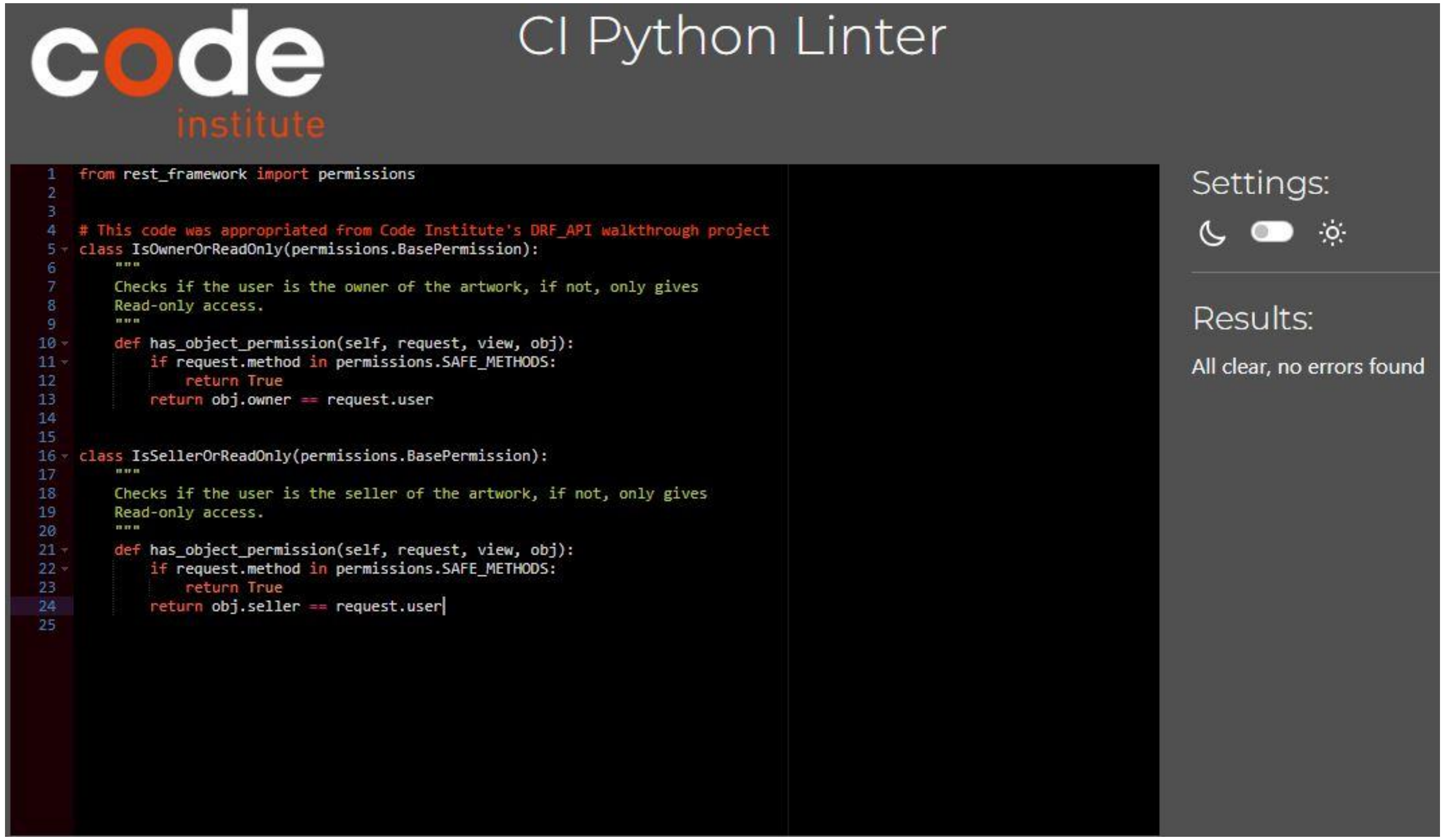
## Artly_api – main project app

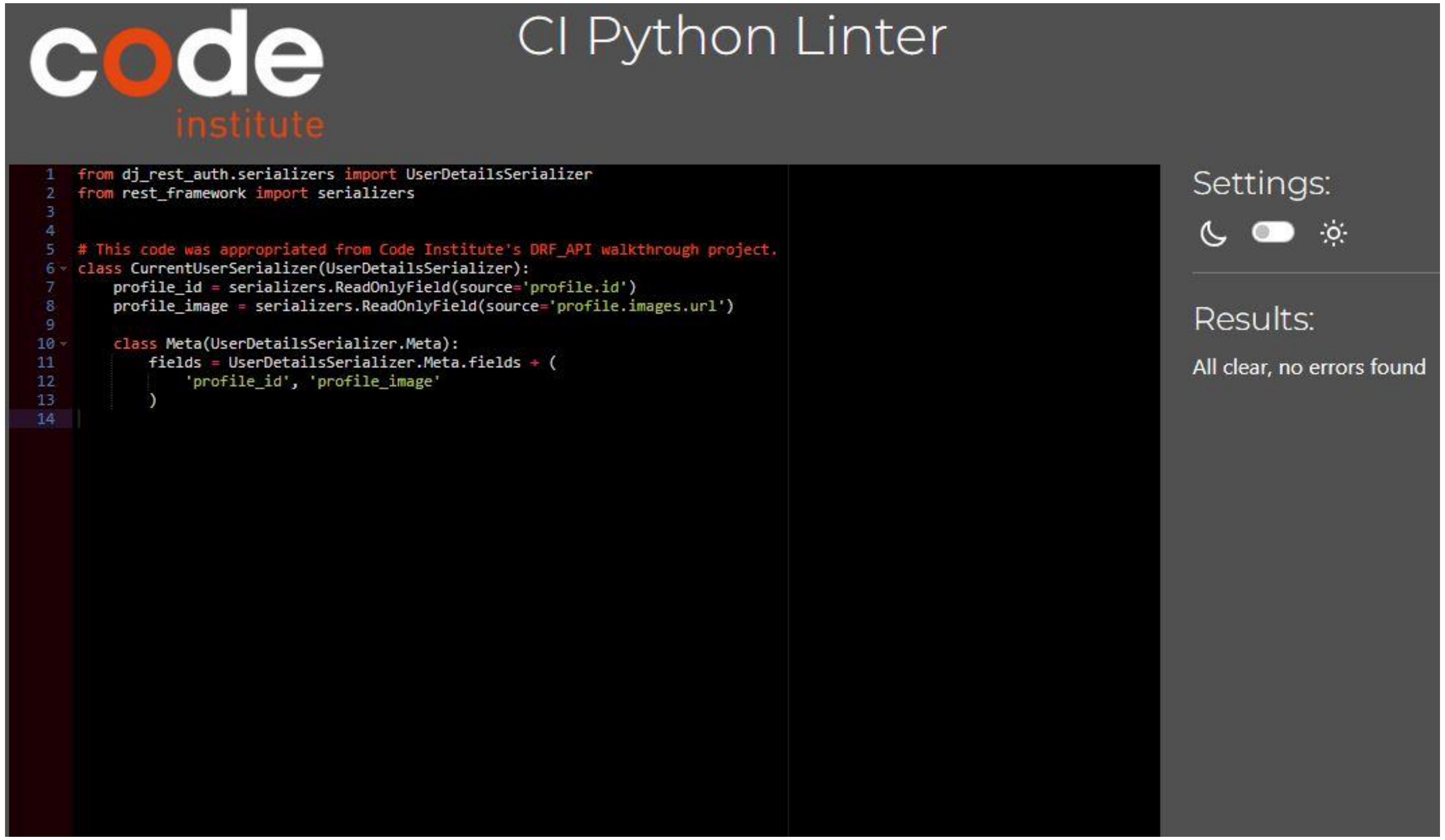

CI Python Linter

```
from rest_framework import permissions


# This code was appropriated from Code Institute's DRF_API walkthrough project
class IsOwnerOrReadOnly(permissions.BasePermission):
    """
    Checks if the user is the owner of the artwork, if not, only gives
    Read-only access.
    """
    def has_object_permission(self, request, view, obj):
        if request.method in permissions.SAFE_METHODS:
            return True
        return obj.owner == request.user


class IsSellerOrReadOnly(permissions.BasePermission):
    """
    Checks if the user is the seller of the artwork, if not, only gives
    Read-only access.
    """
    def has_object_permission(self, request, view, obj):
        if request.method in permissions.SAFE_METHODS:
            return True
        return obj.seller == request.user
```

Settings:

Results:

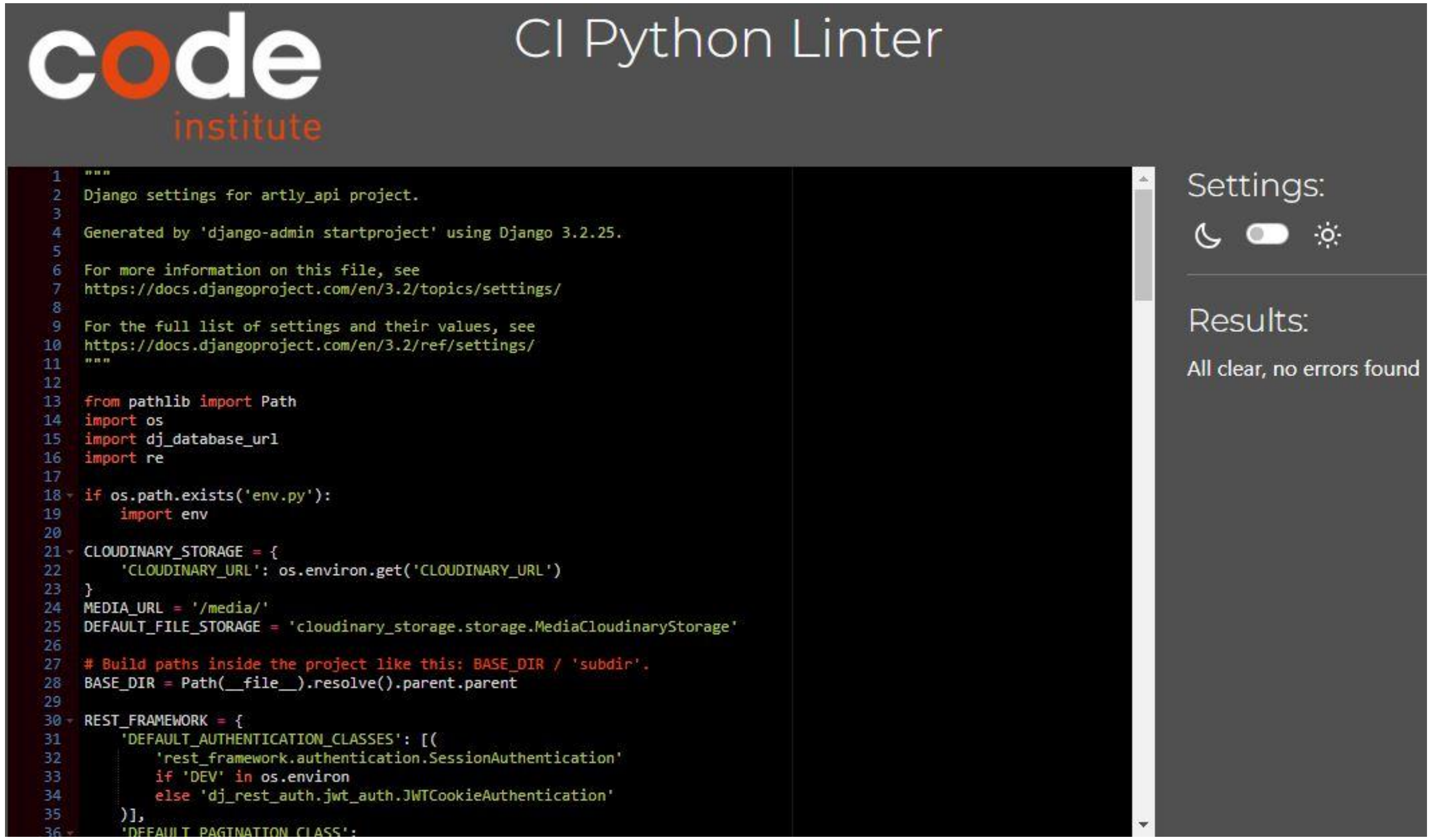
All clear, no errors found


CI Python Linter

```
from dj_rest_auth.serializers import UserDetailsSerializer
from rest_framework import serializers


# This code was appropriated from Code Institute's DRF_API walkthrough project.
class CurrentUserSerializer(UserDetailsSerializer):
    profile_id = serializers.ReadOnlyField(source='profile.id')
    profile_image = serializers.ReadOnlyField(source='profile.images.url')

    class Meta(UserDetailsSerializer.Meta):
        fields = UserDetailsSerializer.Meta.fields + (
            'profile_id', 'profile_image'
        )
```

Results:

All clear, no errors found



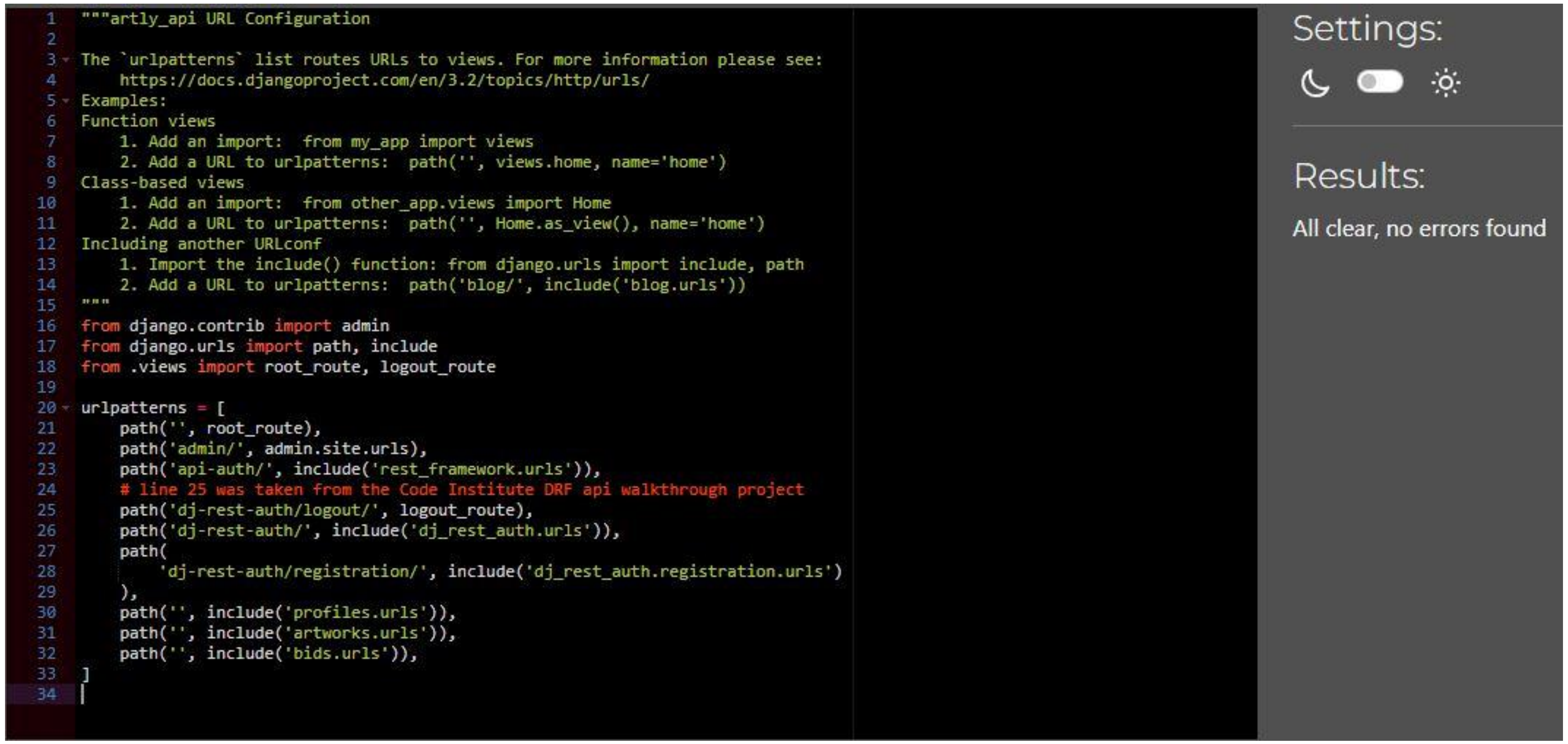
CI Python Linter

```
"""
Django settings for artly_api project.

Generated by 'django-admin startproject' using Django 3.2.25.

For more information on this file, see
https://docs.djangoproject.com/en/3.2/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/3.2/ref/settings/
"""

from pathlib import Path
import os
import dj_database_url
import re

if os.path.exists('env.py'):
    import env

CLOUDINARY_STORAGE = {
    'CLOUDINARY_URL': os.environ.get('CLOUDINARY_URL')
}
MEDIA_URL = '/media/'
DEFAULT_FILE_STORAGE = 'cloudinary_storage.storage.MediaCloudinaryStorage'

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent

REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': [(
        'rest_framework.authentication.SessionAuthentication'
        if 'DEV' in os.environ
        else 'dj_rest_auth.jwt_auth.JWTCookieAuthentication'
    )],
    'DEFAULT_PAGINATION_CLASS':
```

Settings:

Results:

All clear, no errors found

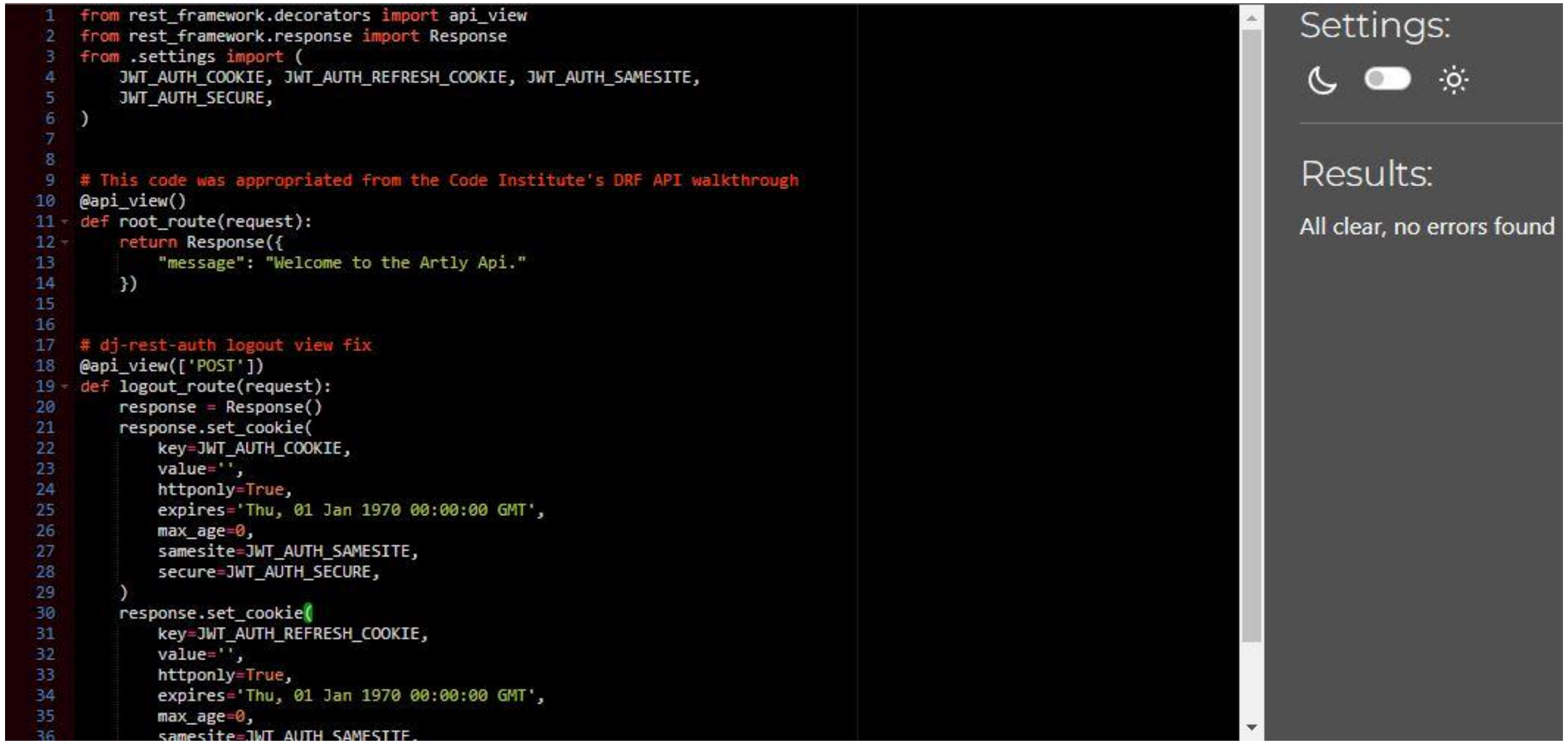


```
1  """artly_api URL Configuration
2  
3  The `urlpatterns` list routes URLs to views. For more information please see:
4      https://docs.djangoproject.com/en/3.2/topics/http/urls/
5  Examples:
6  Function views
7      1. Add an import:  from my_app import views
8      2. Add a URL to urlpatterns:  path('', views.home, name='home')
9  Class-based views
10     1. Add an import:  from other_app.views import Home
11     2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
12 Including another URLconf
13     1. Import the include() function: from django.urls import include, path
14     2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
15 """
16 from django.contrib import admin
17 from django.urls import path, include
18 from .views import root_route, logout_route
19 
20 urlpatterns = [
21     path('', root_route),
22     path('admin/', admin.site.urls),
23     path('api-auth/', include('rest_framework.urls')),
24     # line 25 was taken from the Code Institute DRF api walkthrough project
25     path('dj-rest-auth/logout/', logout_route),
26     path('dj-rest-auth/', include('dj_rest_auth.urls')),
27     path(
28         'dj-rest-auth/registration/', include('dj_rest_auth.registration.urls')
29     ),
30     path('', include('profiles.urls')),
31     path('', include('artworks.urls')),
32     path('', include('bids.urls')),
33 ]
34 
```

Settings:

Results:

All clear, no errors found



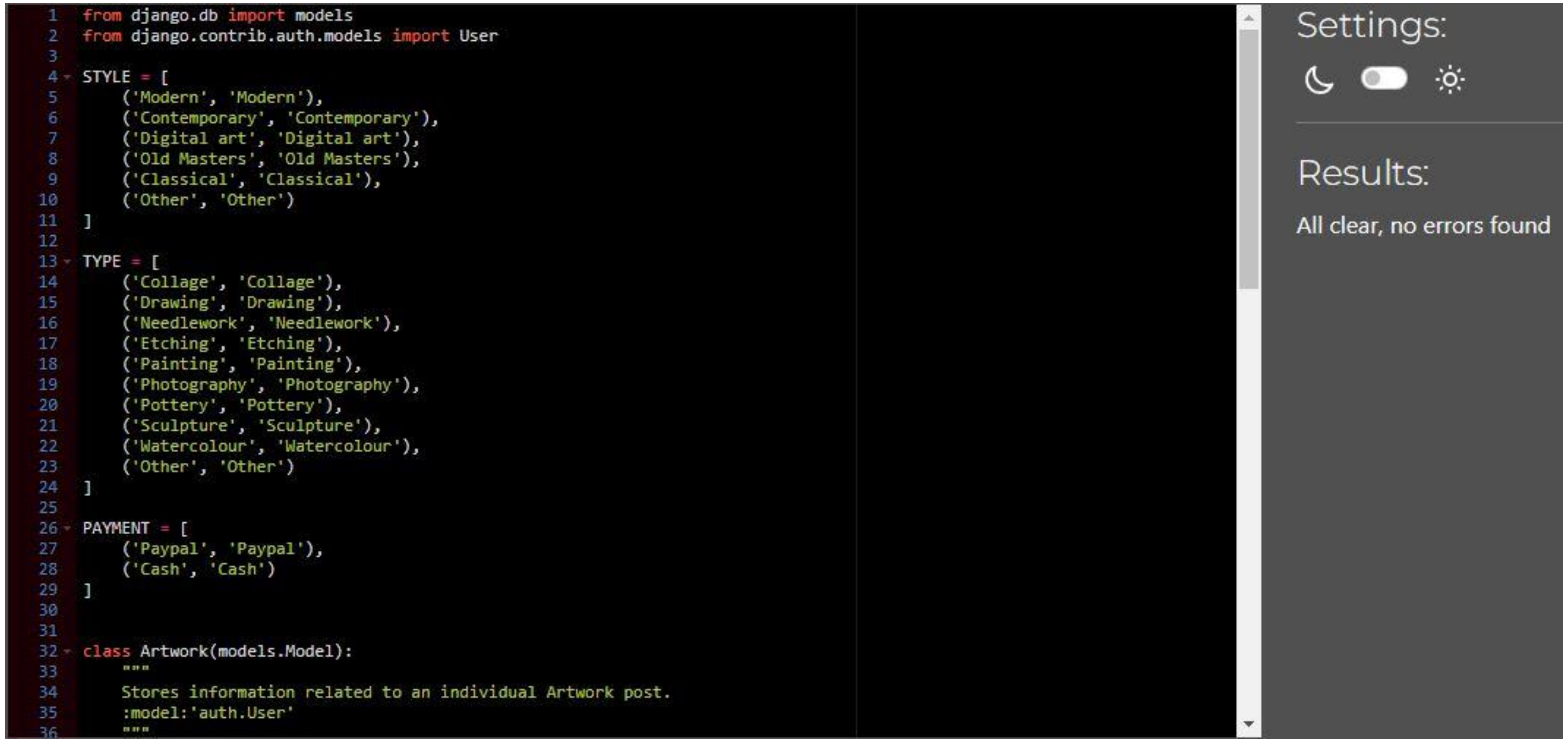
```python
from rest_framework.decorators import api_view
from rest_framework.response import Response
from .settings import (
    JWT_AUTH_COOKIE, JWT_AUTH_REFRESH_COOKIE, JWT_AUTH_SAMESITE,
    JWT_AUTH_SECURE,
)


# This code was appropriated from the Code Institute's DRF API walkthrough
@api_view()
def root_route(request):
    return Response({
        "message": "Welcome to the Artly Api."
    })


# dj-rest-auth logout view fix
@api_view(['POST'])
def logout_route(request):
    response = Response()
    response.set_cookie(
        key=JWT_AUTH_COOKIE,
        value='',
        httponly=True,
        expires='Thu, 01 Jan 1970 00:00:00 GMT',
        max_age=0,
        samesite=JWT_AUTH_SAMESITE,
        secure=JWT_AUTH_SECURE,
    )
    response.set_cookie(
        key=JWT_AUTH_REFRESH_COOKIE,
        value='',
        httponly=True,
        expires='Thu, 01 Jan 1970 00:00:00 GMT',
        max_age=0,
        samesite=JWT_AUTH_SAMESITE,
```

Settings:

Results:

All clear, no errors found

## Artwork app

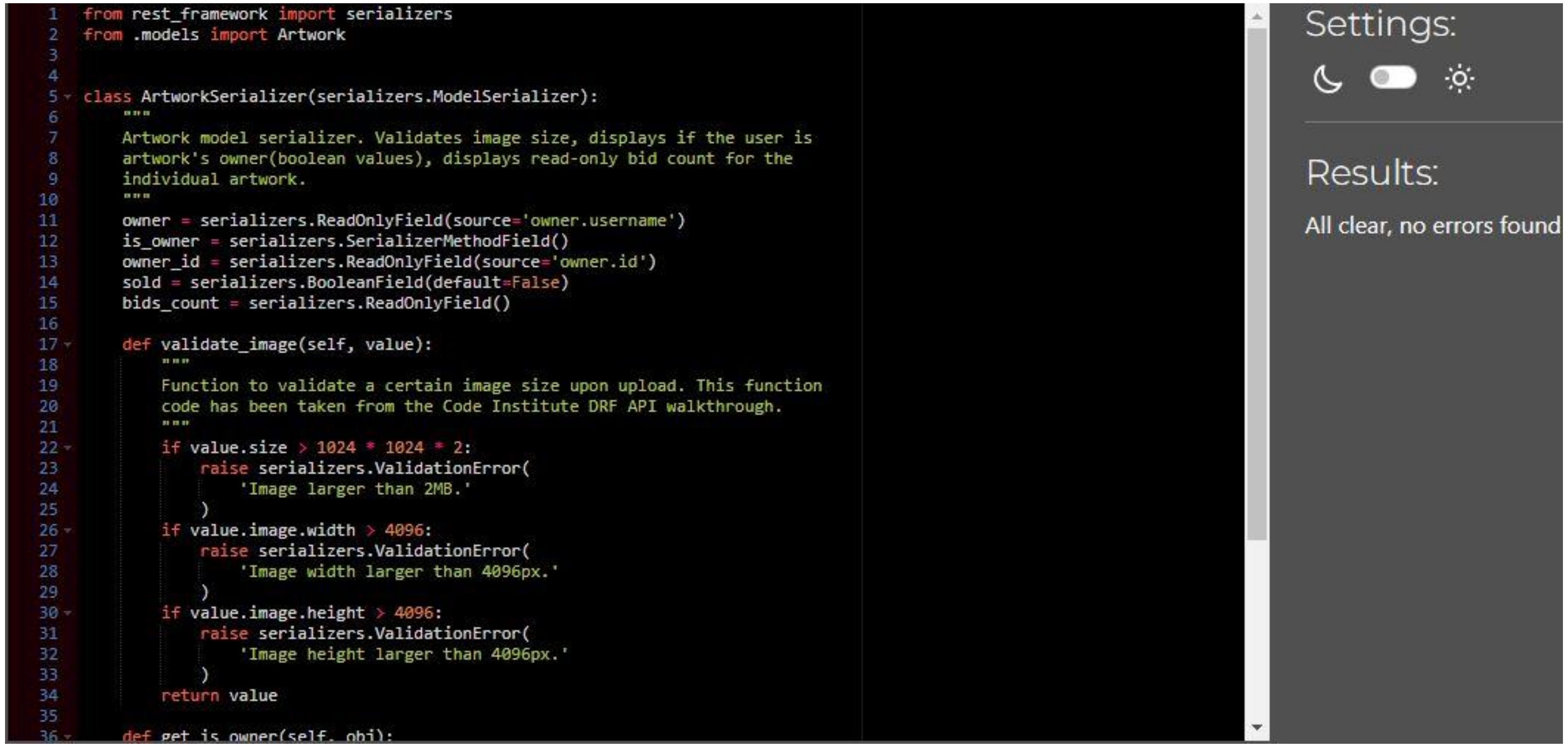


```
from django.db import models
from django.contrib.auth.models import User

STYLE = [
    ('Modern', 'Modern'),
    ('Contemporary', 'Contemporary'),
    ('Digital art', 'Digital art'),
    ('Old Masters', 'Old Masters'),
    ('Classical', 'Classical'),
    ('Other', 'Other')
]

TYPE = [
    ('Collage', 'Collage'),
    ('Drawing', 'Drawing'),
    ('Needlework', 'Needlework'),
    ('Etching', 'Etching'),
    ('Painting', 'Painting'),
    ('Photography', 'Photography'),
    ('Pottery', 'Pottery'),
    ('Sculpture', 'Sculpture'),
    ('Watercolour', 'Watercolour'),
    ('Other', 'Other')
]

PAYMENT = [
    ('Paypal', 'Paypal'),
    ('Cash', 'Cash')
]


class Artwork(models.Model):
    """
    Stores information related to an individual Artwork post.
    :model:'auth.User'
    """
```

Settings:

Results:

All clear, no errors found

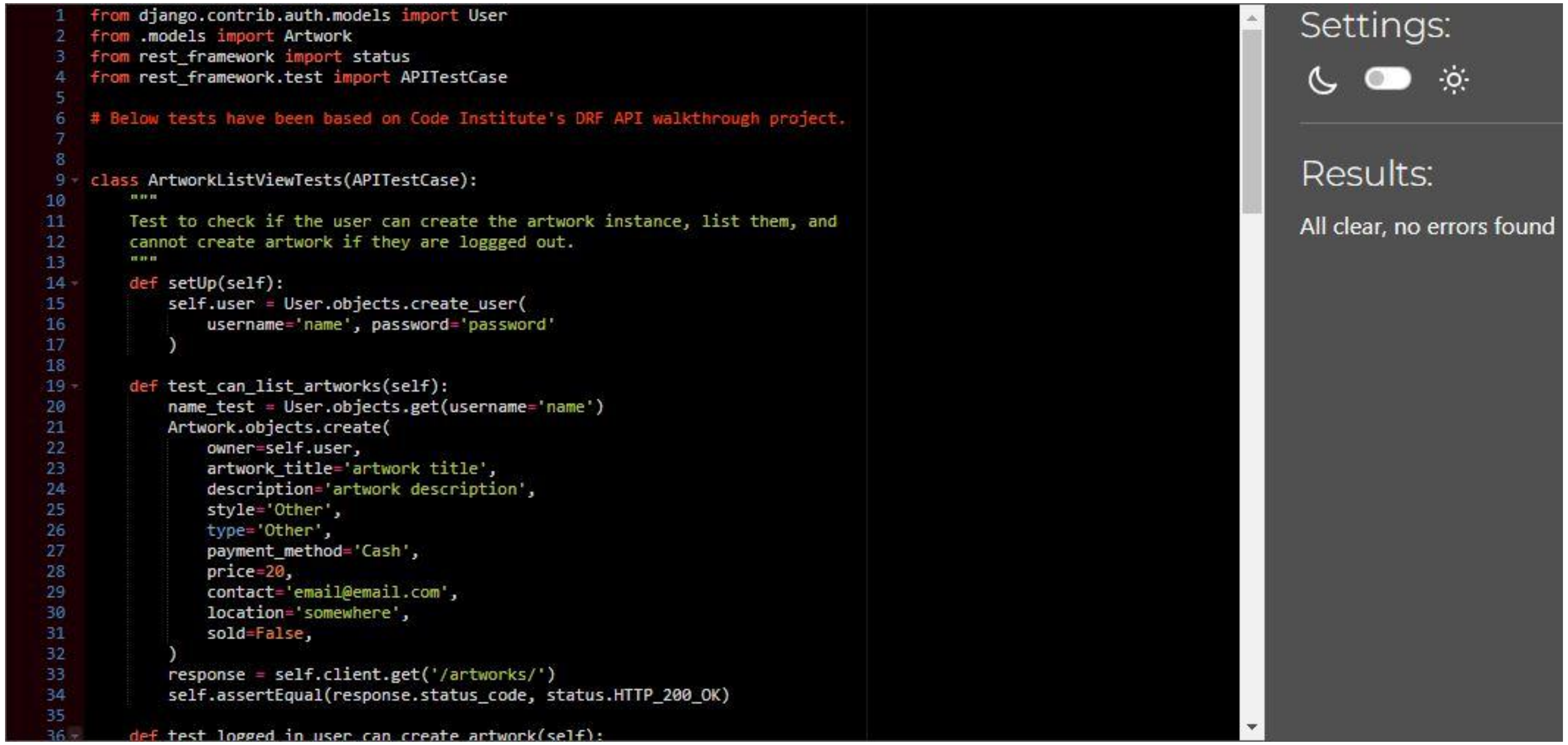


```
from rest_framework import serializers
from .models import Artwork


class ArtworkSerializer(serializers.ModelSerializer):
    """
    Artwork model serializer. Validates image size, displays if the user is
    artwork's owner(boolean values), displays read-only bid count for the
    individual artwork.
    """
    owner = serializers.ReadOnlyField(source='owner.username')
    is_owner = serializers.SerializerMethodField()
    owner_id = serializers.ReadOnlyField(source='owner.id')
    sold = serializers.BooleanField(default=False)
    bids_count = serializers.ReadOnlyField()

    def validate_image(self, value):
        """
        Function to validate a certain image size upon upload. This function
        code has been taken from the Code Institute DRF API walkthrough.
        """
        if value.size > 1024 * 1024 * 2:
            raise serializers.ValidationError(
                'Image larger than 2MB.'
            )
        if value.image.width > 4096:
            raise serializers.ValidationError(
                'Image width larger than 4096px.'
            )
        if value.image.height > 4096:
            raise serializers.ValidationError(
                'Image height larger than 4096px.'
            )
        return value

    def get_is_owner(self, obj):
```

Settings:

Results:

All clear, no errors found

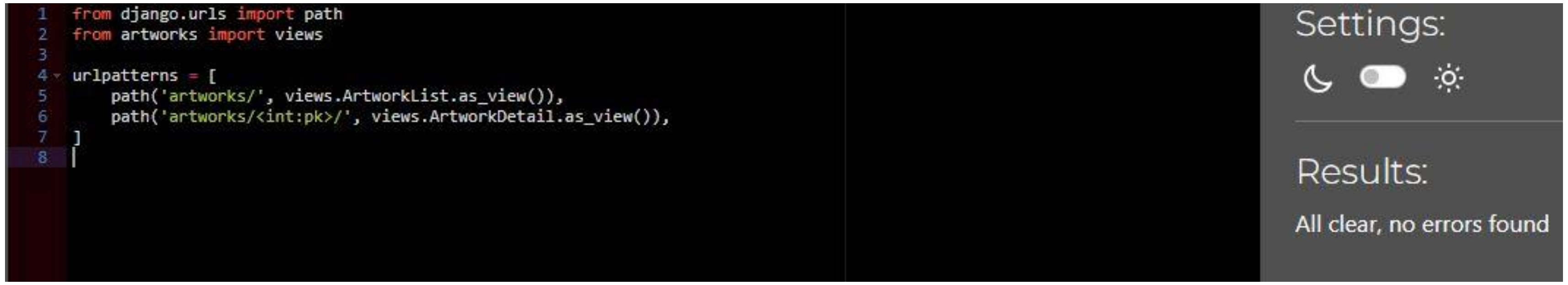


```
from django.contrib.auth.models import User
from .models import Artwork
from rest_framework import status
from rest_framework.test import APITestCase

# Below tests have been based on Code Institute's DRF API walkthrough project.


class ArtworkListViewTests(APITestCase):
    """
    Test to check if the user can create the artwork instance, list them, and
    cannot create artwork if they are loggged out.
    """
    def setUp(self):
        self.user = User.objects.create_user(
            username='name', password='password'
        )

    def test_can_list_artworks(self):
        name_test = User.objects.get(username='name')
        Artwork.objects.create(
            owner=self.user,
            artwork_title='artwork title',
            description='artwork description',
            style='Other',
            type='Other',
            payment_method='Cash',
            price=20,
            contact='email@email.com',
            location='somewhere',
            sold=False,
        )
        response = self.client.get('/artworks/')
        self.assertEqual(response.status_code, status.HTTP_200_OK)

    def test_logged_in_user_can_create_artwork(self):
```

Settings:

Results:

All clear, no errors found

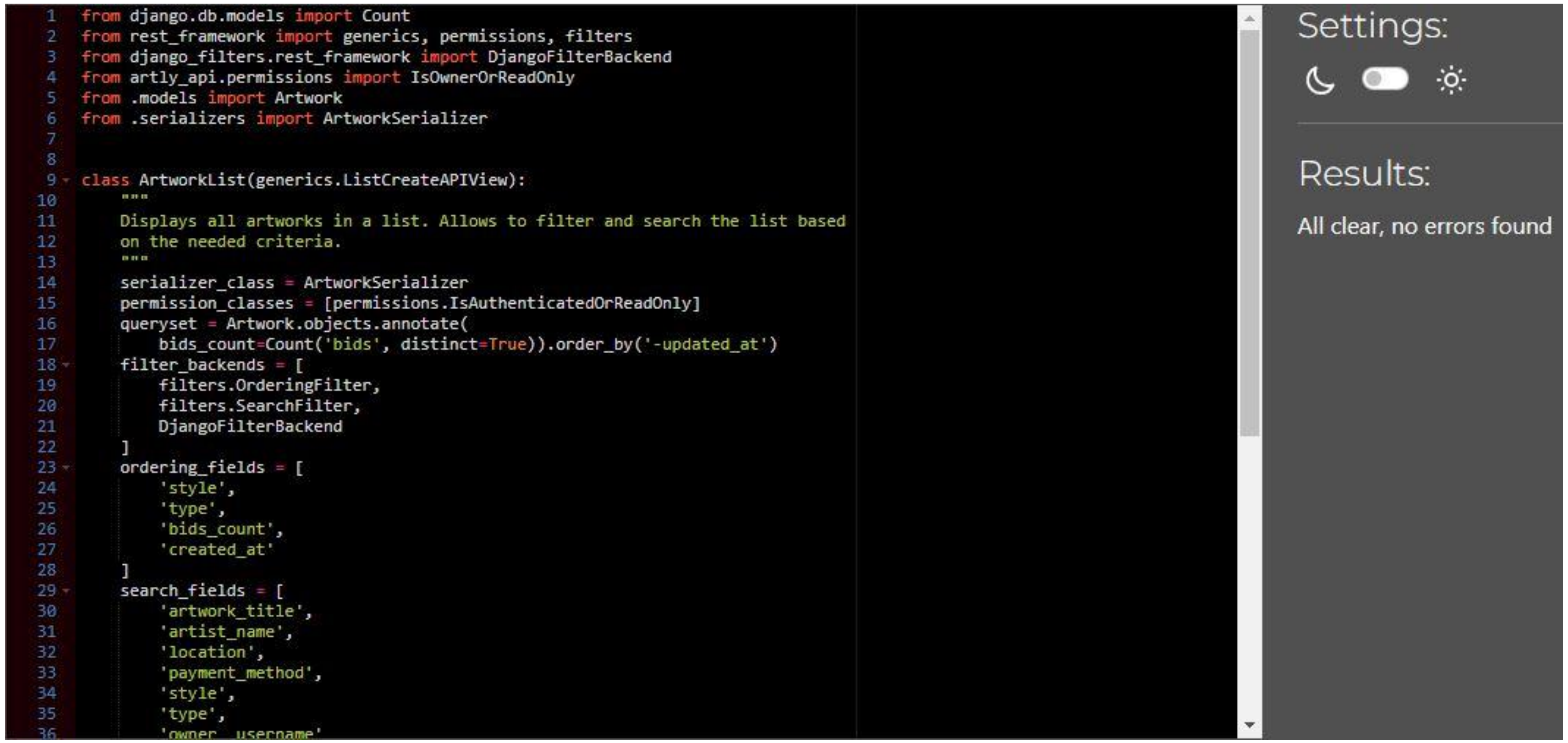


```python
from django.urls import path
from artworks import views

urlpatterns = [
    path('artworks/', views.ArtworkList.as_view()),
    path('artworks/<int:pk>/', views.ArtworkDetail.as_view()),
]
```

Settings:

Results:

All clear, no errors found

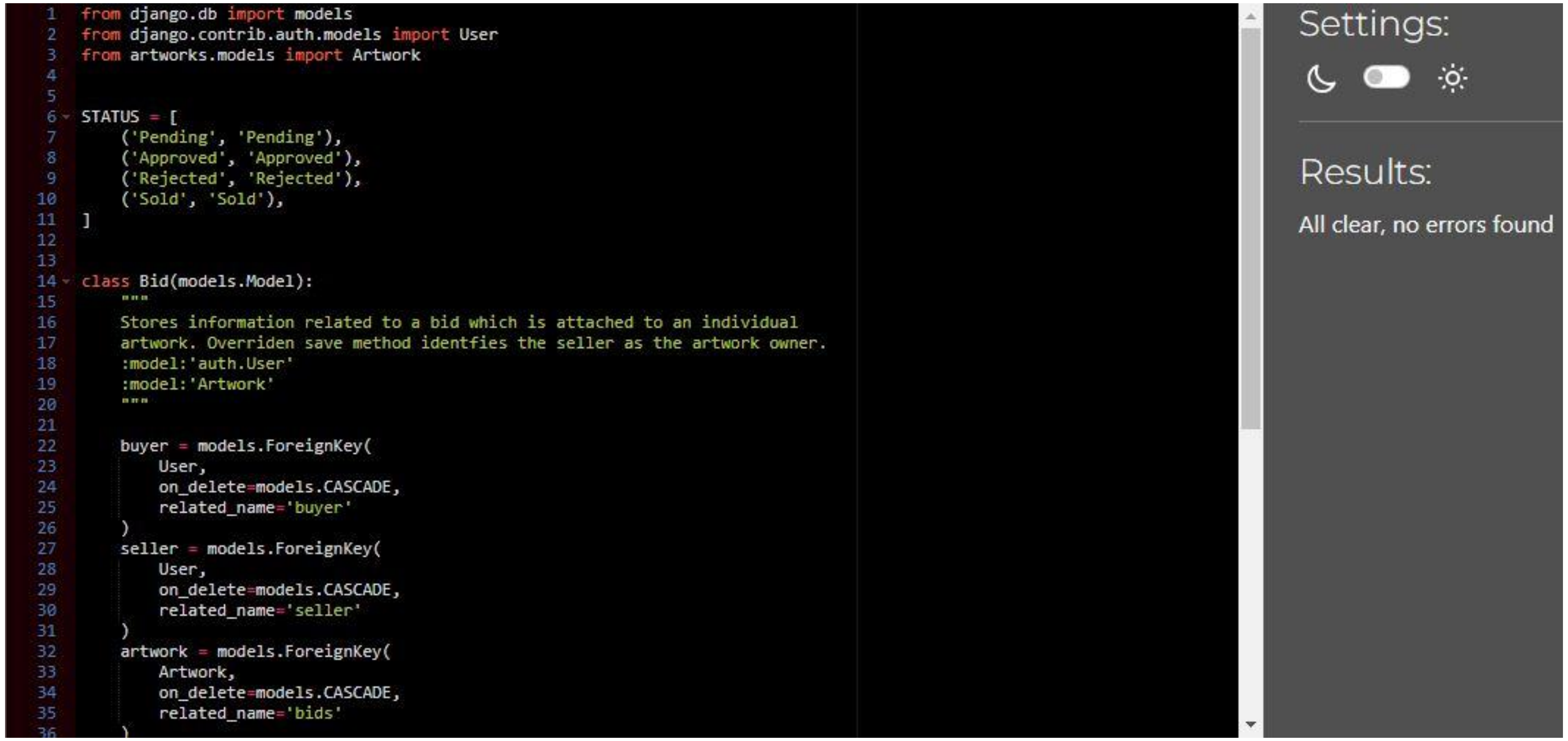


```python
from django.db.models import Count
from rest_framework import generics, permissions, filters
from django_filters.rest_framework import DjangoFilterBackend
from artly_api.permissions import IsOwnerOrReadOnly
from .models import Artwork
from .serializers import ArtworkSerializer


class ArtworkList(generics.ListCreateAPIView):
    """
    Displays all artworks in a list. Allows to filter and search the list based
    on the needed criteria.
    """
    serializer_class = ArtworkSerializer
    permission_classes = [permissions.IsAuthenticatedOrReadOnly]
    queryset = Artwork.objects.annotate(
        bids_count=Count('bids', distinct=True)).order_by('-updated_at')
    filter_backends = [
        filters.OrderingFilter,
        filters.SearchFilter,
        DjangoFilterBackend
    ]
    ordering_fields = [
        'style',
        'type',
        'bids_count',
        'created_at'
    ]
    search_fields = [
        'artwork_title',
        'artist_name',
        'location',
        'payment_method',
        'style',
        'type',
        'owner__username'
```

Results:

All clear, no errors found

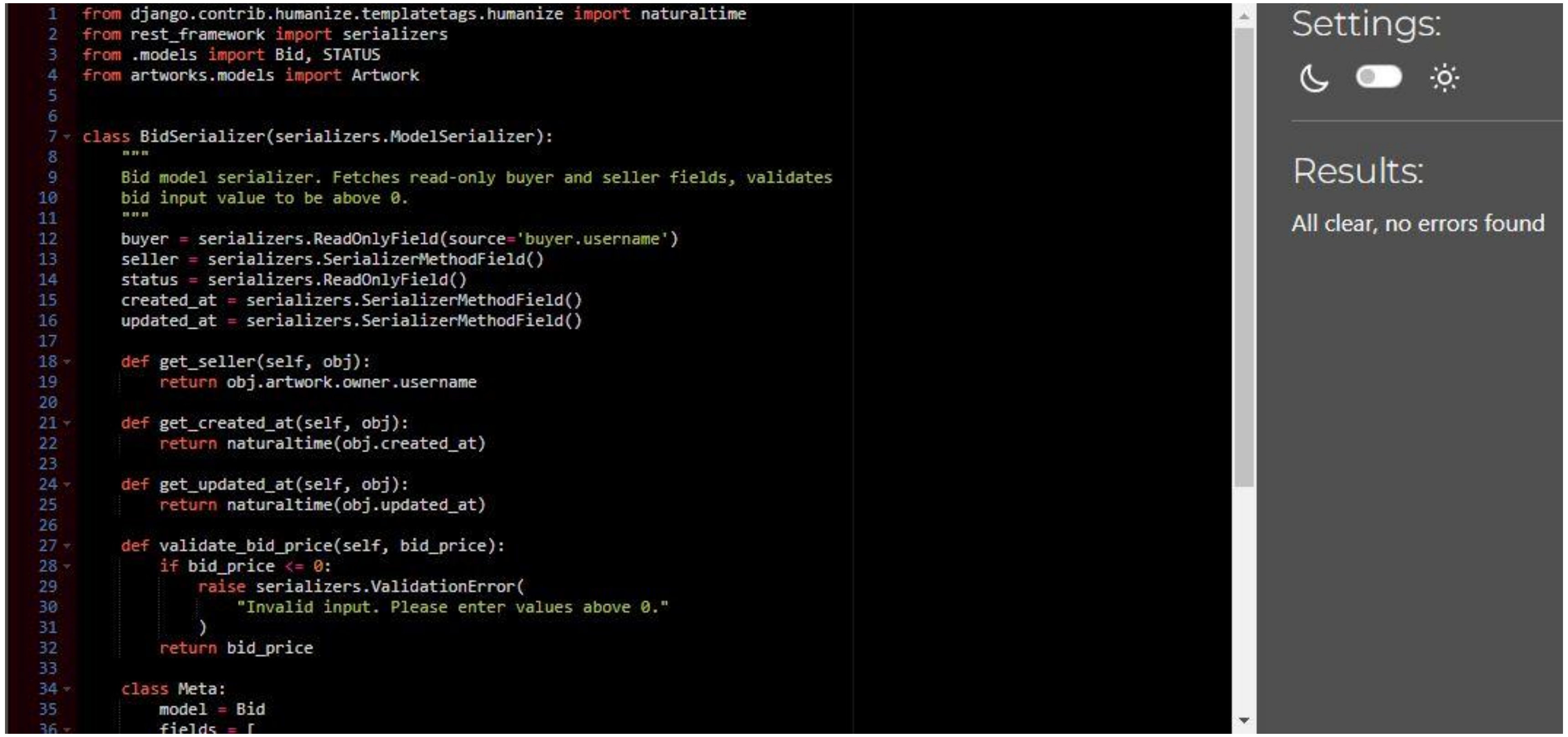
## Bids app


```
from django.db import models
from django.contrib.auth.models import User
from artworks.models import Artwork


STATUS = [
    ('Pending', 'Pending'),
    ('Approved', 'Approved'),
    ('Rejected', 'Rejected'),
    ('Sold', 'Sold'),
]


class Bid(models.Model):
    """
    Stores information related to a bid which is attached to an individual
    artwork. Overriden save method identfies the seller as the artwork owner.
    :model:'auth.User'
    :model:'Artwork'
    """

    buyer = models.ForeignKey(
        User,
        on_delete=models.CASCADE,
        related_name='buyer'
    )
    seller = models.ForeignKey(
        User,
        on_delete=models.CASCADE,
        related_name='seller'
    )
    artwork = models.ForeignKey(
        Artwork,
        on_delete=models.CASCADE,
        related_name='bids'
    )
```

Settings:

Results:

All clear, no errors found

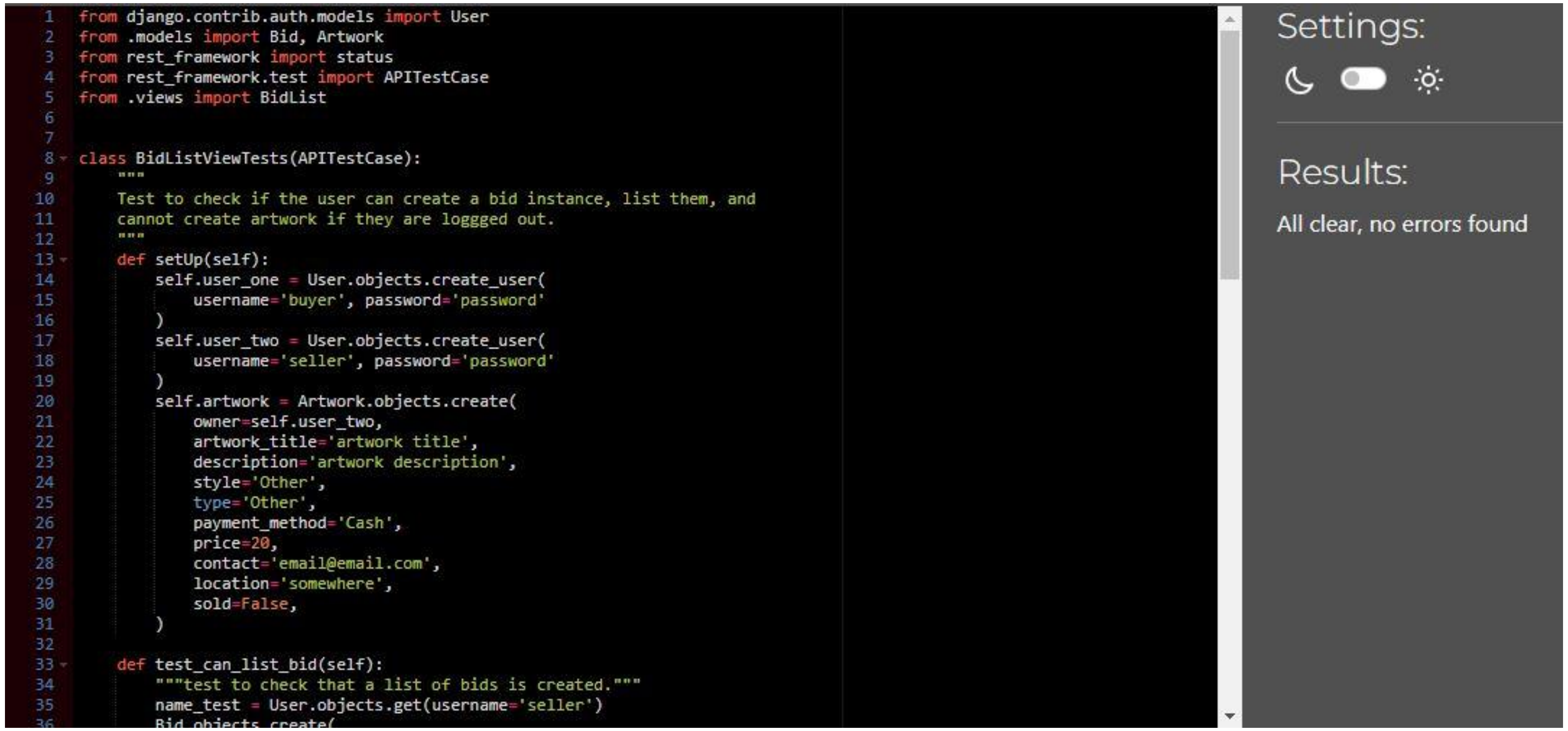


```
from django.contrib.humanize.templatetags.humanize import naturaltime
from rest_framework import serializers
from .models import Bid, STATUS
from artworks.models import Artwork


class BidSerializer(serializers.ModelSerializer):
    """
    Bid model serializer. Fetches read-only buyer and seller fields, validates
    bid input value to be above 0.
    """
    buyer = serializers.ReadOnlyField(source='buyer.username')
    seller = serializers.SerializerMethodField()
    status = serializers.ReadOnlyField()
    created_at = serializers.SerializerMethodField()
    updated_at = serializers.SerializerMethodField()

    def get_seller(self, obj):
        return obj.artwork.owner.username

    def get_created_at(self, obj):
        return naturaltime(obj.created_at)

    def get_updated_at(self, obj):
        return naturaltime(obj.updated_at)

    def validate_bid_price(self, bid_price):
        if bid_price <= 0:
            raise serializers.ValidationError(
                "Invalid input. Please enter values above 0."
            )
        return bid_price

    class Meta:
        model = Bid
        fields = [
```

Settings:

Results:

All clear, no errors found

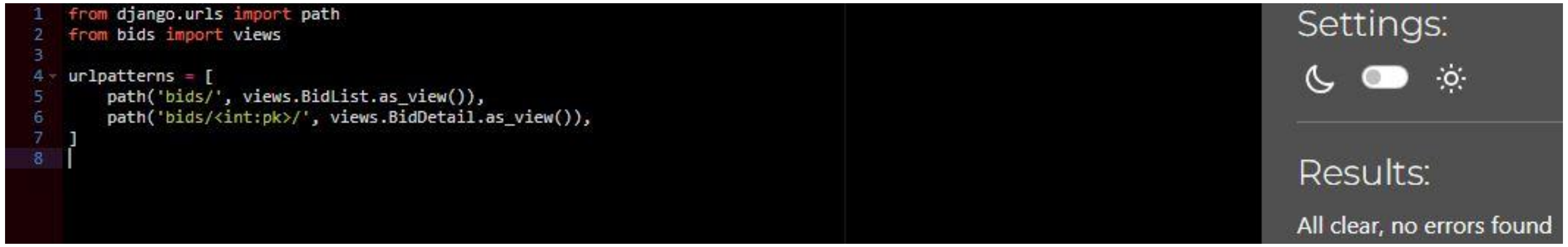


```python
from django.contrib.auth.models import User
from .models import Bid, Artwork
from rest_framework import status
from rest_framework.test import APITestCase
from .views import BidList


class BidListViewTests(APITestCase):
    """
    Test to check if the user can create a bid instance, list them, and
    cannot create artwork if they are loggged out.
    """
    def setUp(self):
        self.user_one = User.objects.create_user(
            username='buyer', password='password'
        )
        self.user_two = User.objects.create_user(
            username='seller', password='password'
        )
        self.artwork = Artwork.objects.create(
            owner=self.user_two,
            artwork_title='artwork title',
            description='artwork description',
            style='Other',
            type='Other',
            payment_method='Cash',
            price=20,
            contact='email@email.com',
            location='somewhere',
            sold=False,
        )

    def test_can_list_bid(self):
        """test to check that a list of bids is created."""
        name_test = User.objects.get(username='seller')
        Bid.objects.create(
```

Settings:

Results:

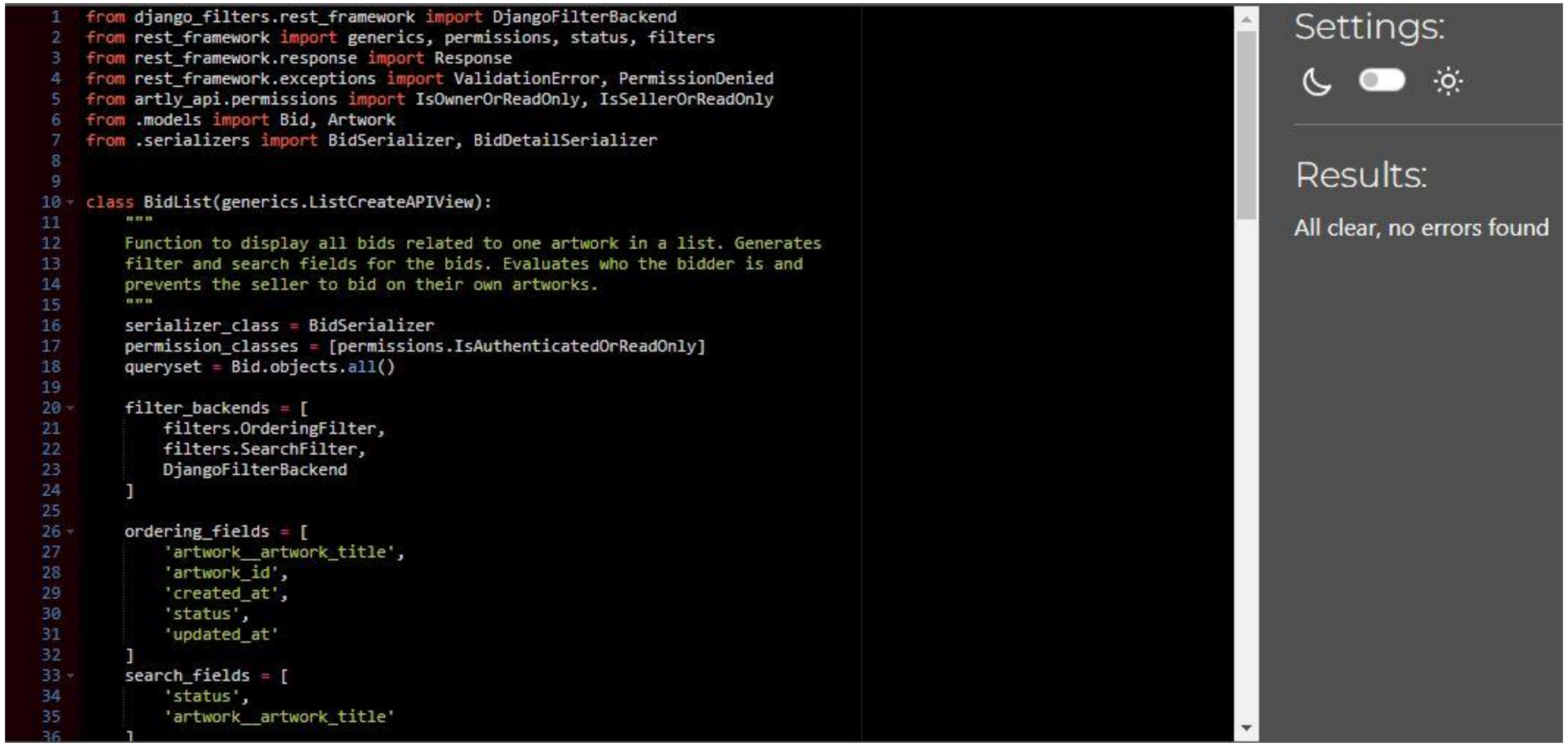
All clear, no errors found



```
from django.urls import path
from bids import views

urlpatterns = [
    path('bids/', views.BidList.as_view()),
    path('bids/<int:pk>/', views.BidDetail.as_view()),
]
```

Results:

All clear, no errors found



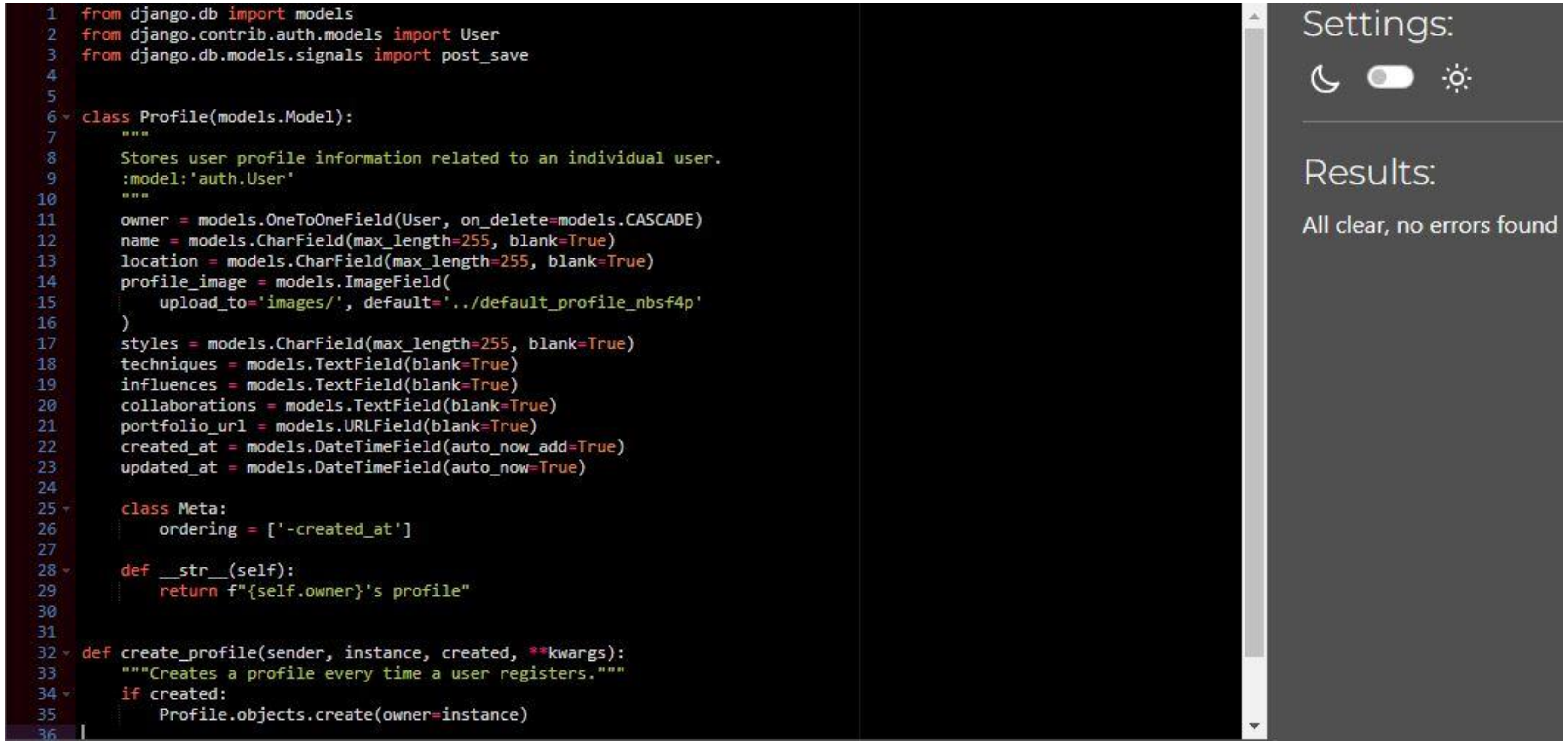
```python
from django_filters.rest_framework import DjangoFilterBackend
from rest_framework import generics, permissions, status, filters
from rest_framework.response import Response
from rest_framework.exceptions import ValidationError, PermissionDenied
from artly_api.permissions import IsOwnerOrReadOnly, IsSellerOrReadOnly
from .models import Bid, Artwork
from .serializers import BidSerializer, BidDetailSerializer


class BidList(generics.ListCreateAPIView):
    """
    Function to display all bids related to one artwork in a list. Generates
    filter and search fields for the bids. Evaluates who the bidder is and
    prevents the seller to bid on their own artworks.
    """
    serializer_class = BidSerializer
    permission_classes = [permissions.IsAuthenticatedOrReadOnly]
    queryset = Bid.objects.all()

    filter_backends = [
        filters.OrderingFilter,
        filters.SearchFilter,
        DjangoFilterBackend
    ]

    ordering_fields = [
        'artwork__artwork_title',
        'artwork_id',
        'created_at',
        'status',
        'updated_at'
    ]
    search_fields = [
        'status',
        'artwork__artwork_title'
    ]
```

Settings:

Results:

All clear, no errors found

## Profiles app

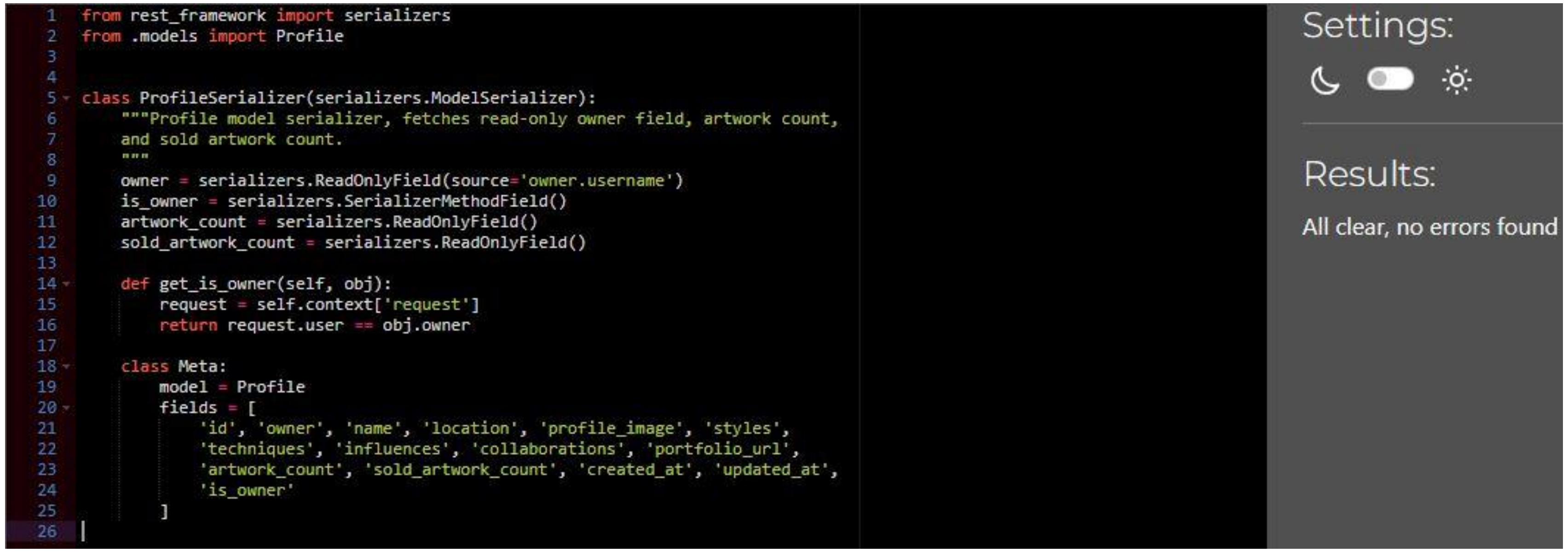


```python
from django.db import models
from django.contrib.auth.models import User
from django.db.models.signals import post_save


class Profile(models.Model):
    """
    Stores user profile information related to an individual user.
    :model:'auth.User'
    """
    owner = models.OneToOneField(User, on_delete=models.CASCADE)
    name = models.CharField(max_length=255, blank=True)
    location = models.CharField(max_length=255, blank=True)
    profile_image = models.ImageField(
        upload_to='images/', default='../default_profile_nbsf4p'
    )
    styles = models.CharField(max_length=255, blank=True)
    techniques = models.TextField(blank=True)
    influences = models.TextField(blank=True)
    collaborations = models.TextField(blank=True)
    portfolio_url = models.URLField(blank=True)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    class Meta:
        ordering = ['-created_at']

    def __str__(self):
        return f"{self.owner}'s profile"


def create_profile(sender, instance, created, **kwargs):
    """Creates a profile every time a user registers."""
    if created:
        Profile.objects.create(owner=instance)
```

Settings:

Results:

All clear, no errors found

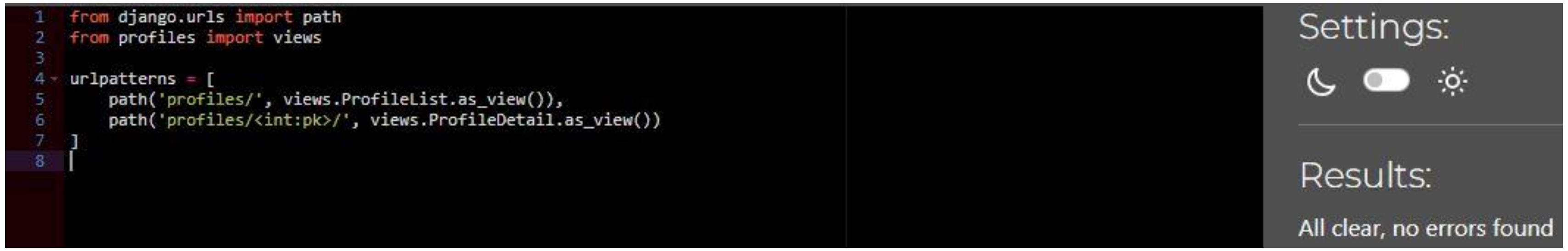


```python
from rest_framework import serializers
from .models import Profile


class ProfileSerializer(serializers.ModelSerializer):
    """Profile model serializer, fetches read-only owner field, artwork count,
    and sold artwork count.
    """
    owner = serializers.ReadOnlyField(source='owner.username')
    is_owner = serializers.SerializerMethodField()
    artwork_count = serializers.ReadOnlyField()
    sold_artwork_count = serializers.ReadOnlyField()

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

    class Meta:
        model = Profile
        fields = [
            'id', 'owner', 'name', 'location', 'profile_image', 'styles',
            'techniques', 'influences', 'collaborations', 'portfolio_url',
            'artwork_count', 'sold_artwork_count', 'created_at', 'updated_at',
            'is_owner'
        ]

```

Settings:

Results:

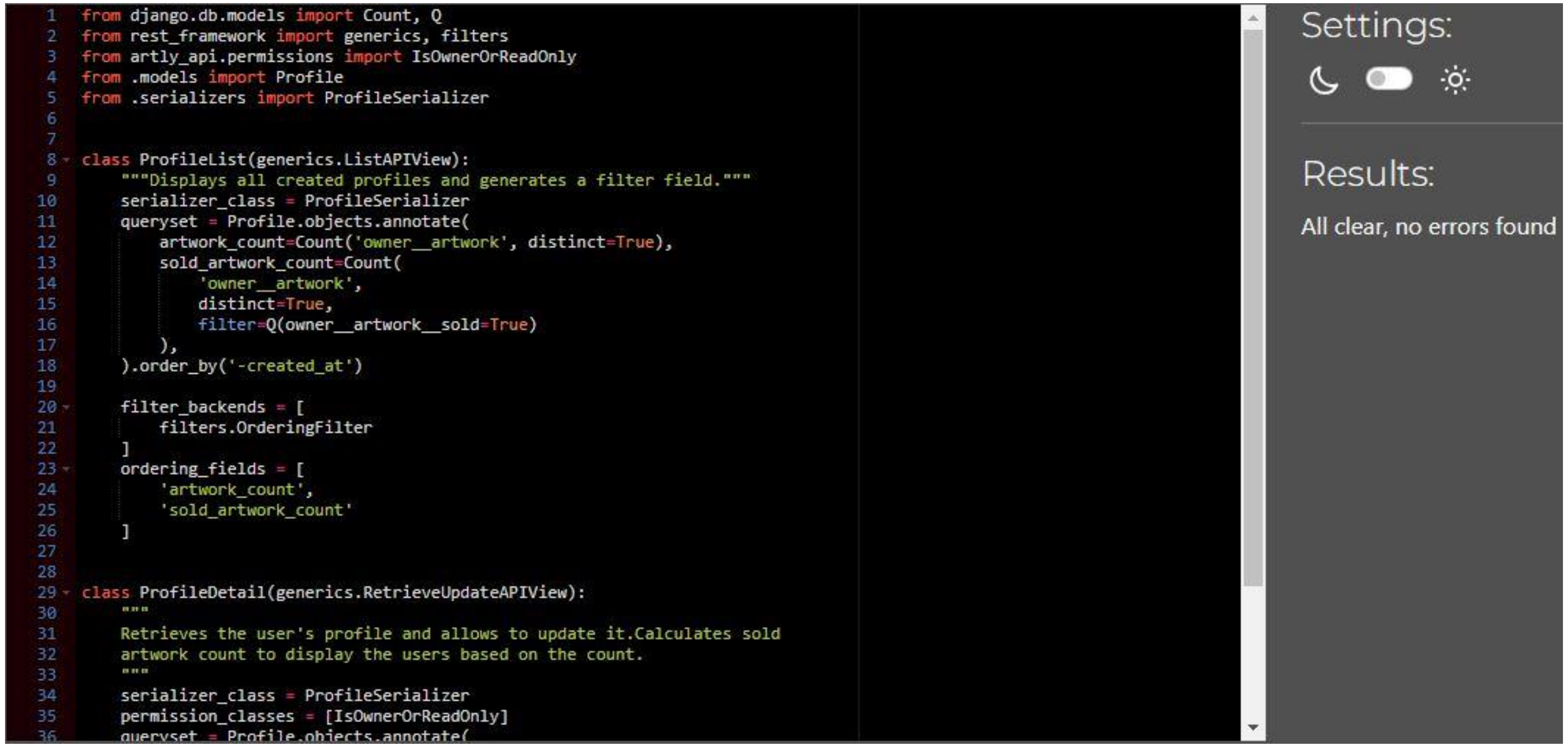
All clear, no errors found



```python
from django.urls import path
from profiles import views

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

Results:

All clear, no errors found



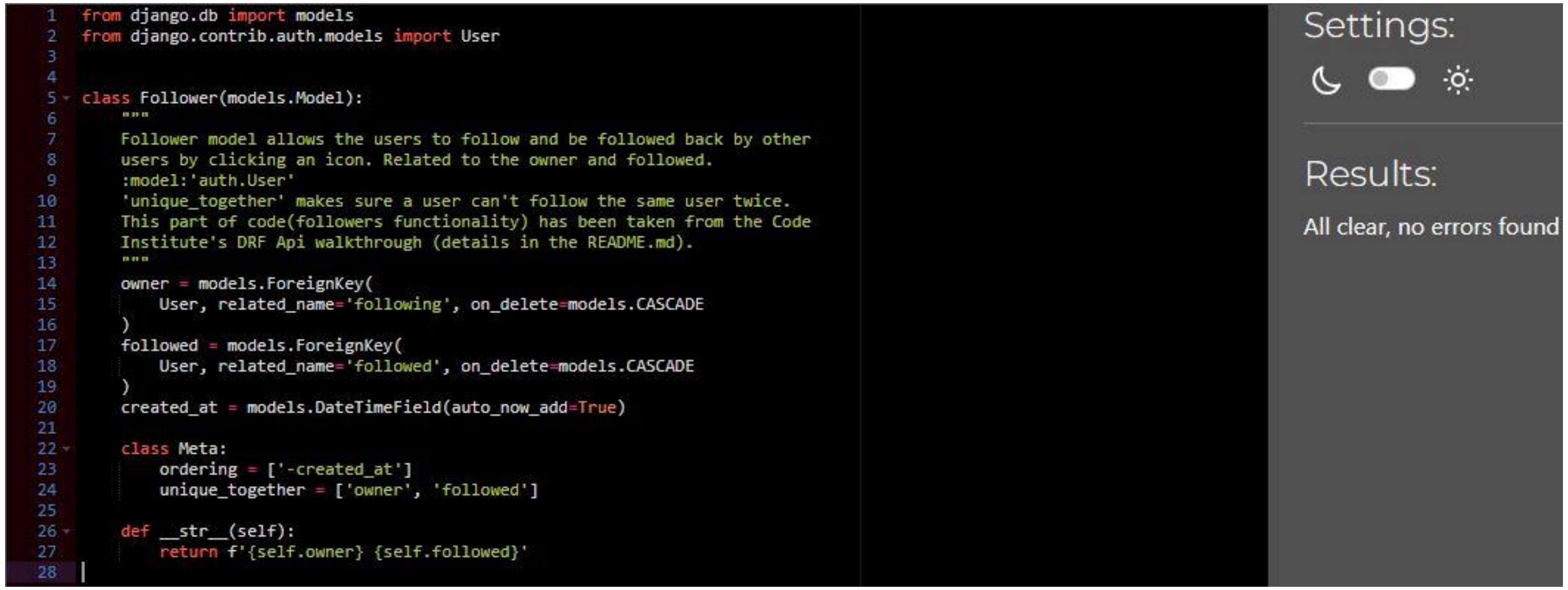
```python
from django.db.models import Count, Q
from rest_framework import generics, filters
from artly_api.permissions import IsOwnerOrReadOnly
from .models import Profile
from .serializers import ProfileSerializer


class ProfileList(generics.ListAPIView):
    """Displays all created profiles and generates a filter field."""
    serializer_class = ProfileSerializer
    queryset = Profile.objects.annotate(
        artwork_count=Count('owner__artwork', distinct=True),
        sold_artwork_count=Count(
            'owner__artwork',
            distinct=True,
            filter=Q(owner__artwork__sold=True)
        ),
    ).order_by('-created_at')

    filter_backends = [
        filters.OrderingFilter
    ]
    ordering_fields = [
        'artwork_count',
        'sold_artwork_count'
    ]


class ProfileDetail(generics.RetrieveUpdateAPIView):
    """
    Retrieves the user's profile and allows to update it.Calculates sold
    artwork count to display the users based on the count.
    """
    serializer_class = ProfileSerializer
    permission_classes = [IsOwnerOrReadOnly]
    queryset = Profile.objects.annotate(
```

Settings:

Results:

All clear, no errors found

## Followers app

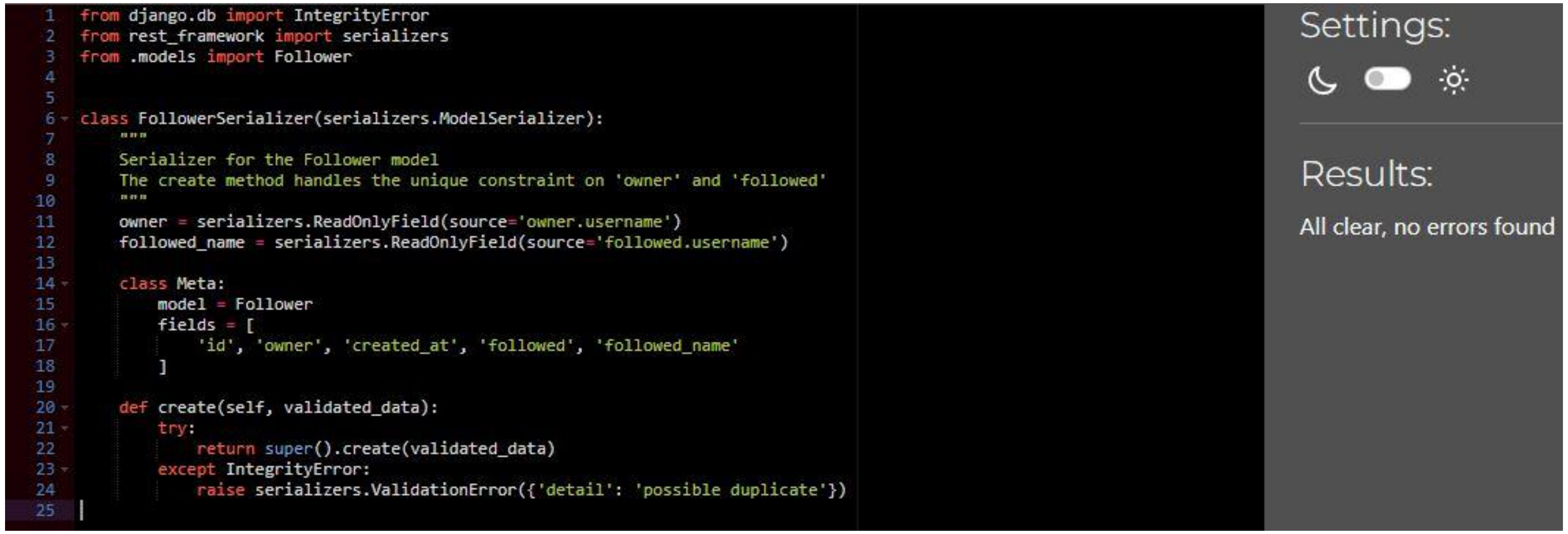


```
from django.db import models
from django.contrib.auth.models import User


class Follower(models.Model):
    """
    Follower model allows the users to follow and be followed back by other
    users by clicking an icon. Related to the owner and followed.
    :model:'auth.User'
    'unique_together' makes sure a user can't follow the same user twice.
    This part of code(followers functionality) has been taken from the Code
    Institute's DRF Api walkthrough (details in the README.md).
    """
    owner = models.ForeignKey(
        User, related_name='following', on_delete=models.CASCADE
    )
    followed = models.ForeignKey(
        User, related_name='followed', on_delete=models.CASCADE
    )
    created_at = models.DateTimeField(auto_now_add=True)

    class Meta:
        ordering = ['-created_at']
        unique_together = ['owner', 'followed']

    def __str__(self):
        return f'{self.owner} {self.followed}'

```

Settings:

Results:

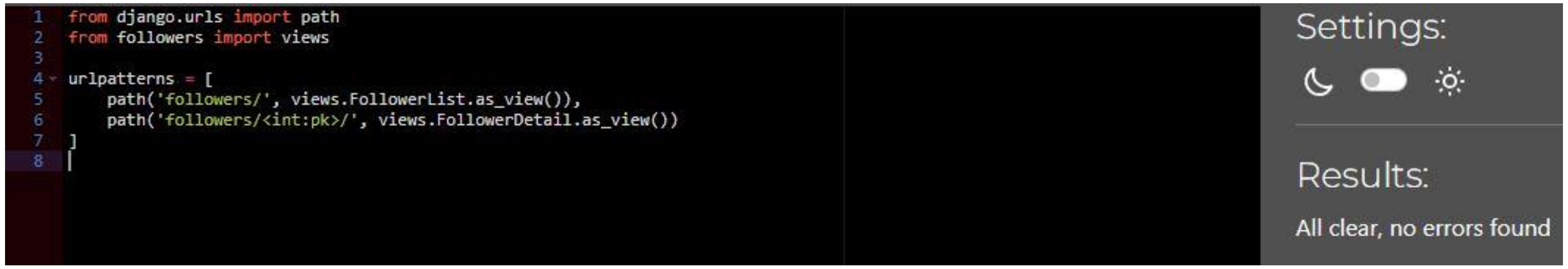
All clear, no errors found



```python
from django.db import IntegrityError
from rest_framework import serializers
from .models import Follower


class FollowerSerializer(serializers.ModelSerializer):
    """
    Serializer for the Follower model
    The create method handles the unique constraint on 'owner' and 'followed'
    """
    owner = serializers.ReadOnlyField(source='owner.username')
    followed_name = serializers.ReadOnlyField(source='followed.username')

    class Meta:
        model = Follower
        fields = [
            'id', 'owner', 'created_at', 'followed', 'followed_name'
        ]

    def create(self, validated_data):
        try:
            return super().create(validated_data)
        except IntegrityError:
            raise serializers.ValidationError({'detail': 'possible duplicate'})

```

Results:

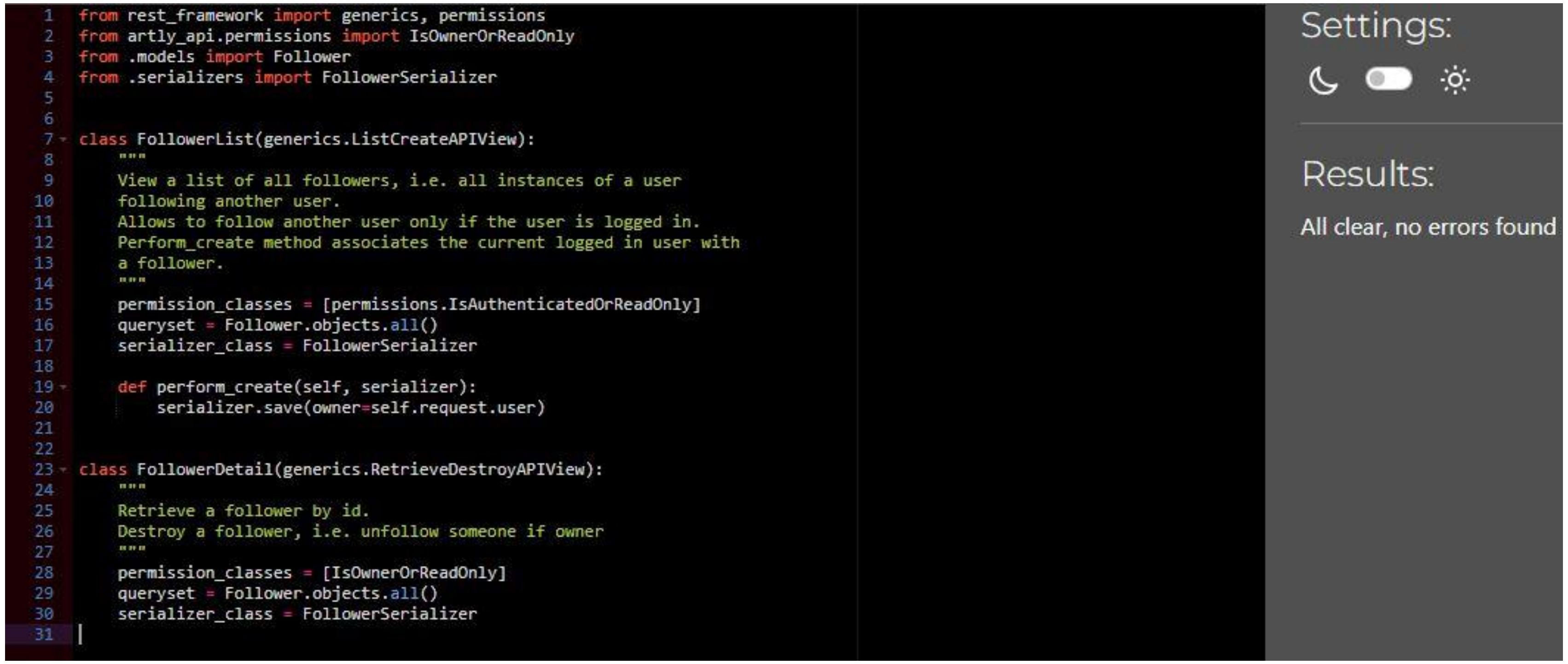
All clear, no errors found



```
from django.urls import path
from followers import views

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

Results:

All clear, no errors found



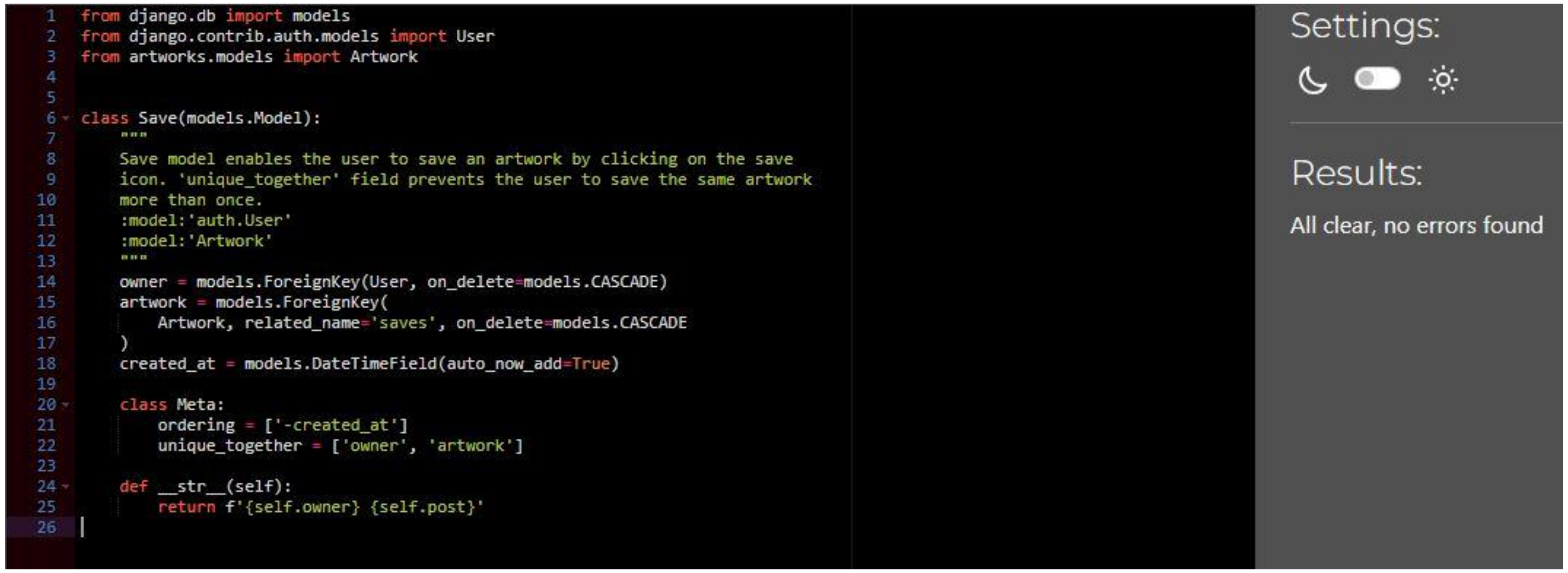
```
from rest_framework import generics, permissions
from artly_api.permissions import IsOwnerOrReadOnly
from .models import Follower
from .serializers import FollowerSerializer


class FollowerList(generics.ListCreateAPIView):
    """
    View a list of all followers, i.e. all instances of a user
    following another user.
    Allows to follow another user only if the user is logged in.
    Perform_create method associates the current logged in user with
    a follower.
    """
    permission_classes = [permissions.IsAuthenticatedOrReadOnly]
    queryset = Follower.objects.all()
    serializer_class = FollowerSerializer

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


class FollowerDetail(generics.RetrieveDestroyAPIView):
    """
    Retrieve a follower by id.
    Destroy a follower, i.e. unfollow someone if owner
    """
    permission_classes = [IsOwnerOrReadOnly]
    queryset = Follower.objects.all()
    serializer_class = FollowerSerializer
```

Settings:

Results:

All clear, no errors found

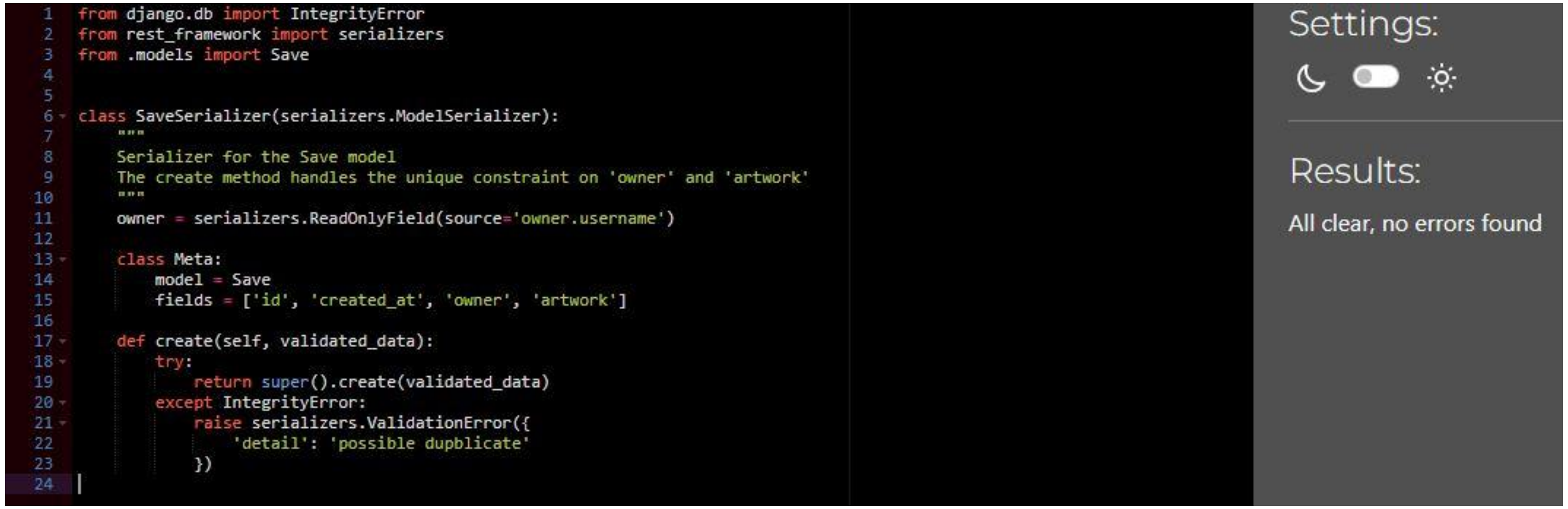
Saves app



```python
from django.db import models
from django.contrib.auth.models import User
from artworks.models import Artwork


class Save(models.Model):
    """
    Save model enables the user to save an artwork by clicking on the save
    icon. 'unique_together' field prevents the user to save the same artwork
    more than once.
    :model:'auth.User'
    :model:'Artwork'
    """
    owner = models.ForeignKey(User, on_delete=models.CASCADE)
    artwork = models.ForeignKey(
        Artwork, related_name='saves', on_delete=models.CASCADE
    )
    created_at = models.DateTimeField(auto_now_add=True)

    class Meta:
        ordering = ['-created_at']
        unique_together = ['owner', 'artwork']

    def __str__(self):
        return f'{self.owner} {self.post}'
```

Results:

All clear, no errors found

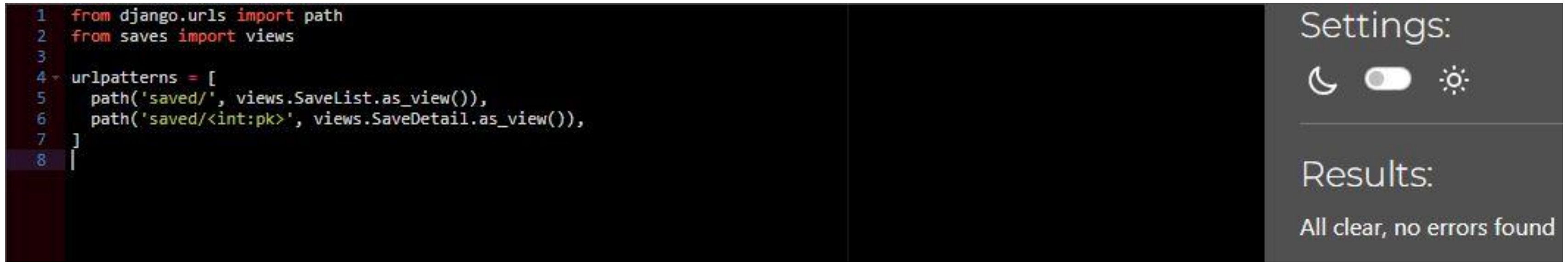


```python
from django.db import IntegrityError
from rest_framework import serializers
from .models import Save


class SaveSerializer(serializers.ModelSerializer):
    """
    Serializer for the Save model
    The create method handles the unique constraint on 'owner' and 'artwork'
    """
    owner = serializers.ReadOnlyField(source='owner.username')

    class Meta:
        model = Save
        fields = ['id', 'created_at', 'owner', 'artwork']

    def create(self, validated_data):
        try:
            return super().create(validated_data)
        except IntegrityError:
            raise serializers.ValidationError({
                'detail': 'possible dupblicate'
            })

```

Settings:

Results:

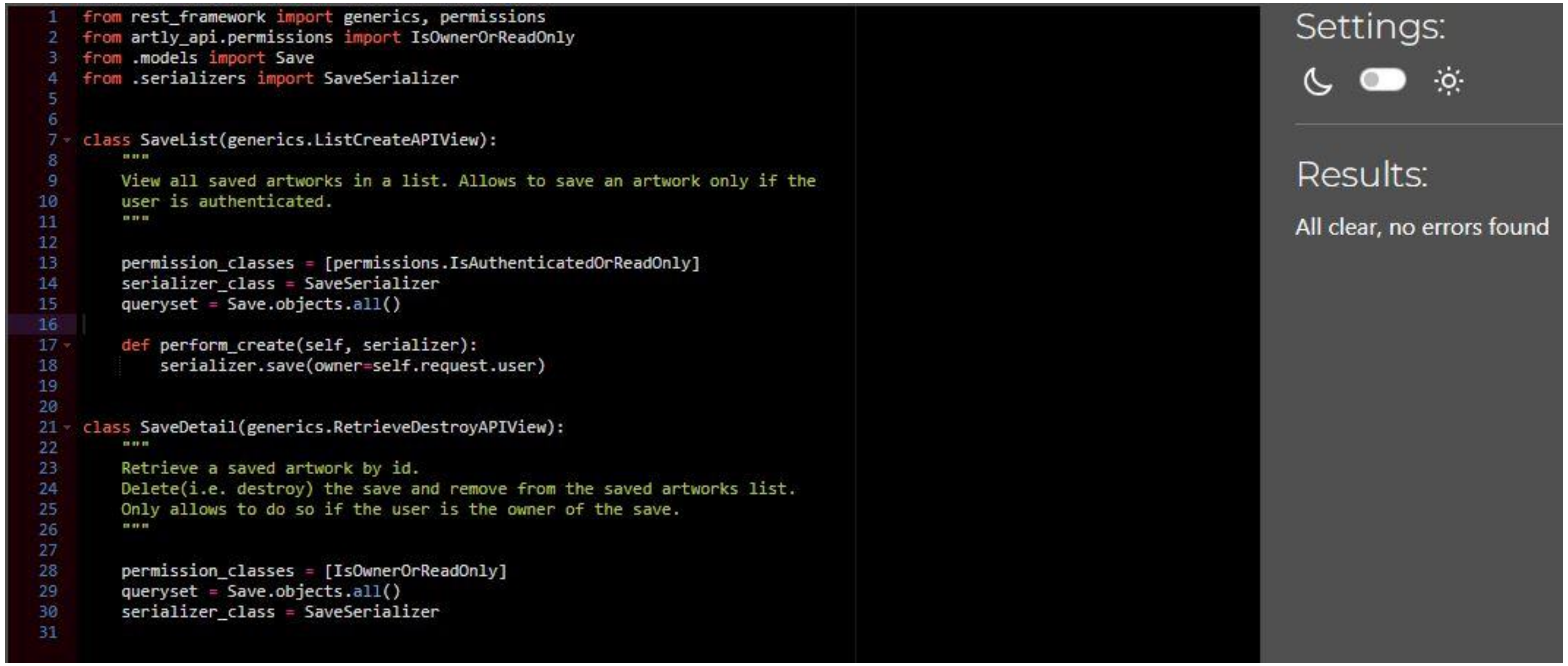
All clear, no errors found



```python
from django.urls import path
from saves import views

urlpatterns = [
  path('saved/', views.SaveList.as_view()),
  path('saved/<int:pk>', views.SaveDetail.as_view()),
]
```

Results:

All clear, no errors found



```python
from rest_framework import generics, permissions
from artly_api.permissions import IsOwnerOrReadOnly
from .models import Save
from .serializers import SaveSerializer


class SaveList(generics.ListCreateAPIView):
    """
    View all saved artworks in a list. Allows to save an artwork only if the
    user is authenticated.
    """

    permission_classes = [permissions.IsAuthenticatedOrReadOnly]
    serializer_class = SaveSerializer
    queryset = Save.objects.all()

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


class SaveDetail(generics.RetrieveDestroyAPIView):
    """
    Retrieve a saved artwork by id.
    Delete(i.e. destroy) the save and remove from the saved artworks list.
    Only allows to do so if the user is the owner of the save.
    """

    permission_classes = [IsOwnerOrReadOnly]
    queryset = Save.objects.all()
    serializer_class = SaveSerializer
```

Settings:

Results:

All clear, no errors found