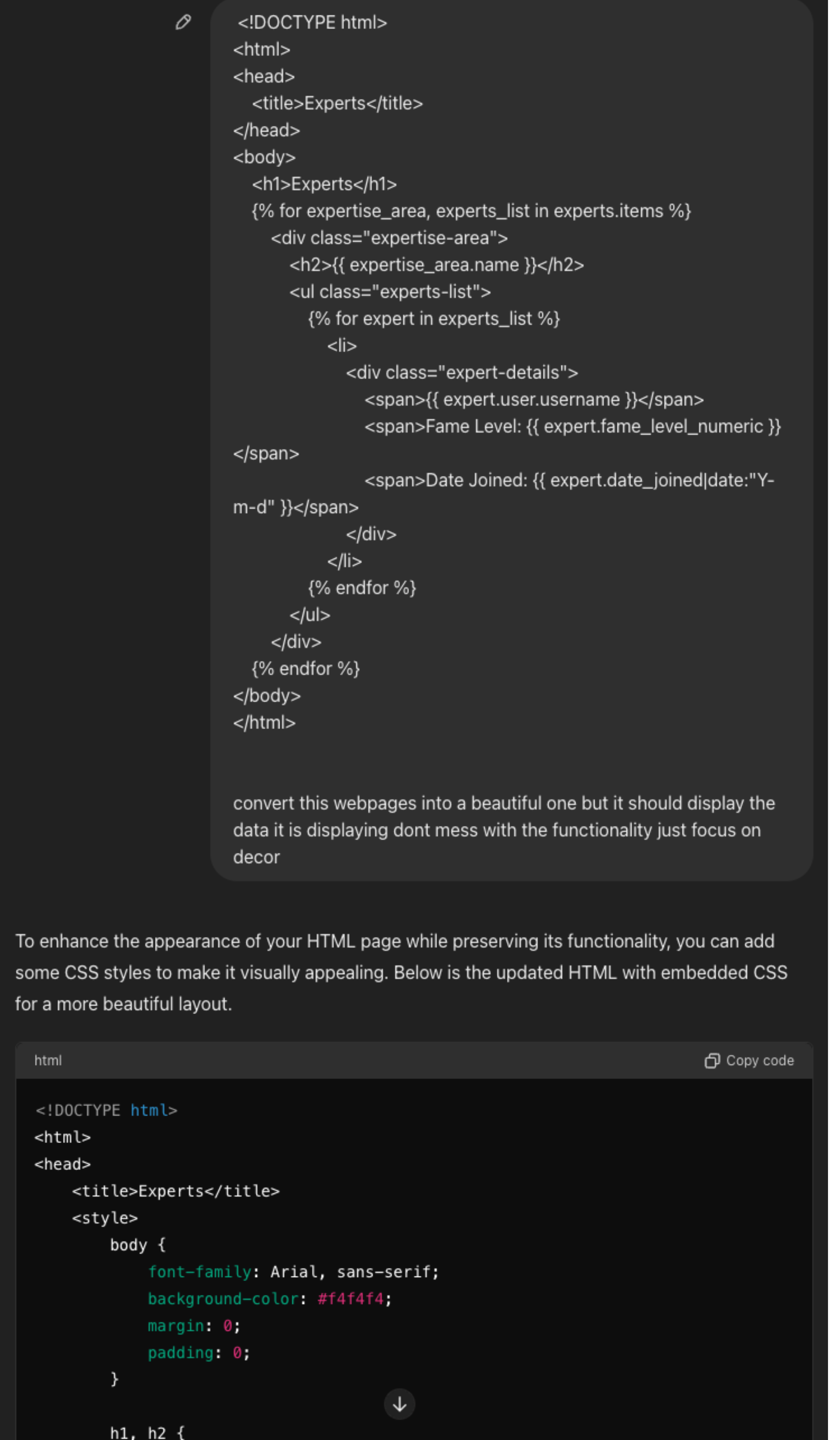
```
def bullshitters():
    """Return for each existing expertise area in the fame profiles a
list of the users having negative fame for that
    expertise area. The list should be a Python dictionary with keys
`user'' (for the user) and fame_level_numeric''
    (for the corresponding fame value), and should be ranked, i.e.
users with the lowest fame are shown first, in case
    there is a tie, within that tie sort by date_joined (most recent first).
Note that expertise areas with no expert
    may be omitted.
    """
    #retrieve query set for all the negative fame levels only selecting
what we need
    posfame= (

Fame.objects.filter(fame_level__numeric_value__lt=0).select_relate
d(
        'user','expertise_area','fame_level')
    )
    #initialising an empty dictionary using in built python dictionary
    posfamearea = defaultdict(list)

   #loop to store the retrived query set into the inittialised dictionary
    for fame in posfame:
        posfamearea[fame.expertise_area].append({
            'user': fame.user,
            'fame_level_numeric': fame.fame_level.numeric_value,
            'date_joined': fame.user.date_joined})



    return dict(posfamearea)
```

i have this dictionary i want to sort in ascending order for fam numeric values if there two fame levels are same then just sort it for the most recent one



To achieve the desired sorting for your dictionary, you need to perform the sorting operation within the loop after collecting the data. Here's a modified version of your function that incorporates this sorting logic:

```python
from collections import defaultdict

def bullshitters():
    """Return for each existing expertise area in the fame profiles a list of the
    expertise area. The list should be a Python dictionary with keys 'user' (for
    (for the corresponding fame value), and should be ranked, i.e. users with the
    there is a tie, within that tie sort by date_joined (most recent first). Note
    may be omitted.
    """
    # Retrieve queryset for all the negative fame levels only selecting what we ne
    posfame = (
        Fame.objects.filter(fame_level__numeric_value__lt=0).select_related(
            'user', 'expertise_area', 'fame_level')
    )

    # Initialize an empty dictionary using built-in Python dictionary
    posfamearea = defaultdict(list)

    # Loop to store the retrieved queryset into the initialized dictionary
    for fame in posfame:
        posfamearea[fame.expertise_area].append({
            'user': fame.user,
            'fame_level_numeric': fame.fame_level.numeric_value,
            'date_joined': fame.user.date_joined
        })

    # Sort the dictionary values as per the specified criteria
    for expertise_area in posfamearea:
        posfamearea[expertise_area].sort(
            key=lambda x: (x['fame_level_numeric'], -x['date_joined'].timestamp())
        )

    return dict(posfamearea)
```

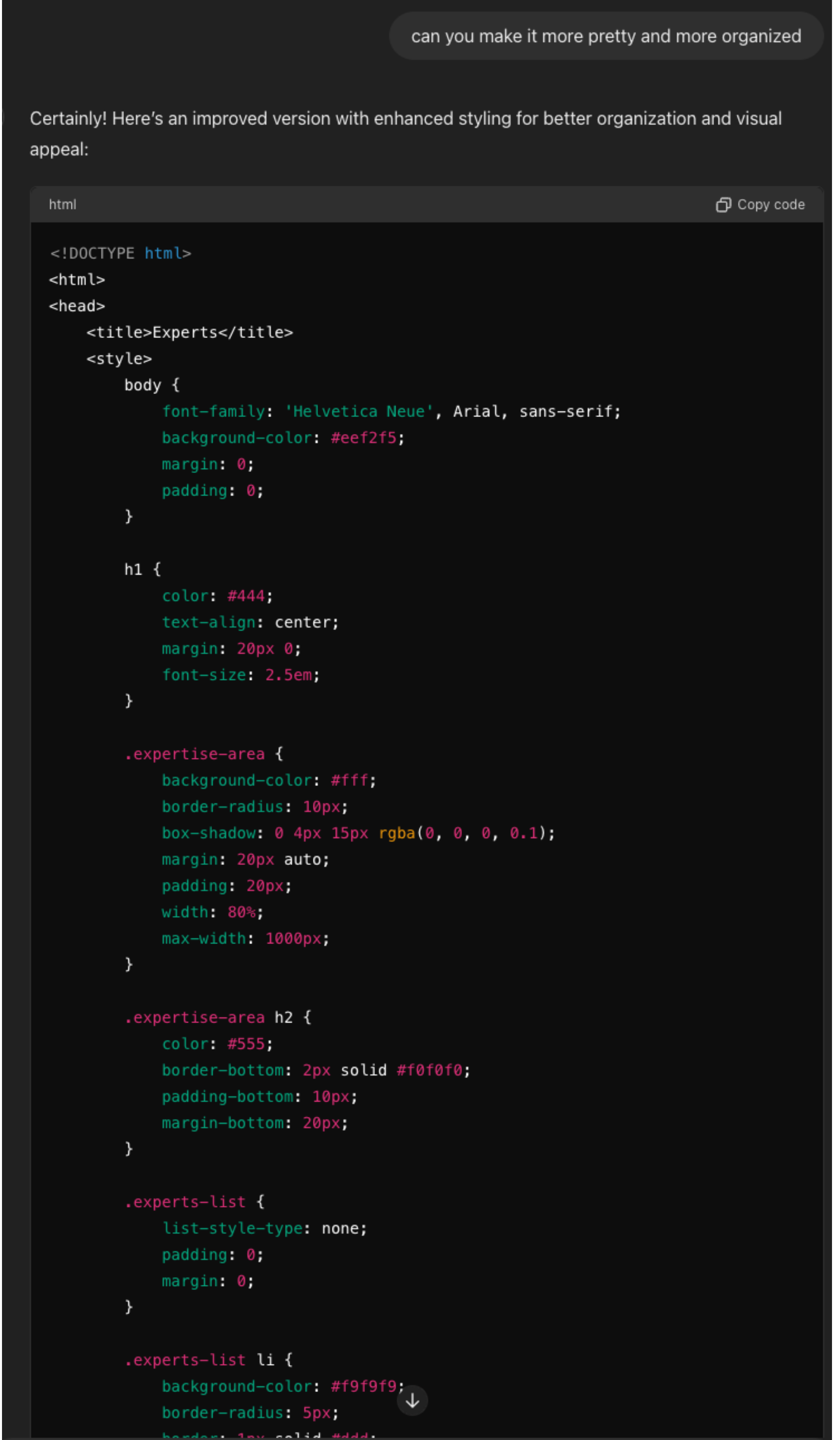


```
<!DOCTYPE html>
<html>
<head>
  <title>Experts</title>
</head>
<body>
  <h1>Experts</h1>
  {% for expertise_area, experts_list in experts.items %}
    <div class="expertise-area">
      <h2>{{ expertise_area.name }}</h2>
      <ul class="experts-list">
        {% for expert in experts_list %}
          <li>
            <div class="expert-details">
              <span>{{ expert.user.username }}</span>
              <span>Fame Level: {{ expert.fame_level_numeric }}
</span>
              <span>Date Joined: {{ expert.date_joined|date:"Y-
m-d" }}</span>
            </div>
          </li>
        {% endfor %}
      </ul>
    </div>
  {% endfor %}
</body>
</html>
```

convert this webpages into a beautiful one but it should display the data it is displaying dont mess with the functionality just focus on decor

To enhance the appearance of your HTML page while preserving its functionality, you can add some CSS styles to make it visually appealing. Below is the updated HTML with embedded CSS for a more beautiful layout.

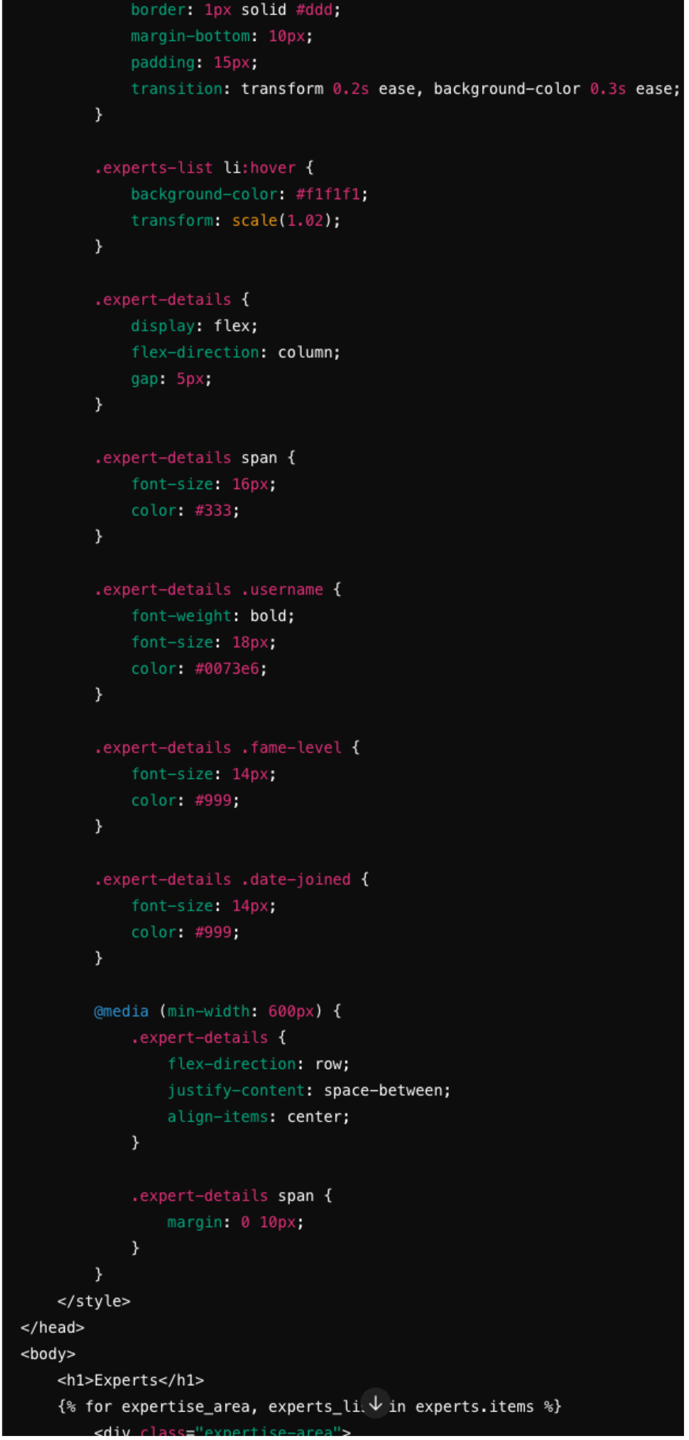
```html
<!DOCTYPE html>
<html>
<head>
    <title>Experts</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            background-color: #f4f4f4;
            margin: 0;
            padding: 0;
        }

        h1, h2 {
```



can you make it more pretty and more organized

Certainly! Here's an improved version with enhanced styling for better organization and visual appeal:

```html
<!DOCTYPE html>
<html>
<head>
    <title>Experts</title>
    <style>
        body {
            font-family: 'Helvetica Neue', Arial, sans-serif;
            background-color: #eef2f5;
            margin: 0;
            padding: 0;
        }

        h1 {
            color: #444;
            text-align: center;
            margin: 20px 0;
            font-size: 2.5em;
        }

        .expertise-area {
            background-color: #fff;
            border-radius: 10px;
            box-shadow: 0 4px 15px rgba(0, 0, 0, 0.1);
            margin: 20px auto;
            padding: 20px;
            width: 80%;
            max-width: 1000px;
        }

        .expertise-area h2 {
            color: #555;
            border-bottom: 2px solid #f0f0f0;
            padding-bottom: 10px;
            margin-bottom: 20px;
        }

        .experts-list {
            list-style-type: none;
            padding: 0;
            margin: 0;
        }

        .experts-list li {
            background-color: #f9f9f9;
            border-radius: 5px;
            border: 1px solid #ddd;
```



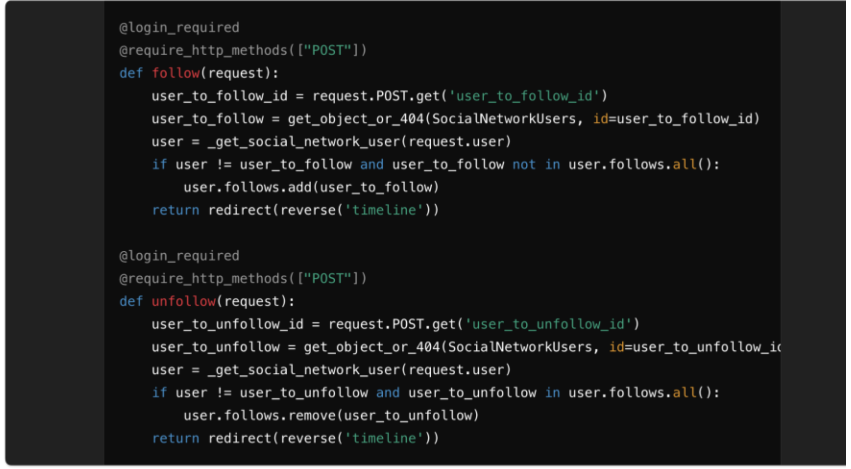
```html
            border: 1px solid #ddd;
            margin-bottom: 10px;
            padding: 15px;
            transition: transform 0.2s ease, background-color 0.3s ease;
        }

        .experts-list li:hover {
            background-color: #f1f1f1;
            transform: scale(1.02);
        }

        .expert-details {
            display: flex;
            flex-direction: column;
            gap: 5px;
        }

        .expert-details span {
            font-size: 16px;
            color: #333;
        }

        .expert-details .username {
            font-weight: bold;
            font-size: 18px;
            color: #0073e6;
        }

        .expert-details .fame-level {
            font-size: 14px;
            color: #999;
        }

        .expert-details .date-joined {
            font-size: 14px;
            color: #999;
        }

        @media (min-width: 600px) {
            .expert-details {
                flex-direction: row;
                justify-content: space-between;
                align-items: center;
            }

            .expert-details span {
                margin: 0 10px;
            }
        }
    </style>
</head>
<body>
    <h1>Experts</h1>
    {% for expertise_area, experts_li in experts.items %}
        <div class="expertise-area">
```

```
            }

            .expert-details span {
                margin: 0 10px;
            }
        }
    </style>
</head>
<body>
    <h1>Experts</h1>
    {% for expertise_area, experts_list in experts.items %}
        <div class="expertise-area">
            <h2>{{ expertise_area.name }}</h2>
            <ul class="experts-list">
                {% for expert in experts_list %}
                    <li>
                        <div class="expert-details">
                            <span class="username">{{ expert.user.username }}</spa
                            <span class="fame-level">Fame Level: {{ expert.fame_le
                            <span class="date-joined">Date Joined: {{ expert.date_
                        </div>
                    </li>
                {% endfor %}
            </ul>
        </div>
    {% endfor %}
</body>
</html>
```

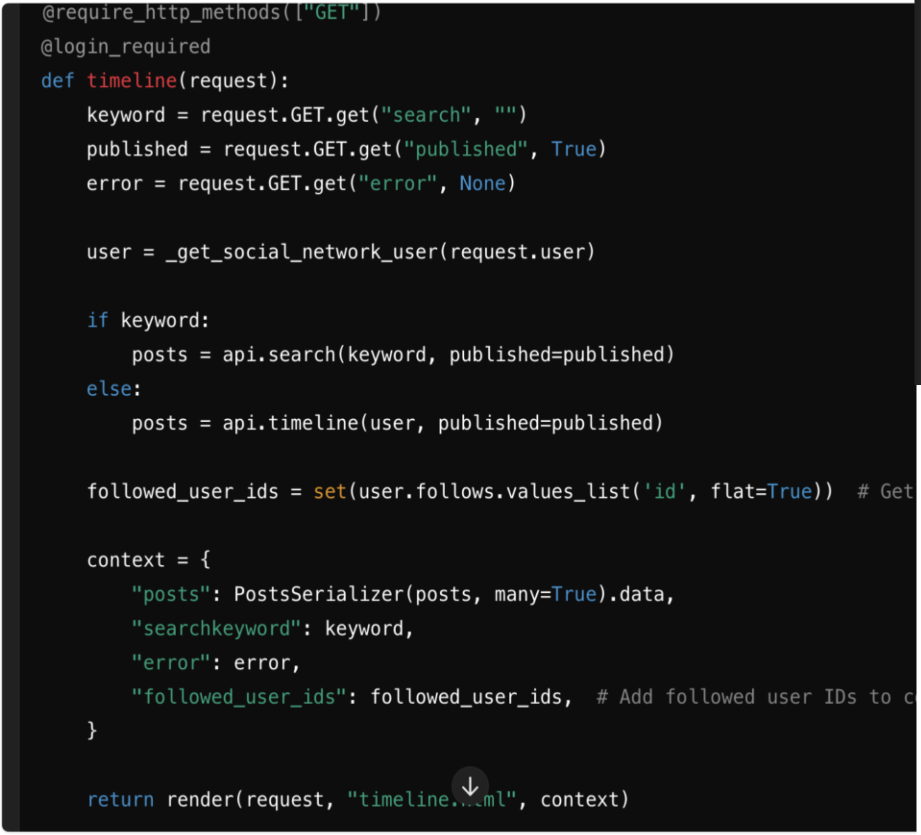
Enhancements:

1. **Typography and Colors**: Updated the font and colors to make the text more appealing and readable.
2. **Layout Improvements**: Added maximum width to the expertise areas to ensure they don't become too wide on large screens.
3. **Responsive Design**: Improved the layout for both mobile and desktop views by using media queries.
4. **Hover Effects**: Added subtle hover effects to list items to make the interaction feel more dynamic.
5. **Spacing and Organization**: Enhanced the spacing and organization of elements to create a clean, modern look.

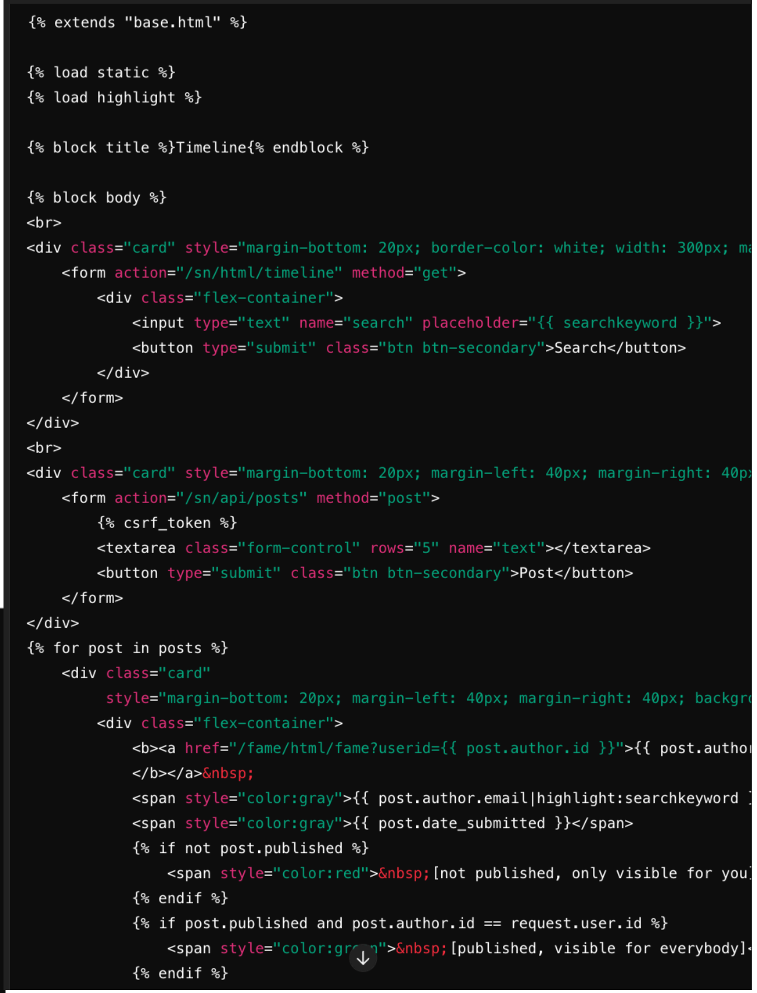
These changes should provide a more polished and organized appearance for your webpage while maintaining the original functionality.



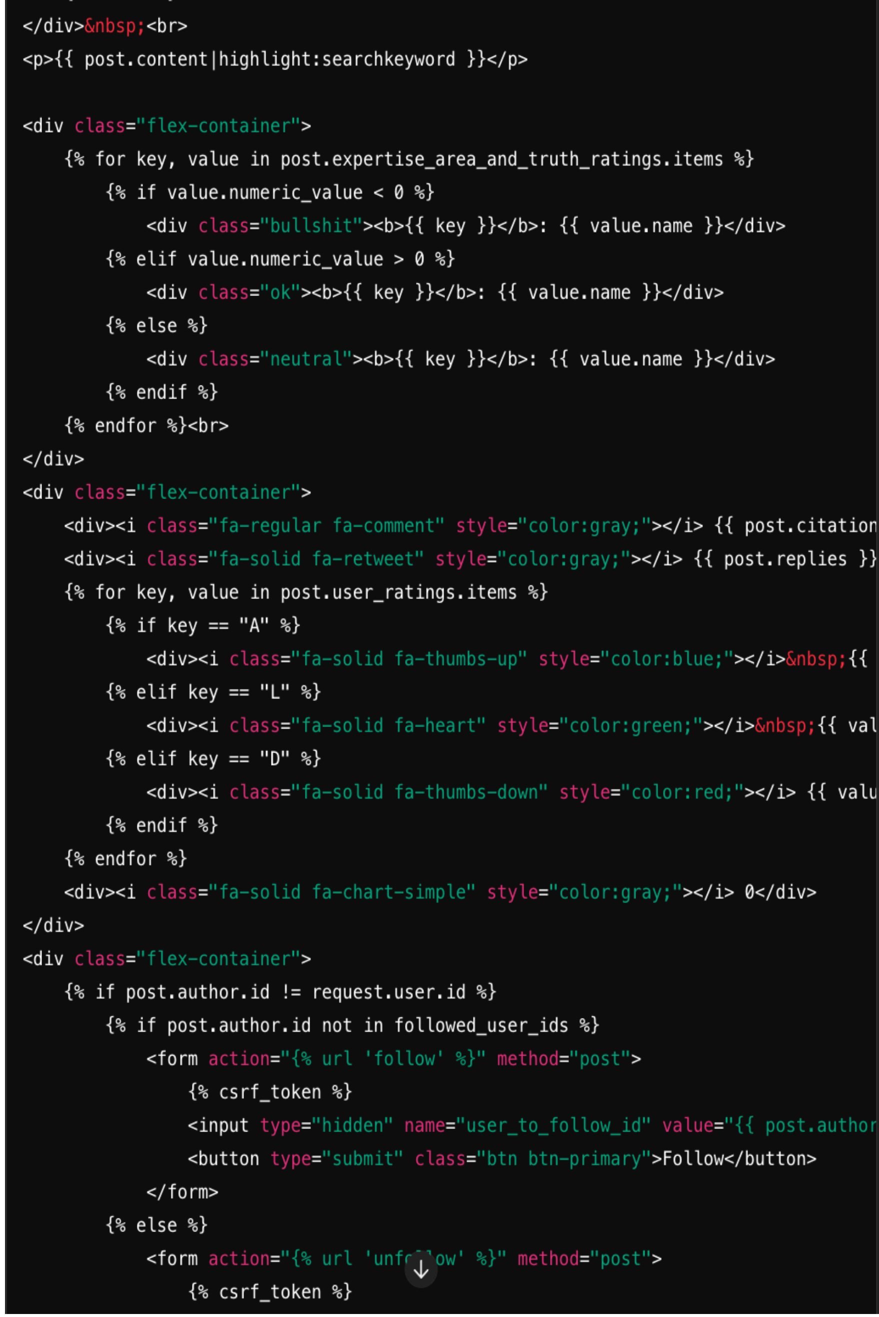
```python
@login_required
@require_http_methods(["POST"])
def follow(request):
    user_to_follow_id = request.POST.get('user_to_follow_id')
    user_to_follow = get_object_or_404(SocialNetworkUsers, id=user_to_follow_id)
    user = _get_social_network_user(request.user)
    if user != user_to_follow and user_to_follow not in user.follows.all():
        user.follows.add(user_to_follow)
    return redirect(reverse('timeline'))

@login_required
@require_http_methods(["POST"])
def unfollow(request):
    user_to_unfollow_id = request.POST.get('user_to_unfollow_id')
    user_to_unfollow = get_object_or_404(SocialNetworkUsers, id=user_to_unfollow_i
    user = _get_social_network_user(request.user)
    if user != user_to_unfollow and user_to_unfollow in user.follows.all():
        user.follows.remove(user_to_unfollow)
    return redirect(reverse('timeline'))
```



```python
@require_http_methods(["GET"])
@login_required
def timeline(request):
    keyword = request.GET.get("search", "")
    published = request.GET.get("published", True)
    error = request.GET.get("error", None)

    user = _get_social_network_user(request.user)

    if keyword:
        posts = api.search(keyword, published=published)
    else:
        posts = api.timeline(user, published=published)

    followed_user_ids = set(user.follows.values_list('id', flat=True))  # Get

    context = {
        "posts": PostsSerializer(posts, many=True).data,
        "searchkeyword": keyword,
        "error": error,
        "followed_user_ids": followed_user_ids,  # Add followed user IDs to c
    }

    return render(request, "timeline.html", context)
```



```html
{% extends "base.html" %}

{% load static %}
{% load highlight %}

{% block title %}Timeline{% endblock %}

{% block body %}
<br>
<div class="card" style="margin-bottom: 20px; border-color: white; width: 300px; ma
    <form action="/sn/html/timeline" method="get">
        <div class="flex-container">
            <input type="text" name="search" placeholder="{{ searchkeyword }}">
            <button type="submit" class="btn btn-secondary">Search</button>
        </div>
    </form>
</div>
<br>
<div class="card" style="margin-bottom: 20px; margin-left: 40px; margin-right: 40p
    <form action="/sn/api/posts" method="post">
        {% csrf_token %}
        <textarea class="form-control" rows="5" name="text"></textarea>
        <button type="submit" class="btn btn-secondary">Post</button>
    </form>
</div>
{% for post in posts %}
    <div class="card"
         style="margin-bottom: 20px; margin-left: 40px; margin-right: 40px; backgr
        <div class="flex-container">
            <b><a href="/fame/html/fame?userid={{ post.author.id }}">{{ post.autho
            </b></a>&nbsp;
            <span style="color:gray">{{ post.author.email|highlight:searchkeyword
            <span style="color:gray">{{ post.date_submitted }}</span>
            {% if not post.published %}
                <span style="color:red">&nbsp;[not published, only visible for you]
            {% endif %}
            {% if post.published and post.author.id == request.user.id %}
                <span style="color:green">&nbsp;[published, visible for everybody]<
            {% endif %}
```



```html
</div>&nbsp;<br>
<p>{{ post.content|highlight:searchkeyword }}</p>

<div class="flex-container">
    {% for key, value in post.expertise_area_and_truth_ratings.items %}
        {% if value.numeric_value < 0 %}
            <div class="bullshit"><b>{{ key }}</b>: {{ value.name }}</div>
        {% elif value.numeric_value > 0 %}
            <div class="ok"><b>{{ key }}</b>: {{ value.name }}</div>
        {% else %}
            <div class="neutral"><b>{{ key }}</b>: {{ value.name }}</div>
        {% endif %}
    {% endfor %}<br>
</div>
<div class="flex-container">
    <div><i class="fa-regular fa-comment" style="color:gray;"></i> {{ post.citation
    <div><i class="fa-solid fa-retweet" style="color:gray;"></i> {{ post.replies }}
    {% for key, value in post.user_ratings.items %}
        {% if key == "A" %}
            <div><i class="fa-solid fa-thumbs-up" style="color:blue;"></i>&nbsp;{{
        {% elif key == "L" %}
            <div><i class="fa-solid fa-heart" style="color:green;"></i>&nbsp;{{ val
        {% elif key == "D" %}
            <div><i class="fa-solid fa-thumbs-down" style="color:red;"></i> {{ valu
        {% endif %}
    {% endfor %}
    <div><i class="fa-solid fa-chart-simple" style="color:gray;"></i> 0</div>
</div>
<div class="flex-container">
    {% if post.author.id != request.user.id %}
        {% if post.author.id not in followed_user_ids %}
            <form action="{% url 'follow' %}" method="post">
                {% csrf_token %}
                <input type="hidden" name="user_to_follow_id" value="{{ post.author
                <button type="submit" class="btn btn-primary">Follow</button>
            </form>
        {% else %}
            <form action="{% url 'unfollow' %}" method="post">
                {% csrf_token %}
```