Nomad Narratives
PEP8 Validation

The following is a series of screenshots from the CI Python Linter broken down by relevant apps. Flake8 and Pylint were used throughout development to highlight PEP8 issues as they arose.

Comments App

admin.py - Passed

```python
'''
Django admin configuration for the Comment model.
'''
from django.contrib import admin
from .models import Comment

admin.site.register(Comment)

```

Settings:

Results:

All clear, no errors found

Comments App

apps.py - Passed

```python
'''
App configuration for the Comments application.

This module defines the configuration settings for the Comments app
within the Django project.
'''
from django.apps import AppConfig


class CommentsConfig(AppConfig):
    '''
    This class sets the default primary key type and specifies the app name.
    '''
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'comments'
```

Settings:

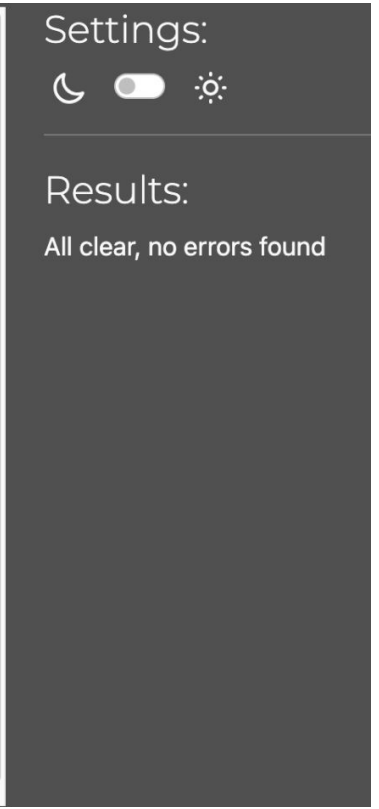
Results:

All clear, no errors found

Comments App

models.py - Passed

```python
'''
Represents a comment made by a user on a specific trip post.
'''

from django.db import models
from django.contrib.auth.models import User
from posts.models import TripPost


class Comment(models.Model):
    '''
    Comment model, related to User and Post.

    '''
    owner = models.ForeignKey(User, on_delete=models.CASCADE)
    post = models.ForeignKey(TripPost, on_delete=models.CASCADE)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
    content = models.TextField()

    class Meta:
        '''
        Orders comments by creation time in descending order.
        '''
        ordering = ['-created_at']

    def __str__(self):
        '''
        Returns a string representation of the comment's content.
        '''
        return str(self.content)
```



Comments App

serializers.py - Passed

```python
from rest_framework import serializers
from django.contrib.humanize.templatetags.humanize import naturaltime
from .models import Comment


class CommentSerializer(serializers.ModelSerializer):
    '''
    Serializer for Comment model.
    '''
    owner = serializers.ReadOnlyField(source='owner.username')
    is_owner = serializers.SerializerMethodField()
    profile_id = serializers.ReadOnlyField(source='owner.profile.id')
    profile_image = serializers.ReadOnlyField(source='owner.profile.image.url')
    created_at = serializers.SerializerMethodField()
    updated_at = serializers.SerializerMethodField()

    def get_is_owner(self, obj):
        '''
        Checks for owner of comment.
        '''
        request = self.context['request']
        return request.user == obj.owner

    def get_created_at(self, obj):
        '''
        Returns as human readable time.
        '''
        return naturaltime(obj.created_at)

    def get_updated_at(self, obj):
        '''
```

Settings:

Results:

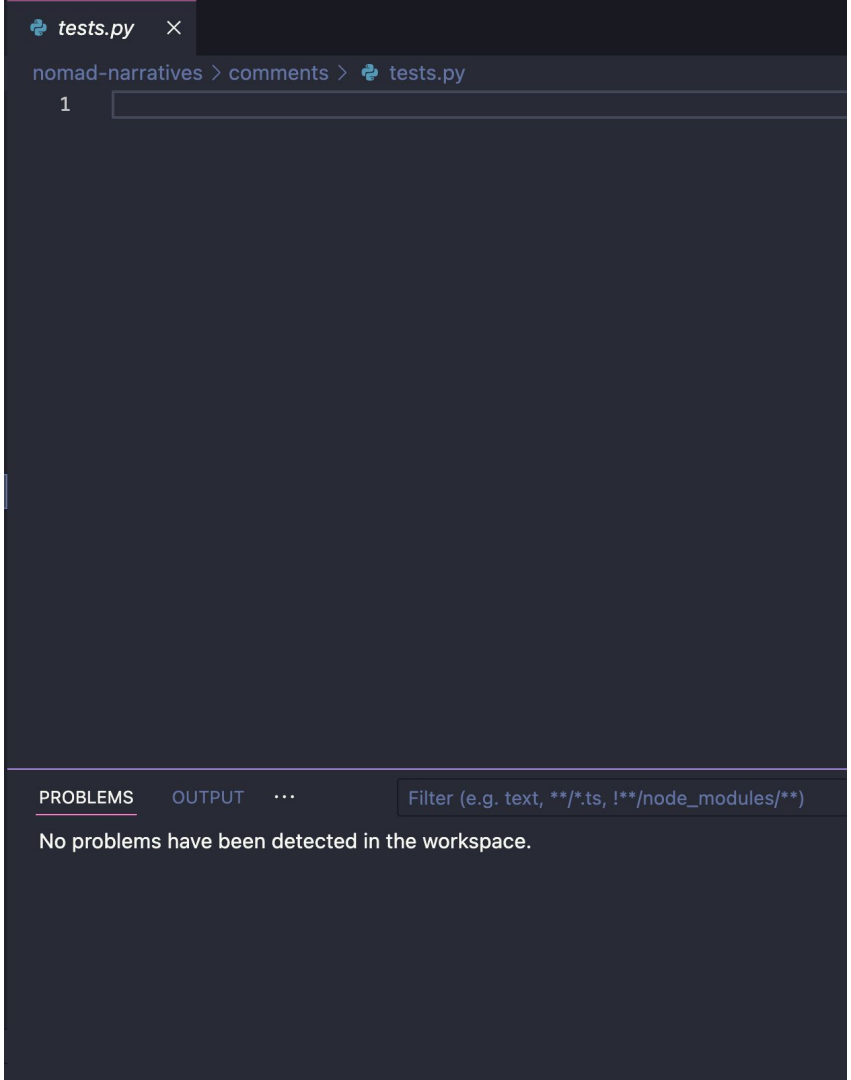
All clear, no errors found

Comments App

tests.py

Empty file -

Passed but no validation needed.



Comments App

urls.py - Passed

```python
'''
URL configuration for the Comments application.

This module defines the URL patterns for the Comments app, mapping
views to specific endpoints for handling comment-related requests.
'''
from django.urls import path
from comments import views

urlpatterns = [
    path('comments/', views.CommentList.as_view()),
    path('comments/<int:pk>/', views.CommentDetail.as_view())
]

```

Settings:

Results:

All clear, no errors found

Comments App

views.py - Passed

```python
'''
Views for the Comments application.

This module defines API views for listing, creating, retrieving, updating,
and deleting comments using Django REST Framework.
'''
from rest_framework import generics, permissions
from django_filters.rest_framework import DjangoFilterBackend
from nomadnarrativesapi.permissions import IsOwnerOrReadOnly
from .models import Comment
from .serializers import CommentSerializer, CommentDetailSerializer


class CommentList(generics.ListCreateAPIView):
    '''
    List all comments
    Create a new comment if authenticated
    Associate the current logged in user with the comment
    '''

    permission_classes = [permissions.IsAuthenticatedOrReadOnly]
    serializer_class = CommentSerializer
    queryset = Comment.objects.all()

    filter_backends = [
        DjangoFilterBackend
        ]
    filterset_fields = [
        'post'
        ]
```

Settings:

Results:

All clear, no errors found

Contact App

admin.py - Passed

```python
'''
Admin configuration for the Contact model.
'''
from django.contrib import admin
from .models import Contact


@admin.register(Contact)
class ContactAdmin(admin.ModelAdmin):
    '''
    Admin interface for managing Contact model entries.

    - Displays sender name, email, and creation date in the admin list view.
    '''
    list_display = ('sender_name', 'email', 'created_at')
    search_fields = ('sender_name', 'last_name', 'email')

```

Settings:

Results:

All clear, no errors found

Contact App

apps.py - Passed

```python
'''
App configuration for the Contact application.
'''

from django.apps import AppConfig


class ContactConfig(AppConfig):
    '''
    Configuration class for the Contact app.
    Sets the default auto field type and specifies the app's name.
    '''
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'contact'
```

Settings:

Results:

All clear, no errors found

Contact App

models.py - Passed

```python
'''
Defines the Contact model for storing user-submitted contact form data.
'''
from django.db import models


class Contact(models.Model):
    '''
    Fields for form to contact admin.
    '''
    sender_name = models.CharField(max_length=100)
    email = models.EmailField()
    message = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return f"{self.sender_name}"

```

Settings:

Results:

All clear, no errors found

Contact App

serializers.py - Passed

```python
'''
Serializer for the Contact model.
'''

from rest_framework import serializers
from .models import Contact


class ContactSerializer(serializers.ModelSerializer):
    '''
    Serializer for contact model.
    '''
    class Meta:
        '''
        Returns all fields.
        '''
        model = Contact
        fields = '__all__'

```

Settings:

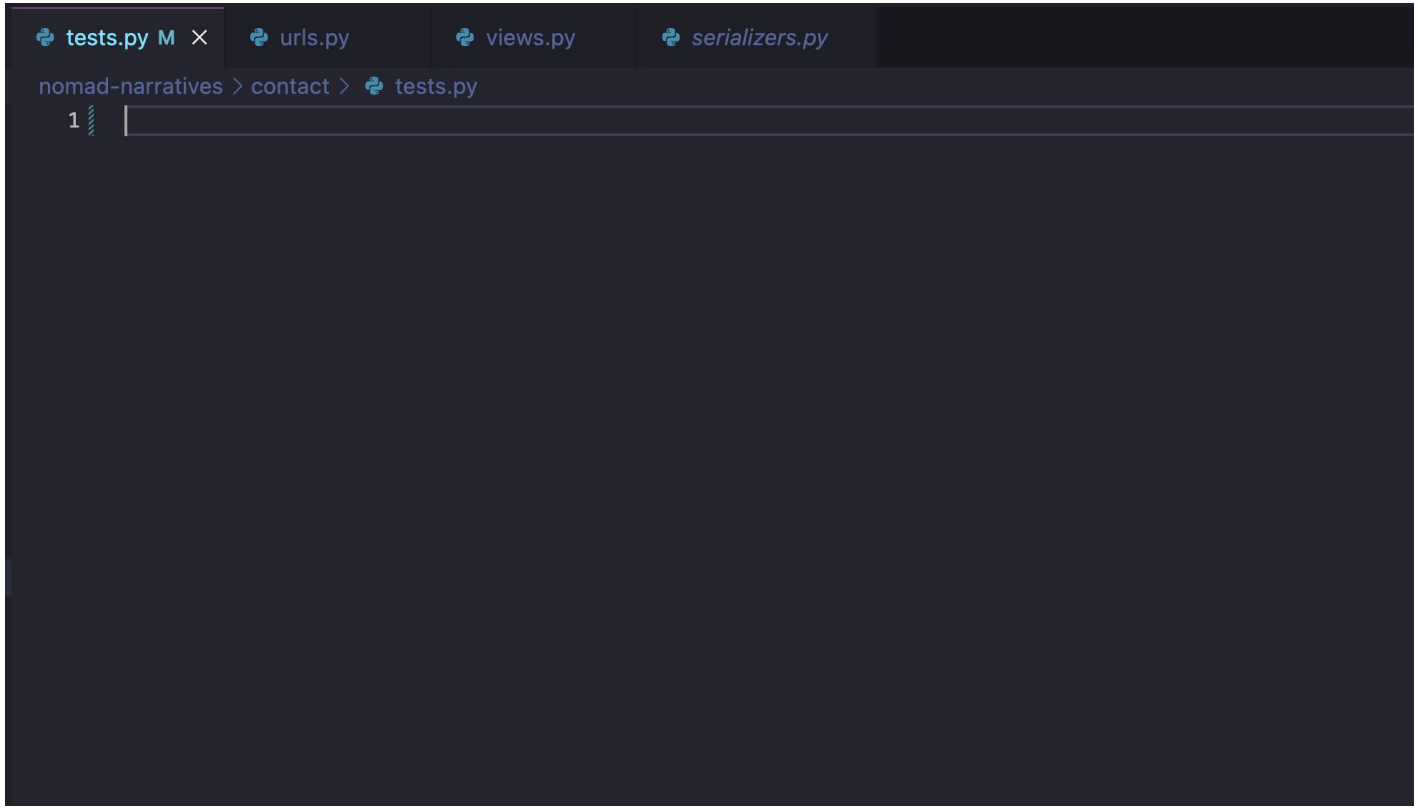
Results:

All clear, no errors found

Contact App

tests.py - Passed

Empty file



Contact App

urls.py - Passed

```python
'''
URL configuration for the Contact app.

Defines the URL pattern for the contact form API endpoint.
'''

from django.urls import path
from contact import views

urlpatterns = [
    path('contact/', views.ContactCreateView.as_view(), name='contact-create'),
]
```

Settings:

Results:

All clear, no errors found

Contact App

views.py - Passed

```python
'''
Defines the ContactCreateView, which provides an API endpoint
for creating Contact model instances.
'''
from rest_framework import generics
from .models import Contact
from .serializers import ContactSerializer


class ContactCreateView(generics.CreateAPIView):
    '''
    View for contact model.
    '''
    queryset = Contact.objects.all()  # pylint: disable=no-member
    serializer_class = ContactSerializer

```

Settings:

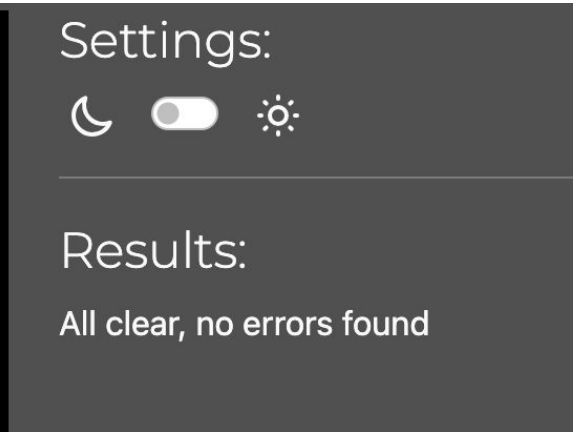
Results:

All clear, no errors found

Followers App

admin.py - Passed

```python
'''
Django admin configuration for the Follower model.
'''
from django.contrib import admin
from .models import Follower

admin.site.register(Follower)

```



Followers App

apps.py - Passed

```python
'''
App configuration for the Followers application.

This module defines the configuration settings for the Followers app
within the Django project.
'''
from django.apps import AppConfig


class FollowersConfig(AppConfig):
    '''
    Configuration class for the Followers app.

    This class sets the default primary key type and specifies the app name.
    '''
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'followers'
```

Settings:

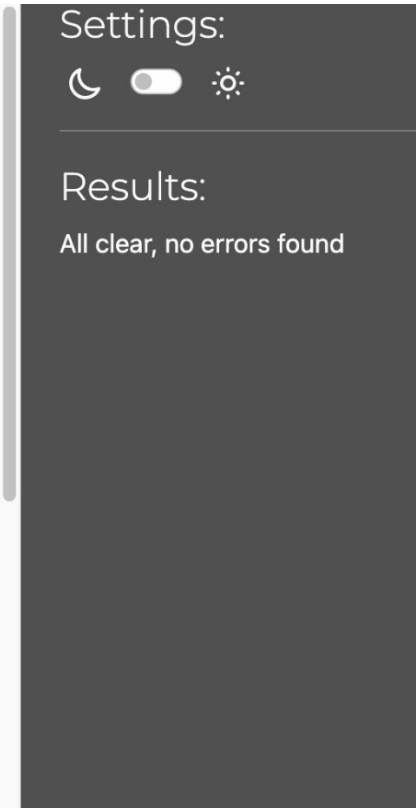
Results:

All clear, no errors found

Followers App

models.py - Passed

```python
'''
The `Follower` model tracks which users are following other users.
'''

from django.db import models
from django.contrib.auth.models import User


class Follower(models.Model):
    '''
    Represents a follower relationship between users.

    - 'owner' refers to the user who is following another user.
    - 'followed' refers to the user being followed.
    - The 'related_name' attribute distinguishes the two User instances.
    - The 'unique_together' constraint ensures a user cannot
    follow the same user more than once.
    '''
    owner = models.ForeignKey(
        User,
        on_delete=models.CASCADE,
        related_name='following'
        )
    followed = models.ForeignKey(
        User,
        on_delete=models.CASCADE,
        related_name='followed'
        )
    created_at = models.DateTimeField(
        auto_now_add=True
        )
```



Followers App

serializers.py - Passed

```python
'''
Serializer for the `Follower` model, handling the creation and serialization
of follower relationships between users.

Includes validation for duplicate follow attempts.
'''
from django.db import IntegrityError
from rest_framework import serializers
from .models import Follower


class FollowerSerializer(serializers.ModelSerializer):
    '''
    Serializer for Followers model.

    It handles serialization of the `owner` and `followed`
        user details, as well as managing errors related to duplicate
        follow requests.
    '''
    owner = serializers.ReadOnlyField(source='owner.username')
    followed_name = serializers.ReadOnlyField(source='followed.username')

    class Meta:
        '''
        Specifies what fields to be returned.
        '''
        model = Follower
        fields = ['id', 'owner', 'created_at', 'followed', 'followed_name']

    def create(self, validated_data):
        '''
```

Settings:

Results:

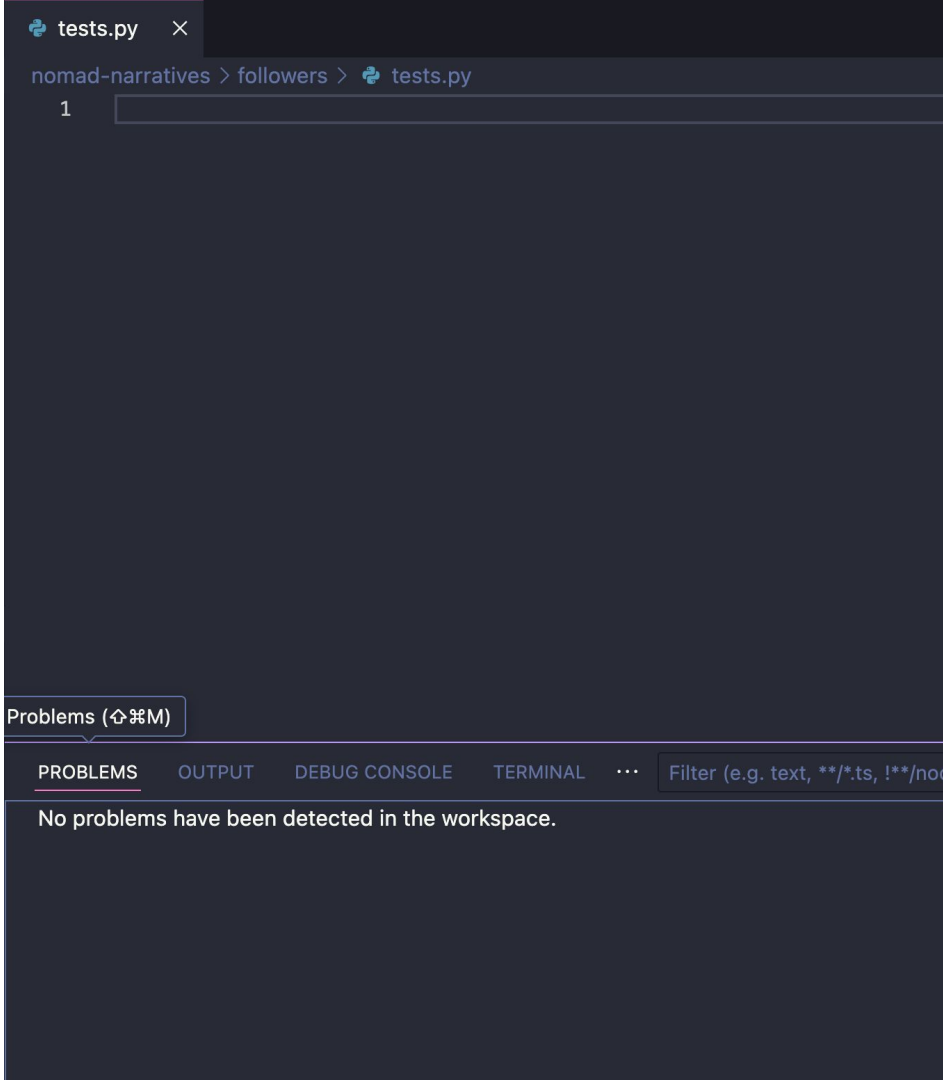
All clear, no errors found

Followers App

tests.py

Empty file -

Passed but no validation needed.



Followers App

urls.py - Passed

```python
'''
Defines URL patterns for managing follower relationships,
including listing followers and viewing details of a specific
follower relationship.
'''

from django.urls import path
from followers import views

urlpatterns = [
    path('followers/', views.FollowerList.as_view()),
    path('followers/<int:pk>/', views.FollowerDetail.as_view())
]

```

Settings:

Results:

All clear, no errors found

Followers App

views.py - Passed

```python
from rest_framework import generics, permissions
from nomadnarrativesapi.permissions import IsOwnerOrReadOnly
from .models import Follower
from .serializers import FollowerSerializer


class FollowerList(generics.ListCreateAPIView):
    '''
    API view to list all followers or create a new follower relationship.

    Allows authenticated users to create a new follow relationship,
    automatically setting the authenticated user as the owner of the follow.
    Read-only access is available to non-authenticated users.
    '''
    serializer_class = FollowerSerializer
    queryset = Follower.objects.all()
    permission_classes = [permissions.IsAuthenticatedOrReadOnly]

    def perform_create(self, serializer):
        serializer.save(owner=self.request.user)


class FollowerDetail(generics.RetrieveDestroyAPIView):
    '''
    Retrieves a follower.
    Allows users to follow or unfollow another user.
    '''
    serializer_class = FollowerSerializer
    queryset = Follower.objects.all()
    permission_classes = [IsOwnerOrReadOnly]

```

Settings:

Results:

All clear, no errors found

Likes App

admin.py - Passed

```
'''
Django admin configuration for the Like model.
'''
from django.contrib import admin
from .models import Like

admin.site.register(Like)

```

Settings:

Results:

All clear, no errors found

Likes App

apps.py - Passed

```python
'''
Configures the "likes" app, defining the default auto field type and app name
    .
'''
from django.apps import AppConfig


class LikesConfig(AppConfig):
    '''
    Configuration for the "likes" app, specifying the default auto field type
    and the app name.
    '''
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'likes'

```

Settings:

Results:

All clear, no errors found

Likes App

models.py - Passed

```python
'''
Contains the `Like` model, representing the relationship between users and
posts they have liked. Ensures that a user can like a post only once.
'''
from django.db import models
from django.contrib.auth.models import User
from posts.models import TripPost


class Like(models.Model):
    '''
    Like model, related to Owner and TripPost.
    'unique_together' ensures a user can't like one post twice.
    '''
    owner = models.ForeignKey(User, on_delete=models.CASCADE)
    post = models.ForeignKey(
        TripPost,
        on_delete=models.CASCADE,
        related_name='likes'
        )
    created_at = models.DateTimeField(auto_now_add=True)

    class Meta:
        '''
        Represents a like on a post by a user.

        Ensures that each user can like a post only once by enforcing a
        unique constraint between the user and post.
        '''
        ordering = ['-created_at']
        unique_together = ['owner', 'post']
```

Settings:

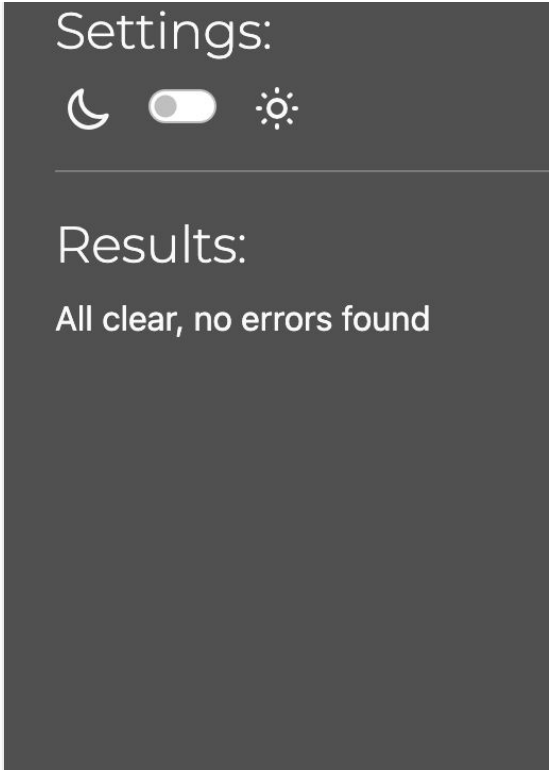
Results:

All clear, no errors found

Likes App

serializers.py - Passed

```python
'''
Contains the serializer for the `Like` model, which handles the creation
and serialization of "like" relationships between users and posts.
Includes error handling for duplicate likes.
'''
from django.db import IntegrityError
from rest_framework import serializers
from .models import Like


class LikeSerializer(serializers.ModelSerializer):
    '''
    Serializer for Like model.
    '''
    owner = serializers.ReadOnlyField(source='owner.username')

    class Meta:
        '''
        Specifies what fields to be returned.
        '''
        model = Like
        fields = ['id', 'created_at', 'owner', 'post']
```

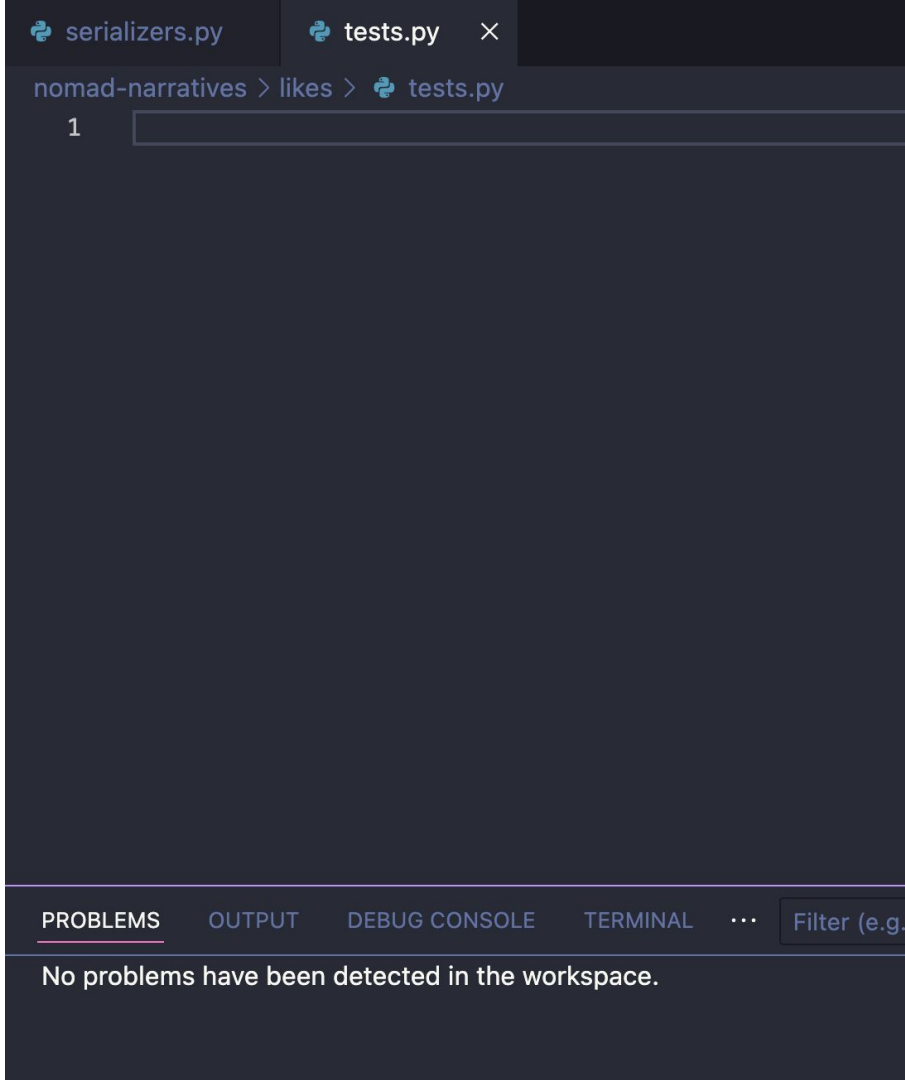


Likes App

tests.py

Empty file -

Passed but no validation needed.



Likes App

urls.py - Passed

```python
'''
Defines URL patterns for managing "like" relationships, including listing
    likes
and viewing details of a specific like.
'''
from django.urls import path
from likes import views

urlpatterns = [
    path('likes/', views.LikeList.as_view()),
    path('likes/<int:pk>/', views.LikeDetail.as_view()),
]

```

Settings:

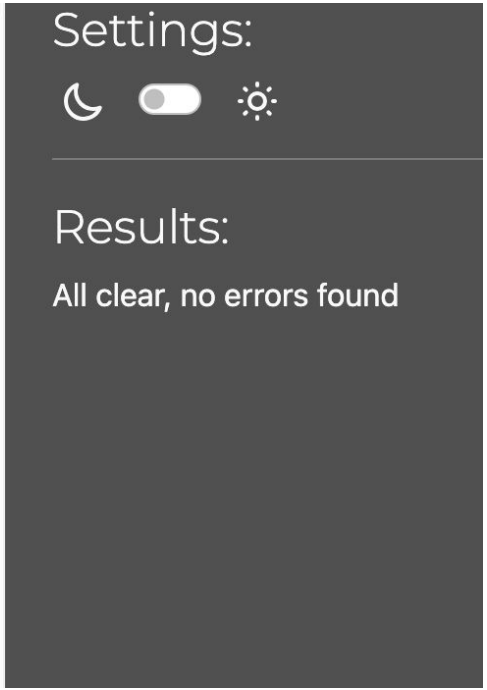
Results:

All clear, no errors found

Likes App

views.py - Passed

```python
'''
Contains API views for managing "like" relationships between users and
    posts.

- `LikeList`: Lists all likes and allows authenticated users to create
  a new like for a post. Non-authenticated users have read-only access.

- `LikeDetail`: Retrieves or deletes a specific like. Only the owner of the
  like can modify or remove it.
'''
from rest_framework import generics, permissions
from nomadnarrativesapi.permissions import IsOwnerOrReadOnly
from .models import Like
from .serializers import LikeSerializer


class LikeList(generics.ListCreateAPIView):
    '''
    Lists all likes.
    Creates a like if user is authenticated.
    The perform_create method links the like with the logged in user.
    '''
```



Posts App

admin.py - Passed

```python
'''
Admin configuration for the TripPost and TripDetails models.
- Manages trip-related data, including inline editing of TripDetails
within TripPost.
'''
from django.contrib import admin
from .models import TripPost, TripDetails


class TripDetailsInline(admin.StackedInline):
    '''
    TripDetails Inline Admin: Allow editing TripDetails
    directly within TripPost.
    '''
    model = TripDetails
    extra = 1  # Number of empty forms to show by default
    fields = (
        'country', 'city', 'traveller_number', 'relevant_for',
        'duration_value', 'duration_unit'
    )  # Fields to display in the inline form

    def get_cities(self, obj):
        """
        Custom method to display a list of cities as a
        comma-separated string.
        """
```

Settings:

Results:

All clear, no errors found

Posts App

apps.py - Passed

```python
'''
App configuration for the Posts application.

This module defines the configuration settings for the Posts app
within the Django project.
'''
from django.apps import AppConfig


class PostsConfig(AppConfig):
    '''
    This class sets the default primary key type and specifies the app name.
    '''
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'posts'
```

Settings:

Results:

All clear, no errors found

Posts App

models.py - Passed

```python

from django.db import models
from django.core.exceptions import ValidationError
from django.contrib.auth.models import User
from cities_light.models import City, Country
from utils.continents import get_continent_by_country


class TripPost(models.Model):
    '''
    Trip Post model, related to 'owner', i.e. a User instance.
    Default image set so that we can always reference image.url.

    This model handles the content of the post (image, article content, title).
    This model does not handle the details of the trip (see TripDetails model).
    '''

    owner = models.ForeignKey(
        User,
        on_delete=models.CASCADE
        )
    created_at = models.DateTimeField(
        auto_now_add=True
```

Settings:

Results:

All clear, no errors found

Posts App

serializers.py - Passed

```python
'''
Posts Serializer
'''
from rest_framework import serializers
from cities_light.models import City, Country
from likes.models import Like
from .models import TripPost, TripDetails


class TripDetailsSerializer(serializers.ModelSerializer):
    '''
    Serializer for TripDetails model.
    '''
    continent = serializers.ReadOnlyField()
    country = serializers.PrimaryKeyRelatedField(
        queryset=Country.objects.all())
    city = serializers.PrimaryKeyRelatedField(
        queryset=City.objects.all()
        )
```

Settings:

Results:

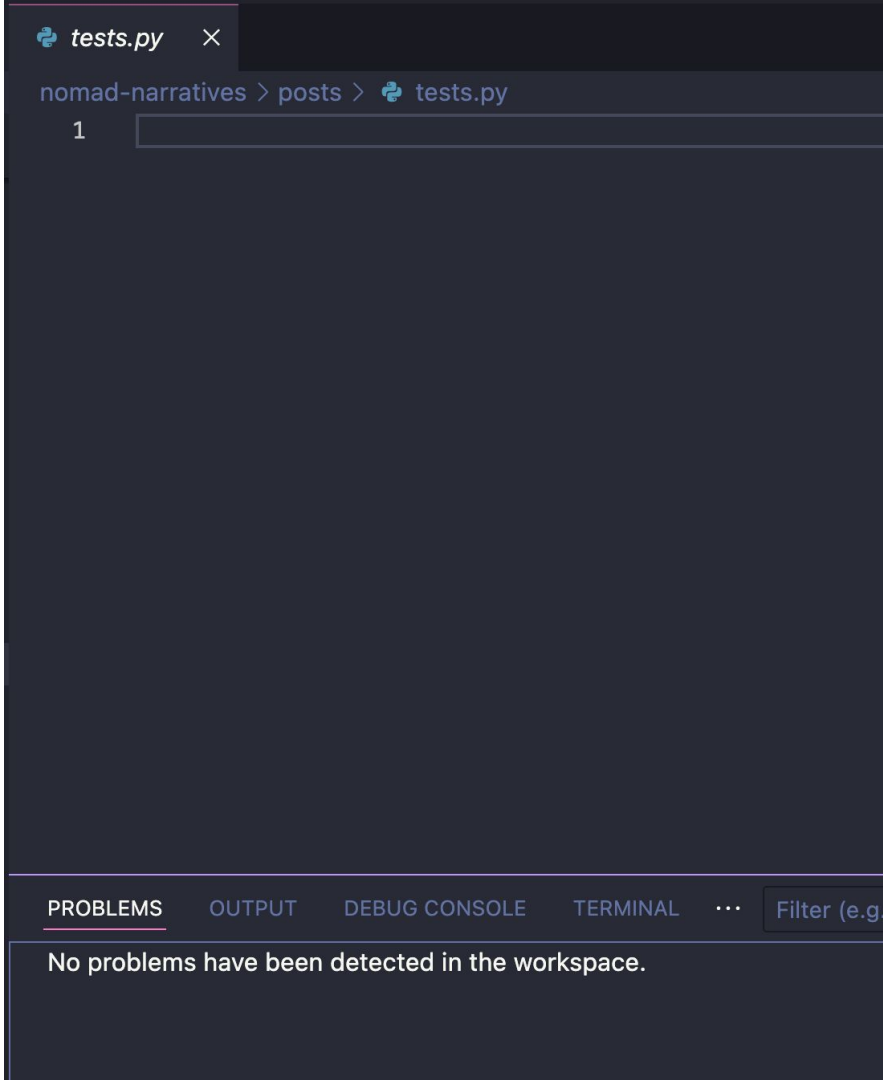
All clear, no errors found

Posts App

tests.py

Empty file -

Passed but no validation needed.



Posts App

urls.py - Passed

```python
'''
URL configuration for the posts app.

This module defines the URL patterns for the post-related views, including:

- A list view for all posts (`PostList`).
- A detail view for individual posts (`PostDetail`).
'''
from django.urls import path
from posts import views

urlpatterns = [
    path('posts/', views.PostList.as_view()),
    path('posts/<int:pk>/', views.PostDetail.as_view()),
    ]

```

Settings:

Results:

All clear, no errors found

Posts App

views.py - Passed

```python
'''
Imports for views related to TripPost.

This module imports various libraries and components used for handling
TripPost views, including Django, REST framework components, and custom
permissions and serializers.
'''
from django.db.models import Count
from django.shortcuts import get_object_or_404
from rest_framework import generics, permissions, filters
from django_filters.rest_framework import DjangoFilterBackend
from nomadnarrativesapi.permissions import IsOwnerOrReadOnly
from utils.continents import get_continent_by_country
from cities_light.models import Country, City
from .serializers import TripPostSerializer
from .models import TripPost


class PostList(generics.ListCreateAPIView):
    '''
    Handles Post logic.
    '''
```

Settings:

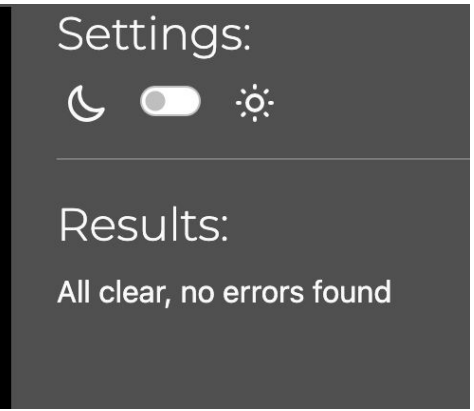
Results:

All clear, no errors found

Profiles App

admin.py - Passed

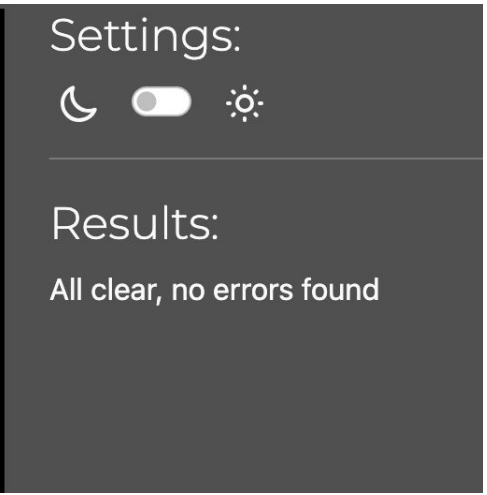
```python
'''
Django admin configuration for the Profile model.
'''
from django.contrib import admin
from .models import Profile

admin.site.register(Profile)

```



Profiles App

apps.py - Passed

```python
'''
Configures the "profiles" app, specifying the default auto field type
and the app name.
'''
from django.apps import AppConfig


class ProfilesConfig(AppConfig):
    '''
    This class specifies the default auto field type for the app
    (BigAutoField) and sets the name of the app to "Profiles".
    '''
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'profiles'

```



Profiles App

models.py - Passed

```python
'''
Defines the Profile model and its related logic for user profiles.

- `Profile`: Stores user profile information such as name, content (bio),
    and image.
- A new profile is automatically created when a new User instance is created.
'''

from django.db import models
from django.db.models.signals import post_save
from django.dispatch import receiver
from django.contrib.auth.models import User


class Profile(models.Model):
    '''
    Profile model.
    '''
    owner = models.OneToOneField(User, on_delete=models.CASCADE)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
    name = models.CharField(max_length=255, blank=True)
    content = models.TextField(blank=True)
    image = models.ImageField(
        upload_to='images/', default='../default_profile_cf7kd5'
    )
```

Settings:

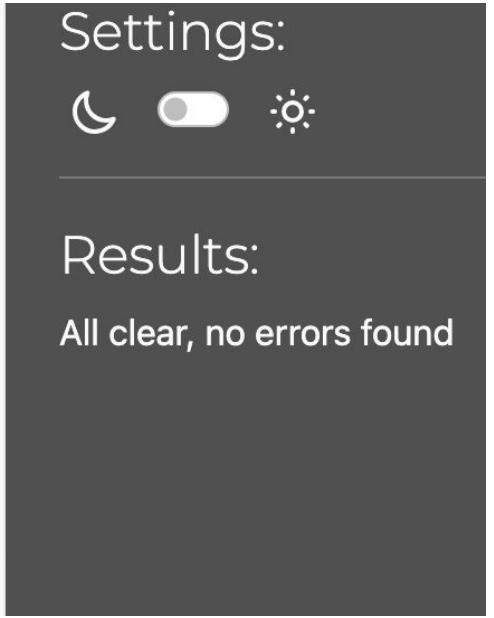
Results:

All clear, no errors found

Profiles App

serializers.py - Passed

```python
'''
Serializer for the Profile model and related fields.

- `ProfileSerializer`: Serializes Profile model data, including user
    profile
    details and dynamic fields like follow status, post counts, and
    follower counts.
'''
from rest_framework import serializers
from followers.models import Follower
from .models import Profile


class ProfileSerializer(serializers.ModelSerializer):
    '''
    Serializer for Profile model.
    '''
```

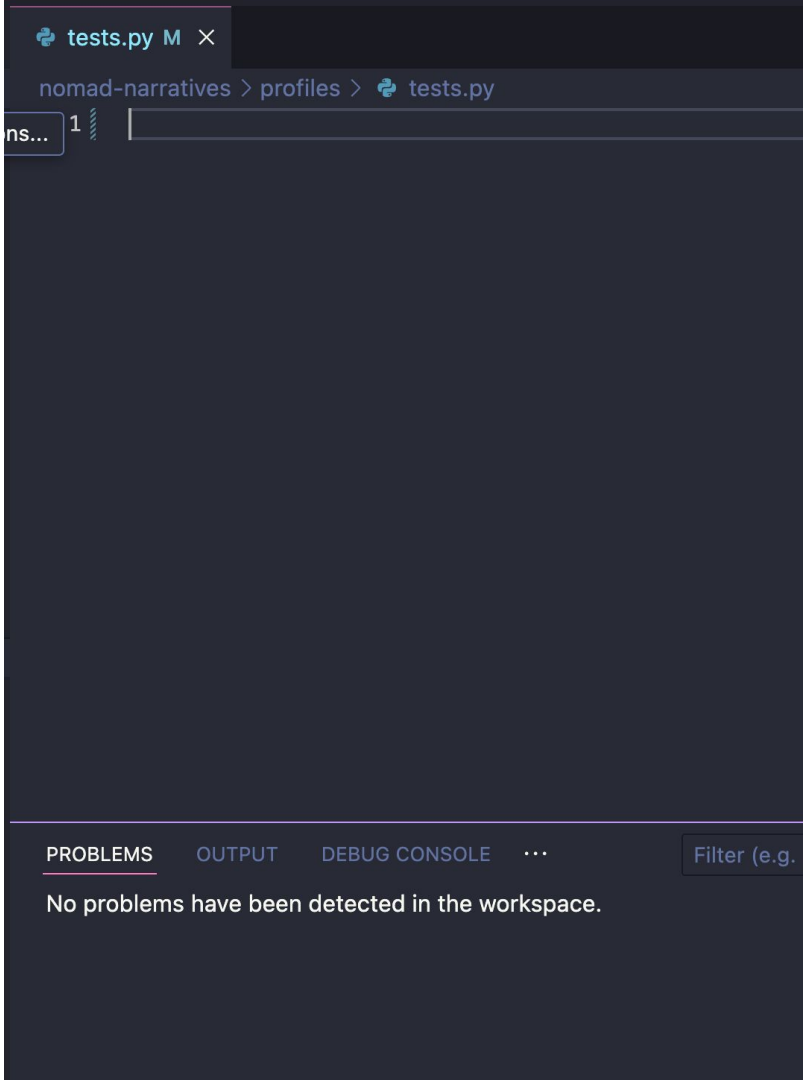


Profiles App

tests.py

Empty file -

Passed but no validation needed.



Profiles App

urls.py - Passed

```python
'''
Defines URL routes for accessing and interacting with user profiles.

- `ProfileList`: View for listing all profiles and creating new profiles.
- `ProfileDetail`: View for retrieving, updating, and
   deleting a specific profile.

URLs:
- `/profiles/`: List all profiles or create a new one.
- `/profiles/<int:pk>/`: Retrieve, update, or delete a profile by
   its primary key.
'''

from django.urls import path
from profiles import views

urlpatterns = [
    path('profiles/', views.ProfileList.as_view()),
    path('profiles/<int:pk>/', views.ProfileDetail.as_view()),
]
```

Settings:

Results:

All clear, no errors found

Profiles App

views.py - Passed

```
'''
Profile Views for handling profile-related API endpoints.

Includes views for retrieving, listing, and updating user profiles. It also
handles custom filtering and ordering options for profiles based on
various metrics.
'''

from django.db.models import Count
from django_filters.rest_framework import DjangoFilterBackend
from rest_framework import generics, filters
from nomadnarrativesapi.permissions import IsOwnerOrReadOnly
from .models import Profile
from .serializers import ProfileSerializer


class ProfileList(generics.ListAPIView):
    '''
    Returns a list of all profiles.
    No Create view (profile creation handled by Django signals).
    '''
    queryset = Profile.objects.annotate(  # pylint: disable=no-member
```

Settings:

Results:

All clear, no errors found

Utils

continents.py - Passed

```
'''
This module provides a mapping of countries to their respective continents.
It includes a function to retrieve the continent of a given country.

The `CONTINENT_MAPPING` dictionary contains the continent as the key and
a list of countries as the value. The function `get_continent_by_country`
returns the continent name for a given country.
'''

CONTINENT_MAPPING = {
    "Africa": [
        "Algeria", "Angola", "Benin", "Botswana", "Burkina Faso",
        "Burundi", "Cabo Verde", "Cameroon", "Central African Republic",
        "Chad", "Comoros", "Democratic Republic of the Congo", "Djibouti",
        "Egypt", "Equatorial Guinea", "Eritrea", "Eswatini", "Ethiopia",
        "Gabon", "Gambia", "Ghana", "Guinea", "Guinea-Bissau",
        "Ivory Coast", "Kenya", "Lesotho", "Liberia", "Libya",
        "Madagascar", "Malawi", "Mali", "Mauritania", "Mauritius",
```

Settings:

Results:

All clear, no errors found