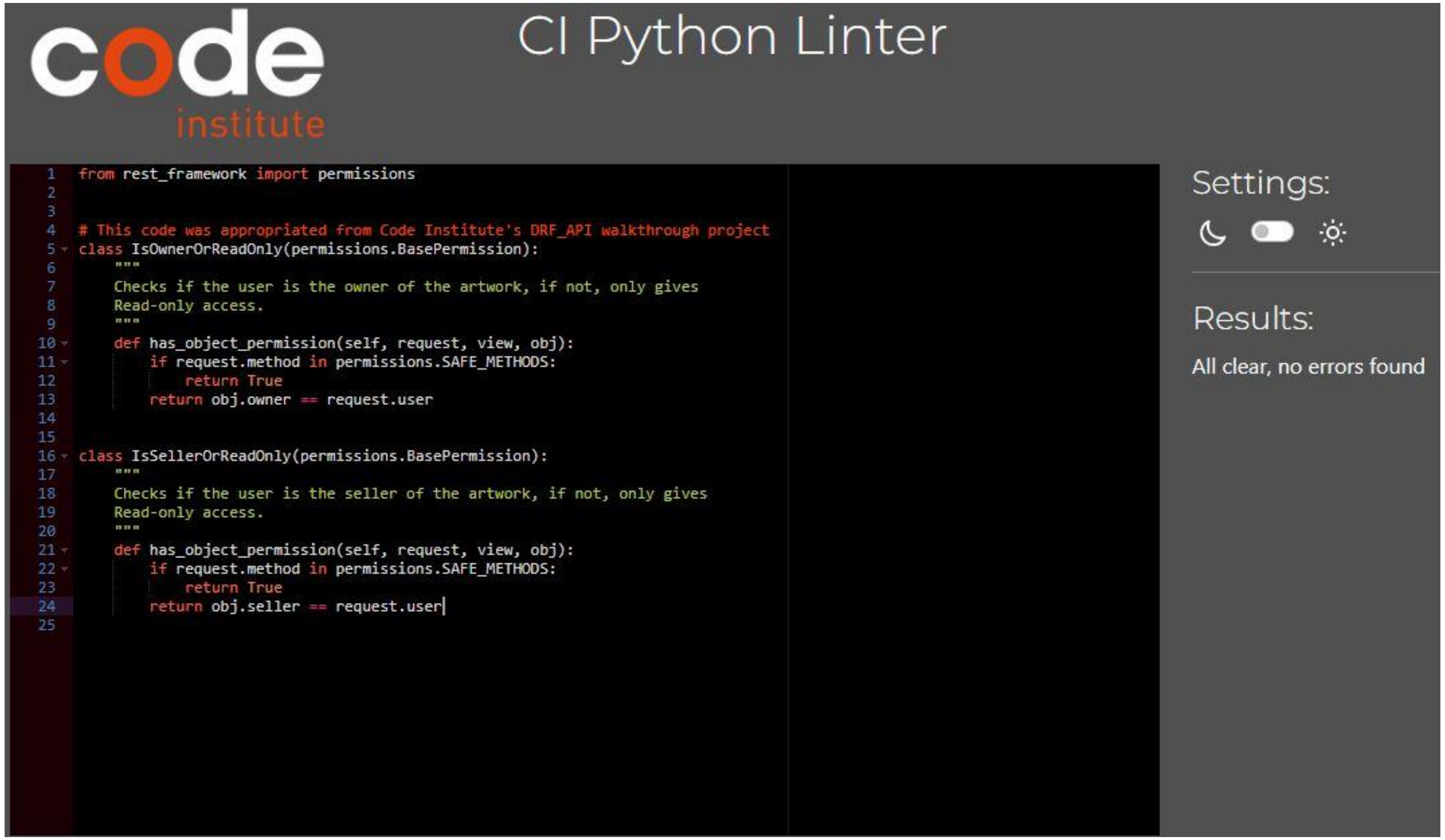


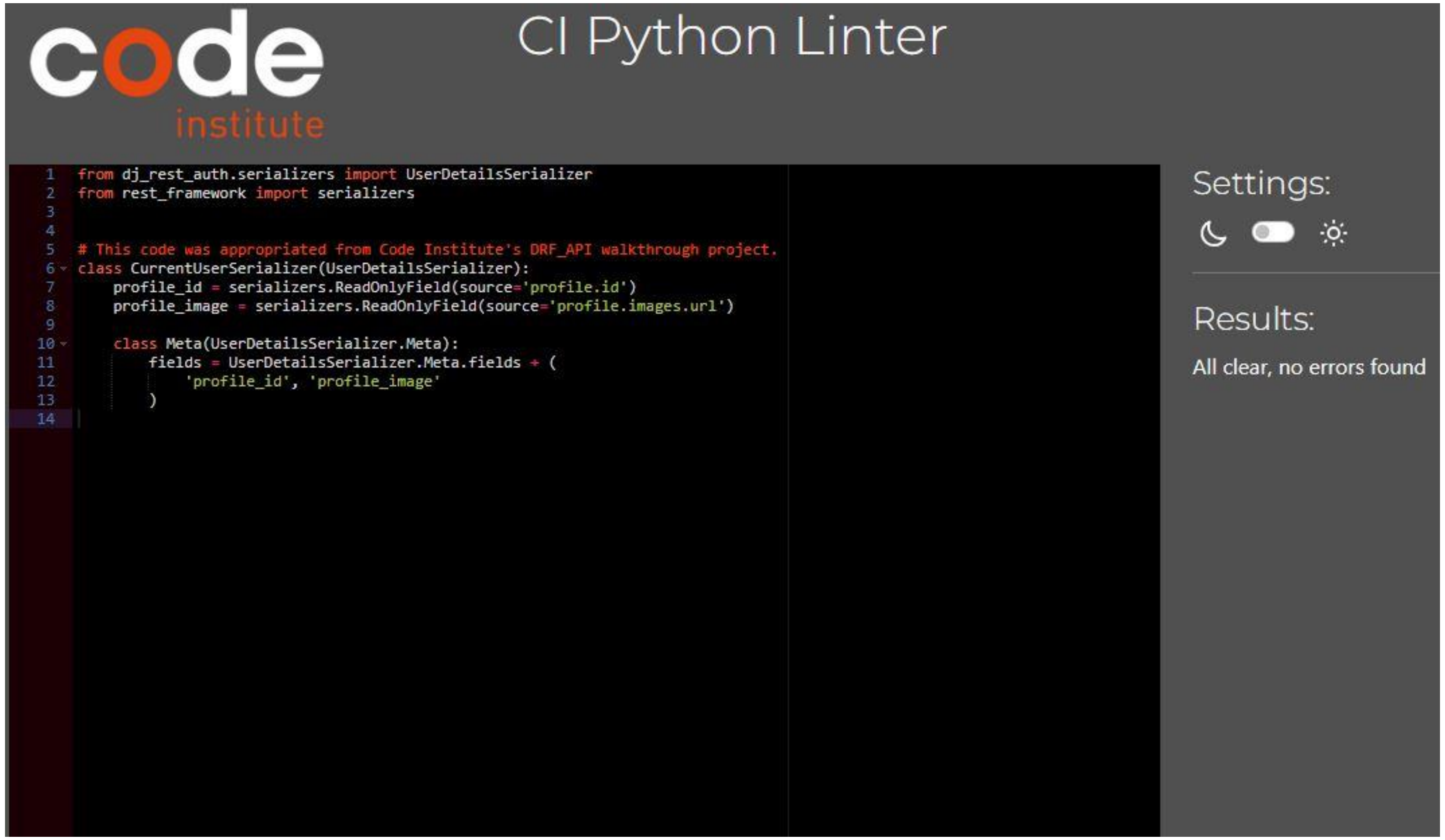
CI Python Linter

```python
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

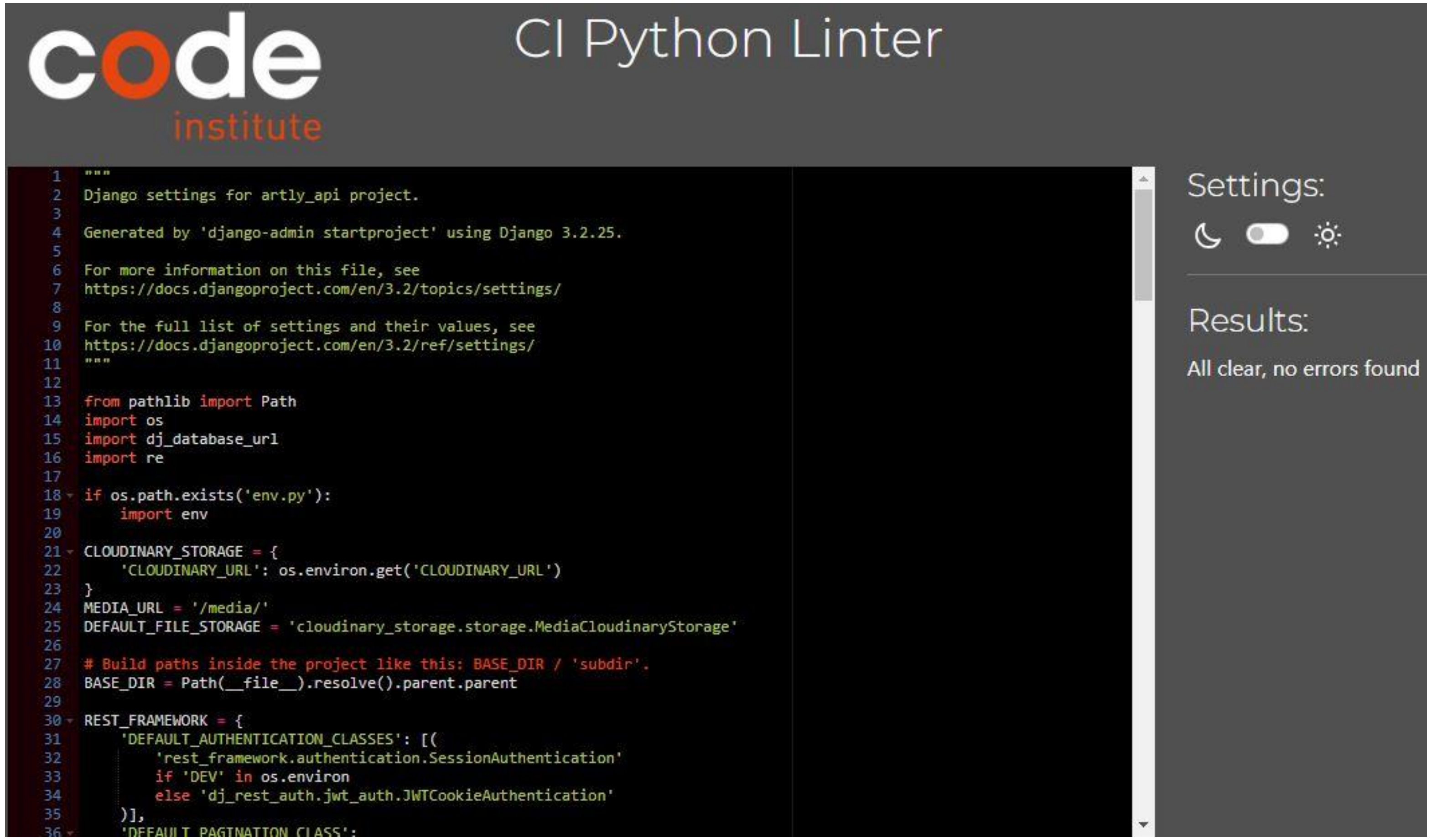


```python
from dj_rest_auth.serializers import UserDetailsSerializer
from rest_framework import serializers


# 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'
        )
```

Settings:

Results:

All clear, no errors found

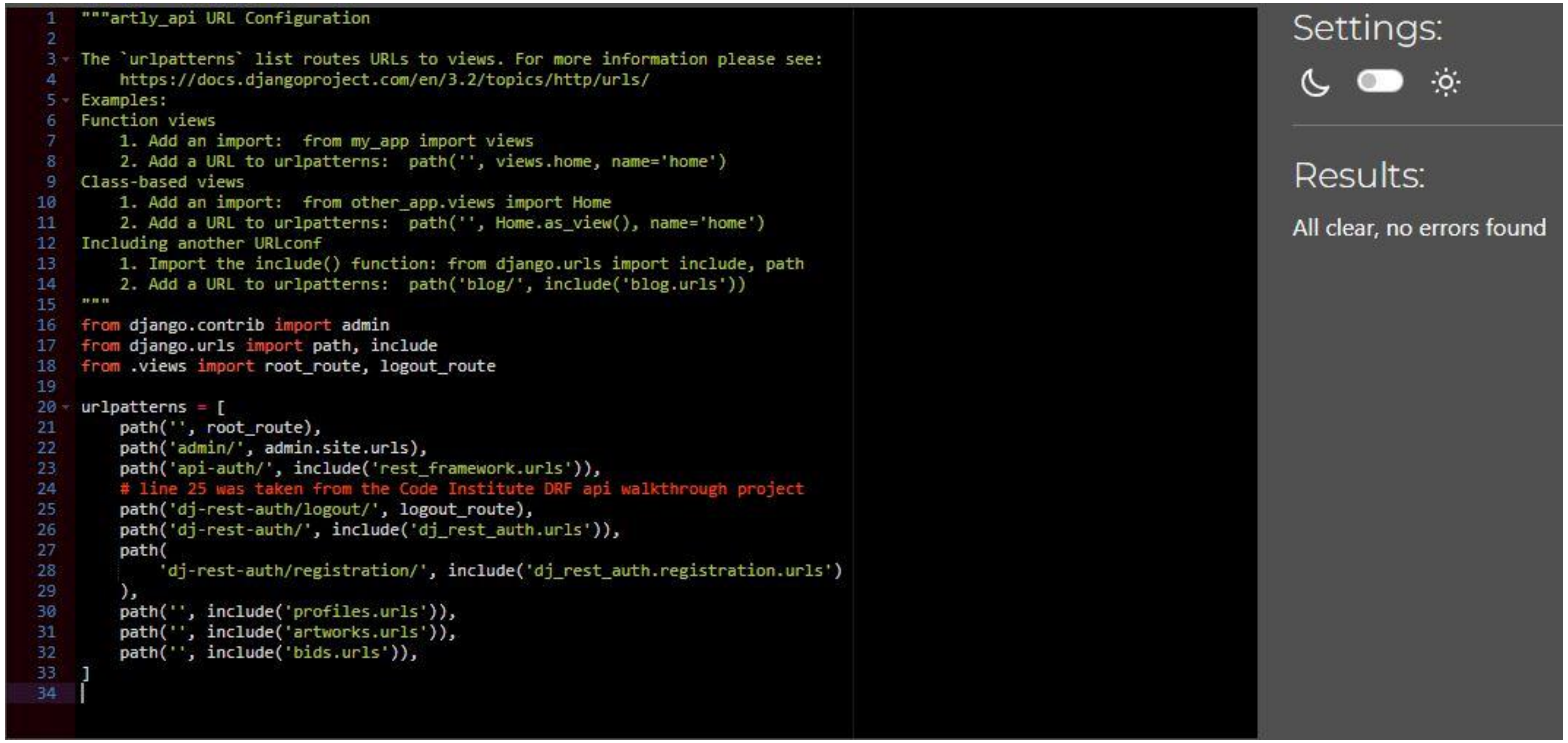


```
"""
Django settings for artly_api project.

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

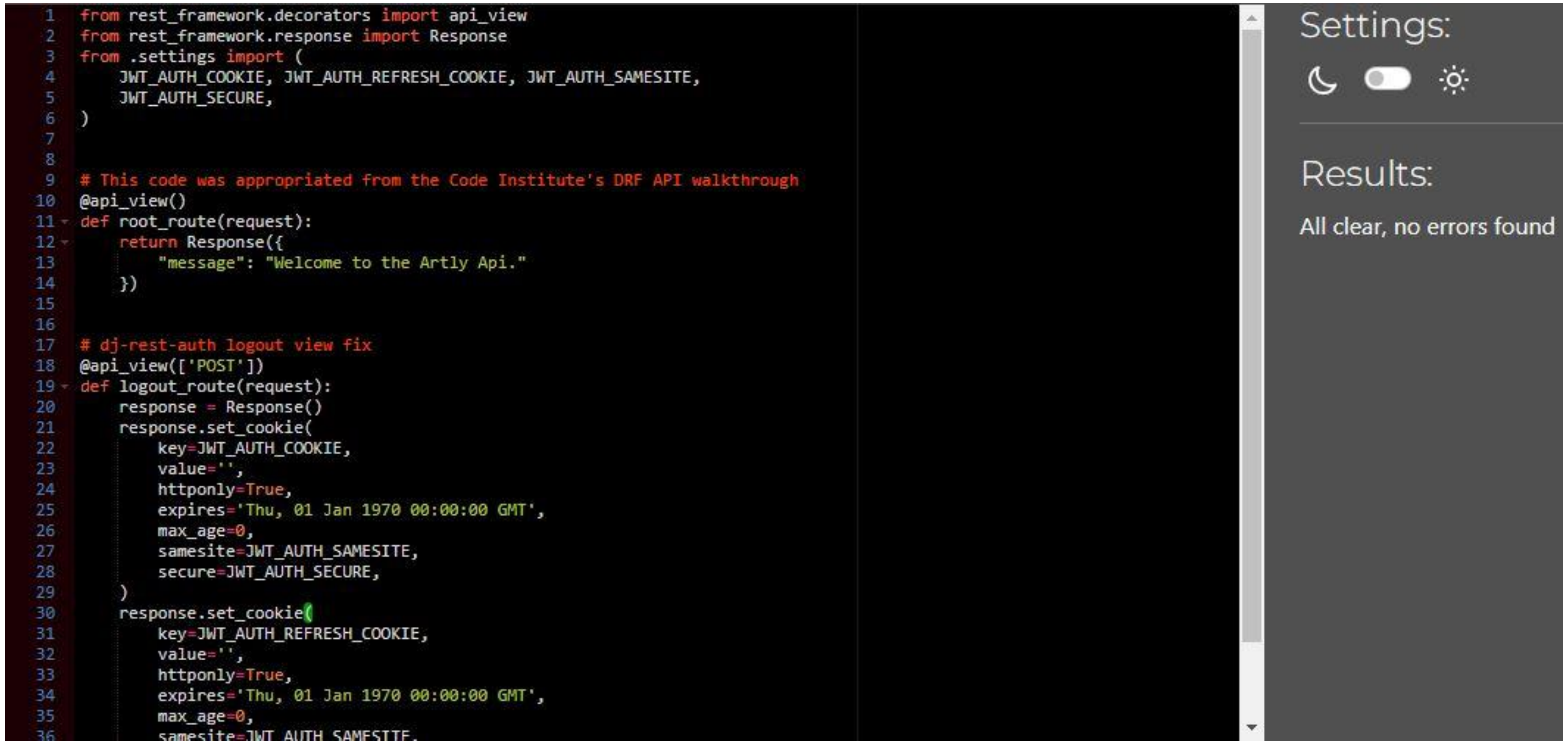


```python
"""artly_api URL Configuration

The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/3.2/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import path, include
from .views import root_route, logout_route

urlpatterns = [
    path('', root_route),
    path('admin/', admin.site.urls),
    path('api-auth/', include('rest_framework.urls')),
    # line 25 was taken from the Code Institute DRF api walkthrough project
    path('dj-rest-auth/logout/', logout_route),
    path('dj-rest-auth/', include('dj_rest_auth.urls')),
    path(
        'dj-rest-auth/registration/', include('dj_rest_auth.registration.urls')
    ),
    path('', include('profiles.urls')),
    path('', include('artworks.urls')),
    path('', include('bids.urls')),
]

```

Settings:

Results:

All clear, no errors found

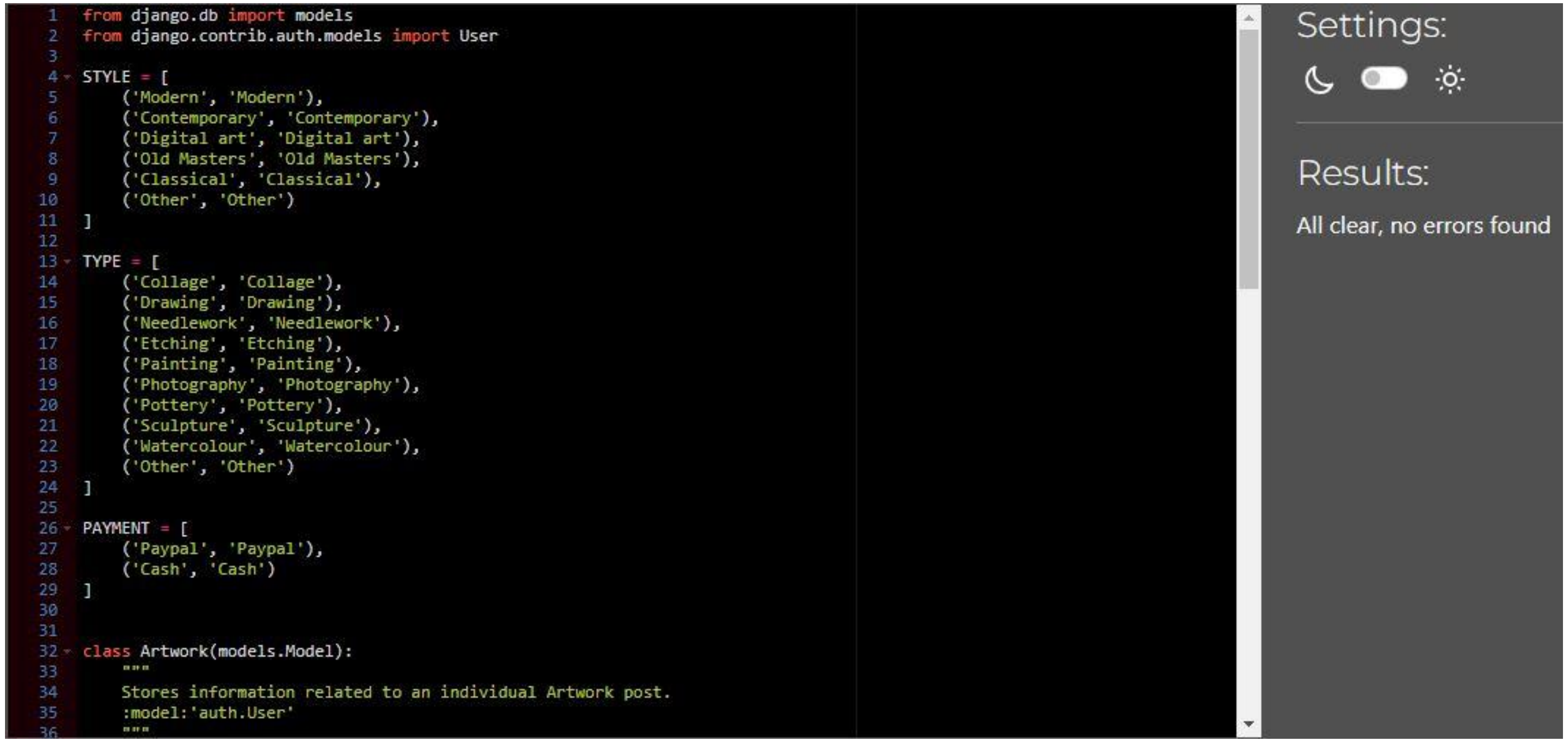


```
from rest_framework.decorators import api_view
from rest_framework.response import Response
from .settings import (
    JWT_AUTH_COOKIE, JWT_AUTH_REFRESH_COOKIE, JWT_AUTH_SAMESITE,
    JWT_AUTH_SECURE,
)


# 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

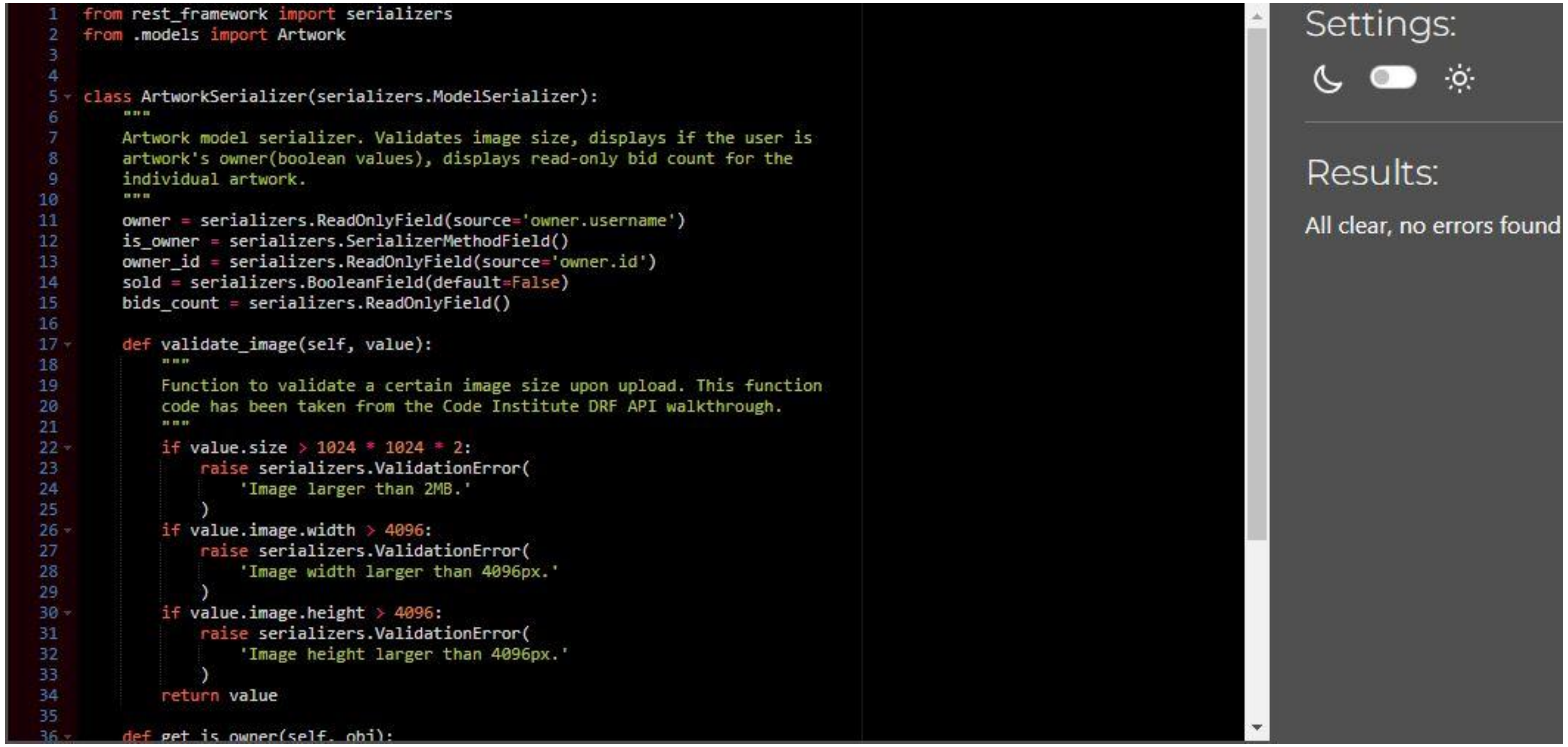


```python
from django.db import models
from django.contrib.auth.models import User

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

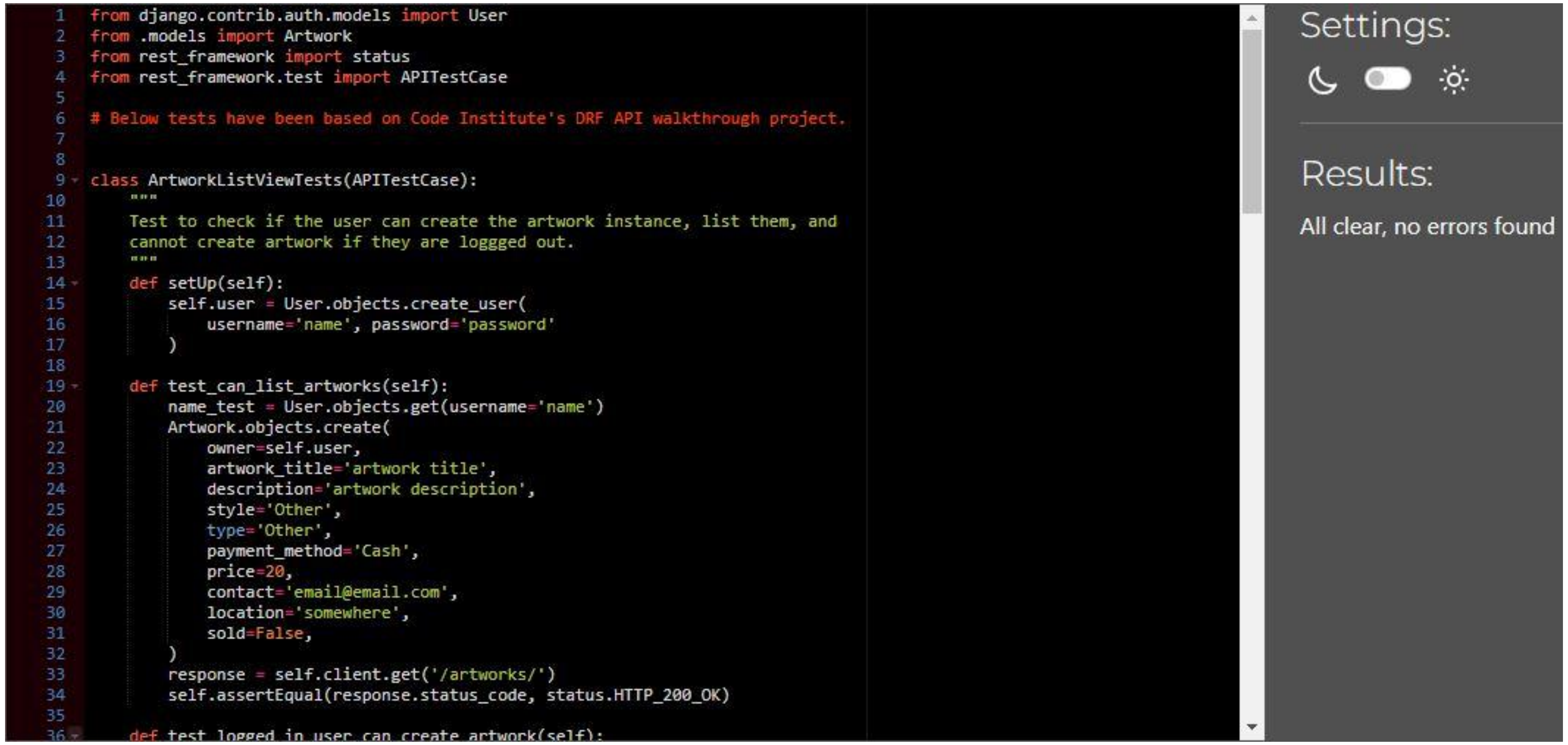


```python
from rest_framework import serializers
from .models import Artwork


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

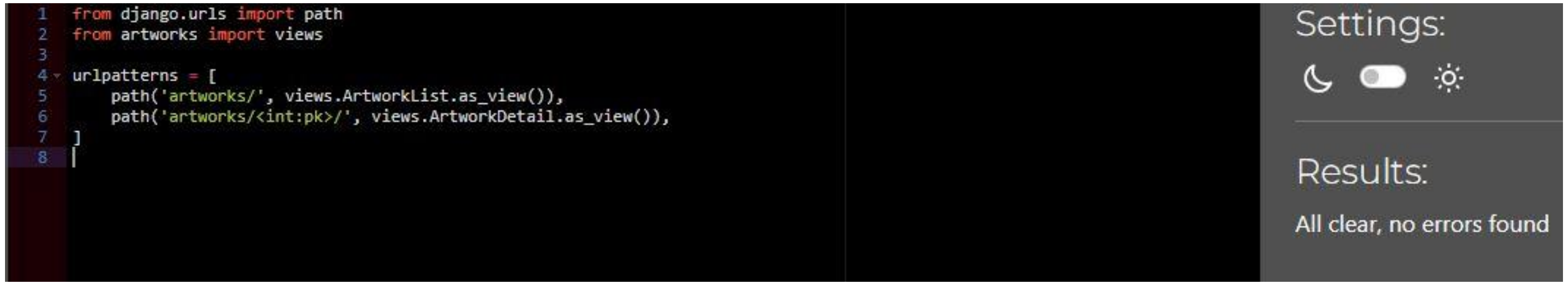


```python
from django.contrib.auth.models import User
from .models import Artwork
from rest_framework import status
from rest_framework.test import APITestCase

# 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:

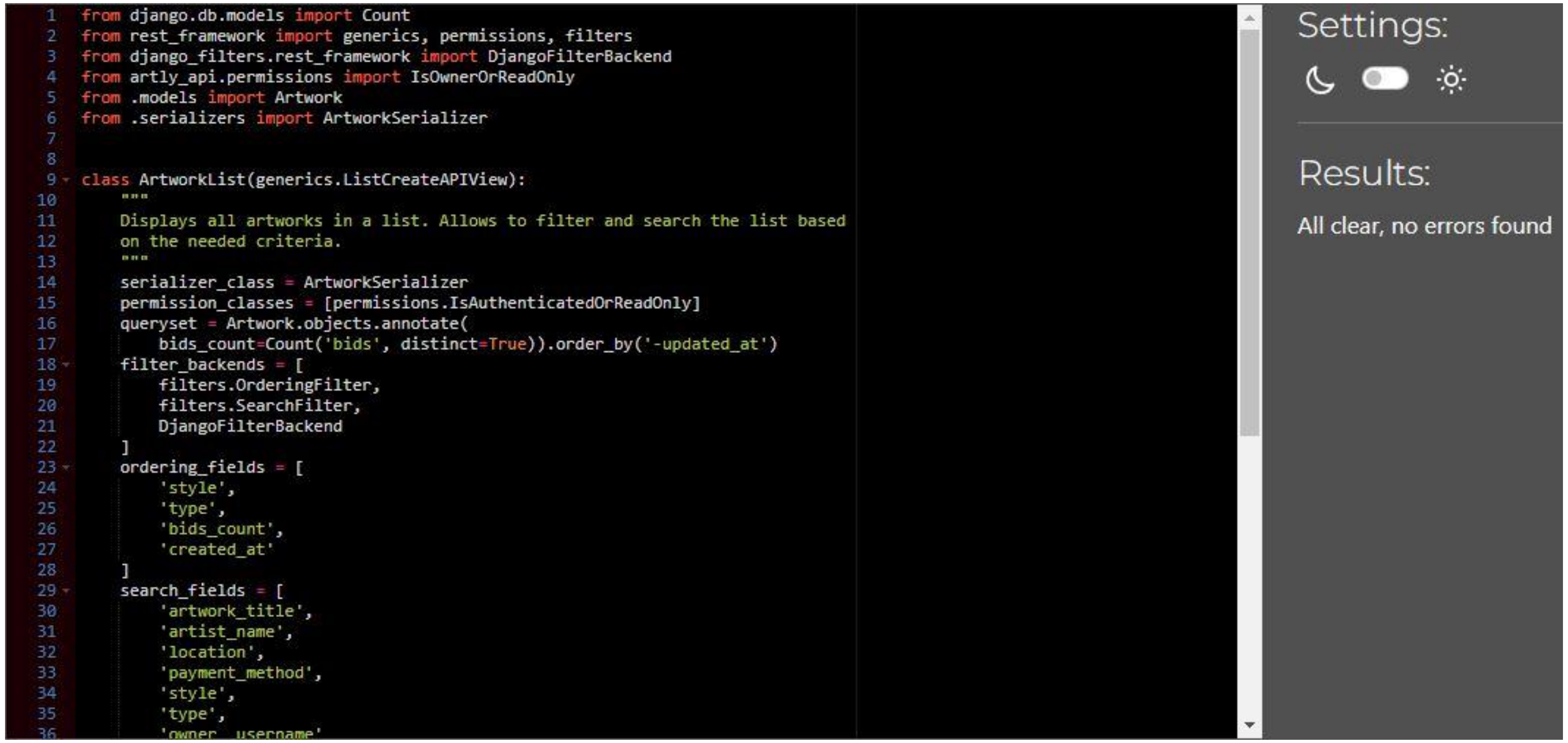
All clear, no errors found



```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()),
]
```

Results:

All clear, no errors found

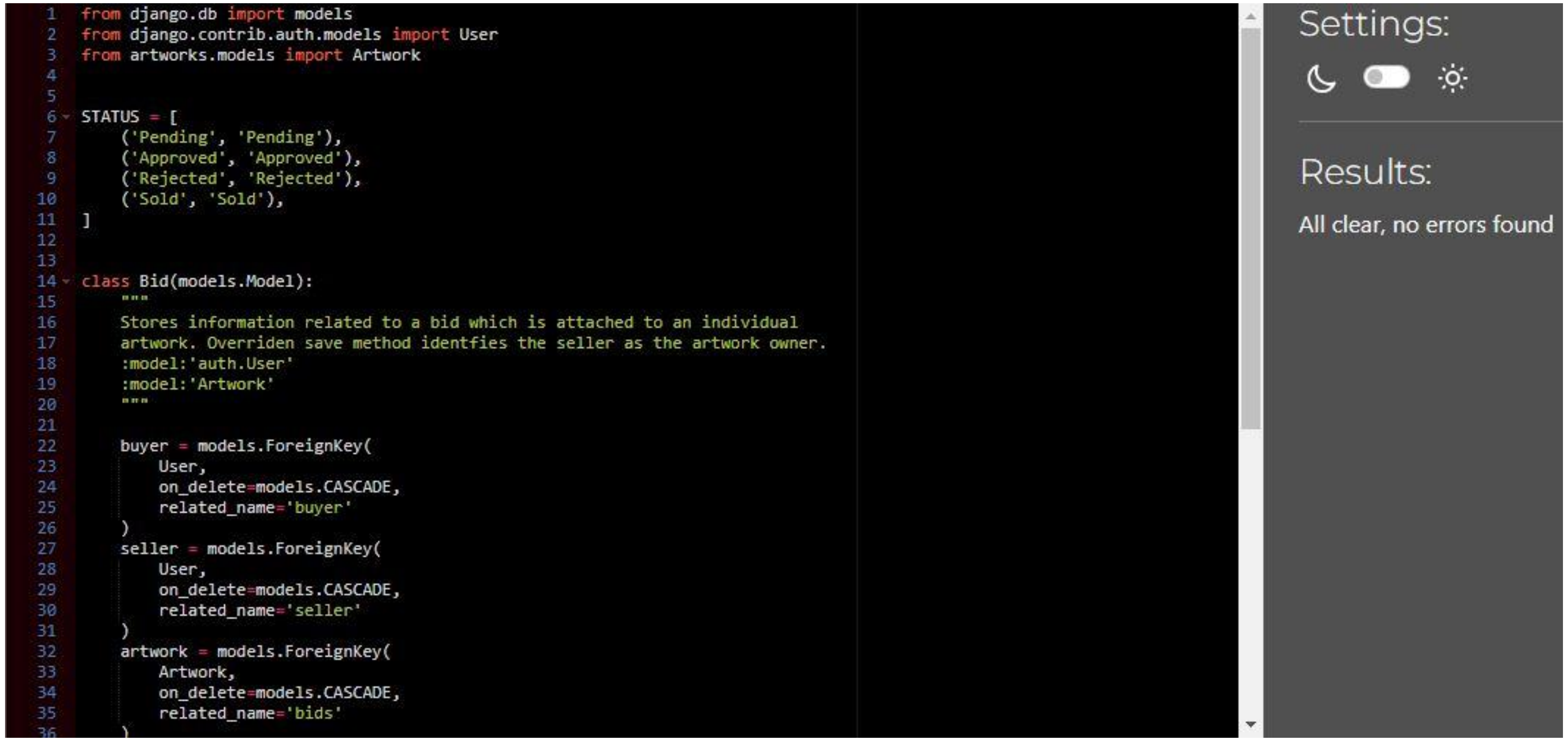


```python
from django.db.models import Count
from rest_framework import generics, permissions, filters
from django_filters.rest_framework import DjangoFilterBackend
from artly_api.permissions import IsOwnerOrReadOnly
from .models import Artwork
from .serializers import ArtworkSerializer


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'
```

Settings:

Results:

All clear, no errors found

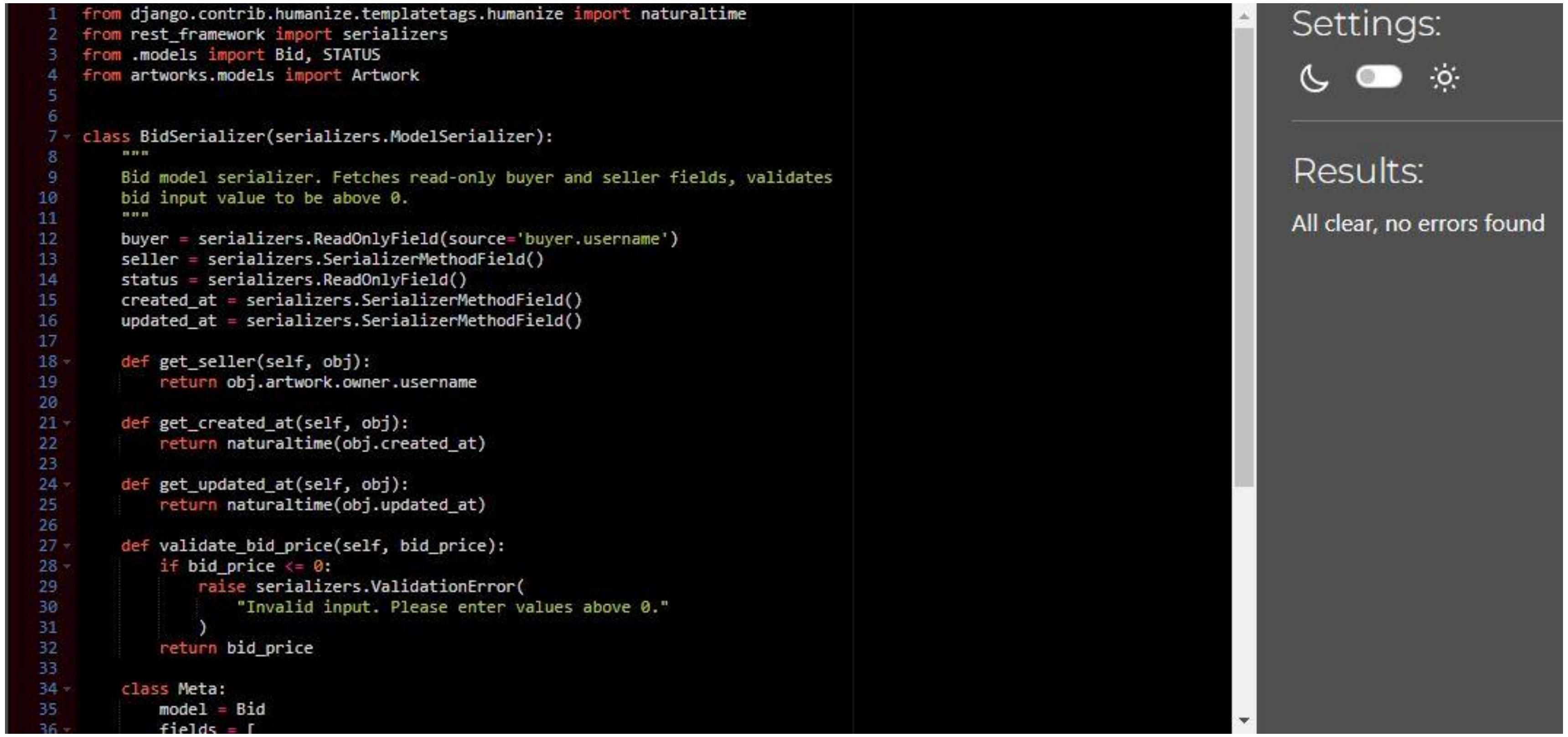


```python
from django.db import models
from django.contrib.auth.models import User
from artworks.models import Artwork


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

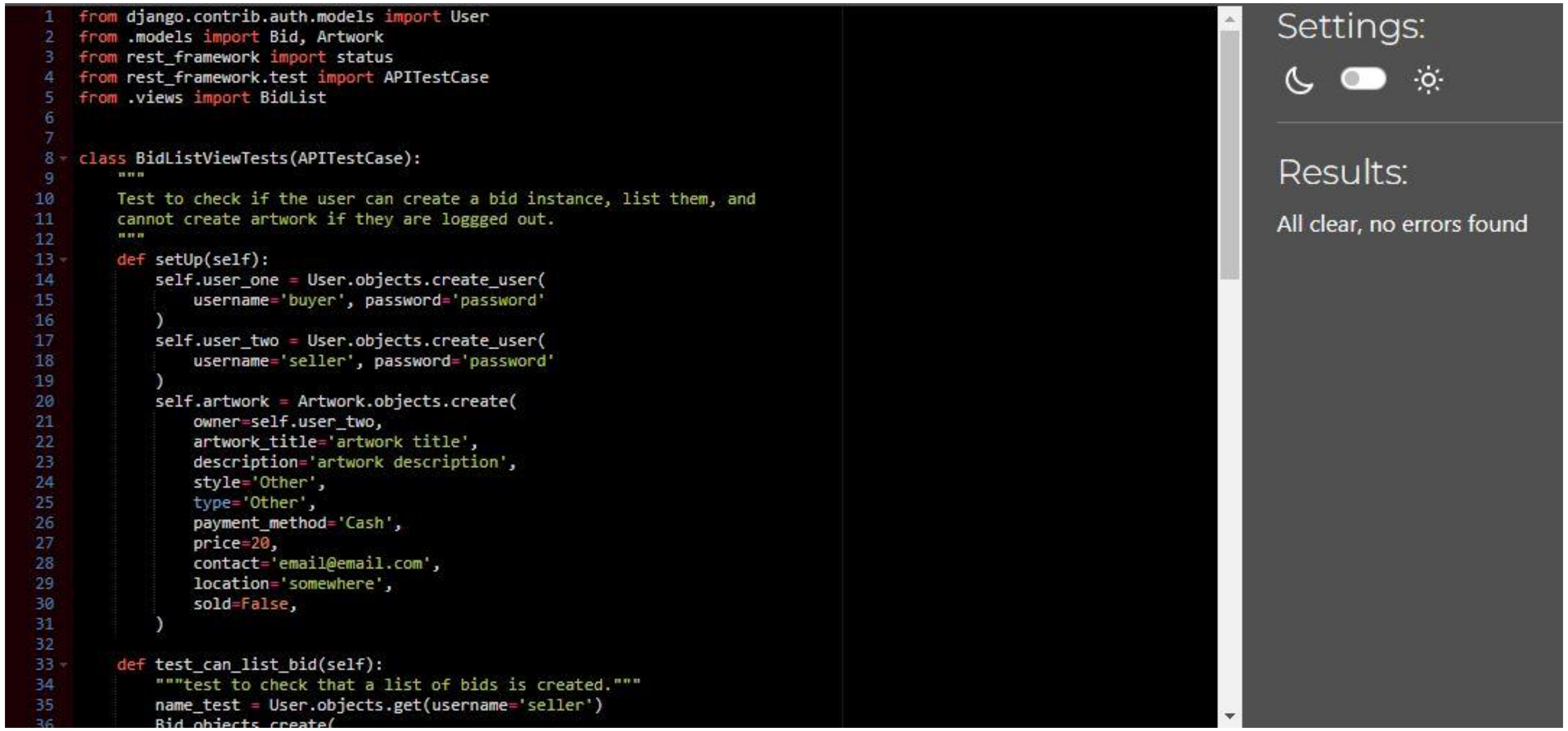


```
from django.contrib.humanize.templatetags.humanize import naturaltime
from rest_framework import serializers
from .models import Bid, STATUS
from artworks.models import Artwork


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

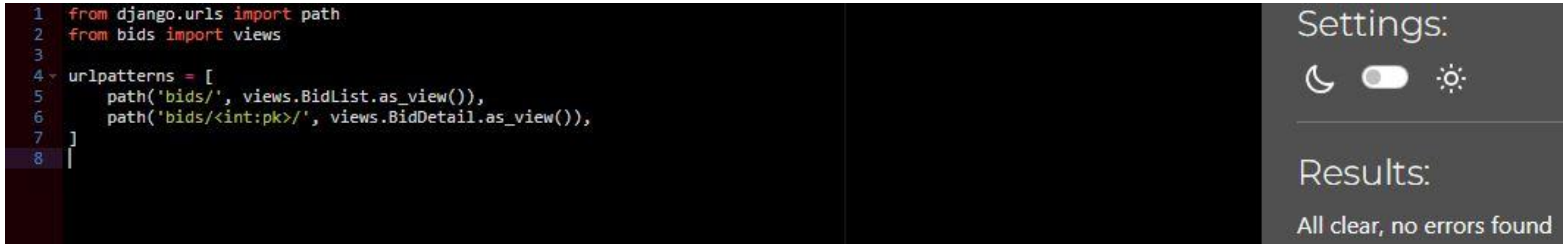


```
from django.contrib.auth.models import User
from .models import Bid, Artwork
from rest_framework import status
from rest_framework.test import APITestCase
from .views import BidList


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:

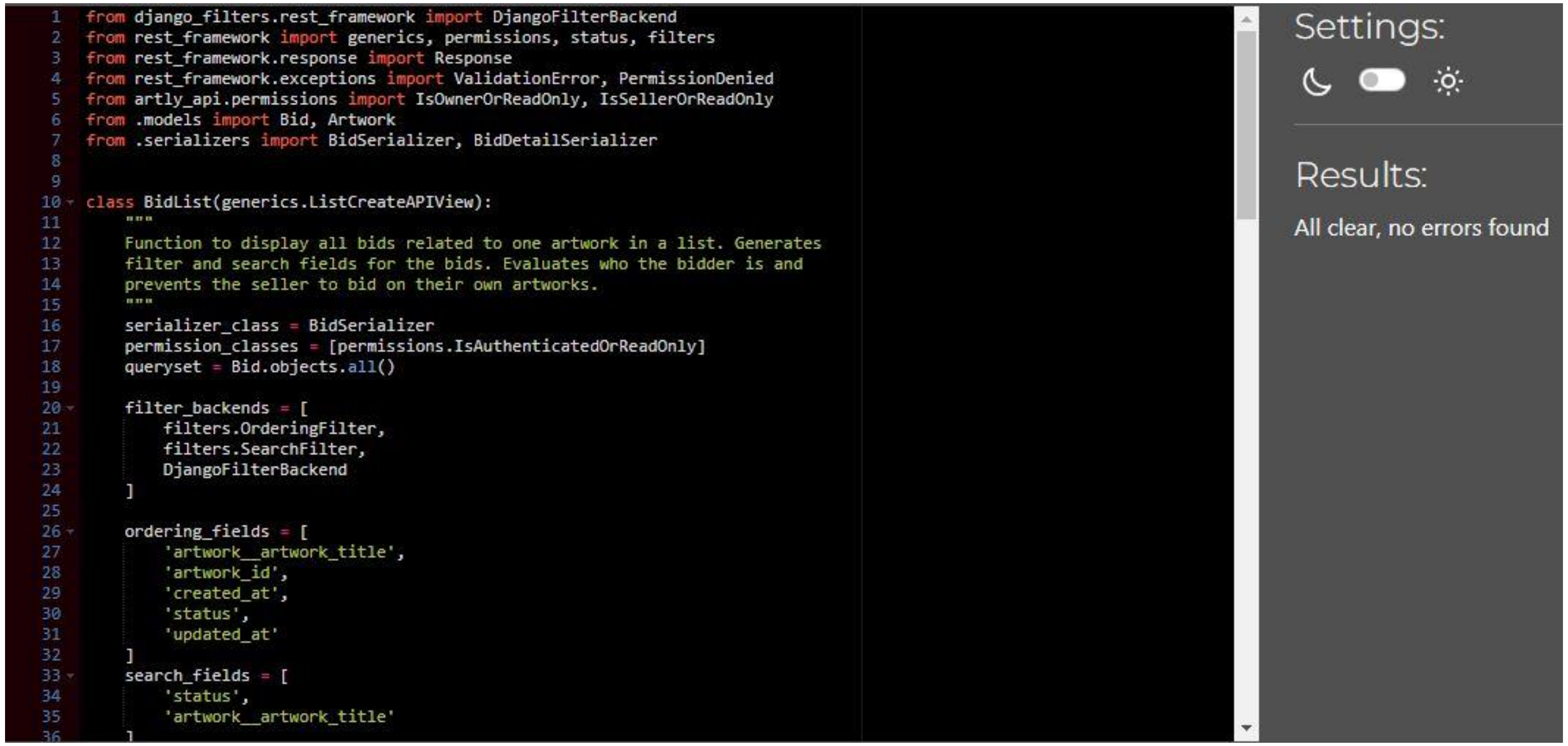
All clear, no errors found



```
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

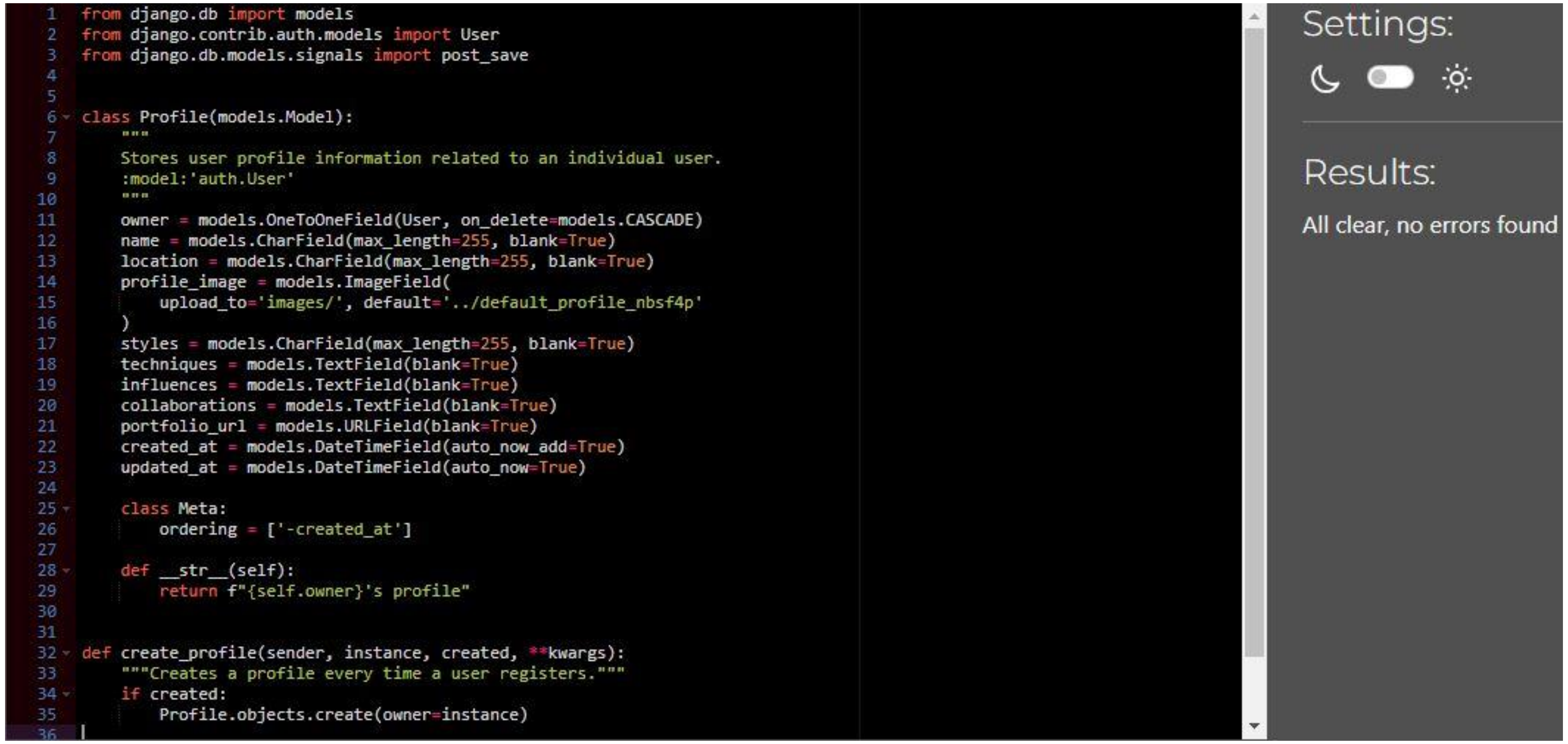


```python
from django_filters.rest_framework import DjangoFilterBackend
from rest_framework import generics, permissions, status, filters
from rest_framework.response import Response
from rest_framework.exceptions import ValidationError, PermissionDenied
from artly_api.permissions import IsOwnerOrReadOnly, 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:

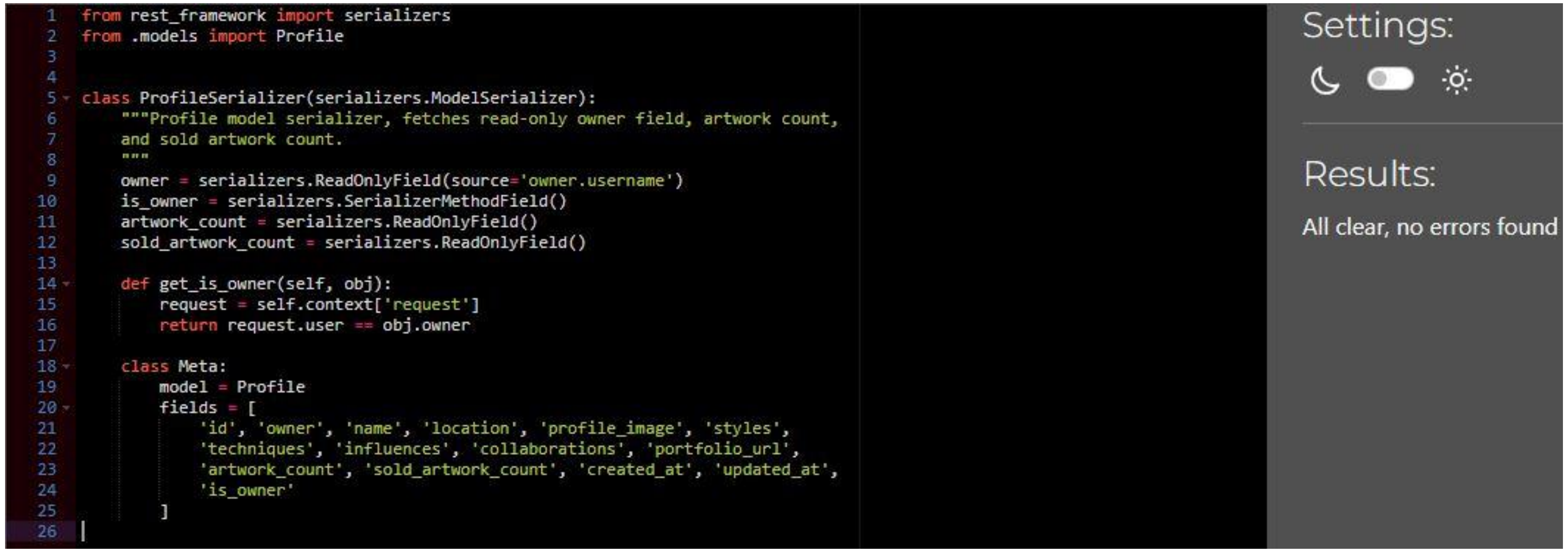
All clear, no errors found



```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)
```

Results:

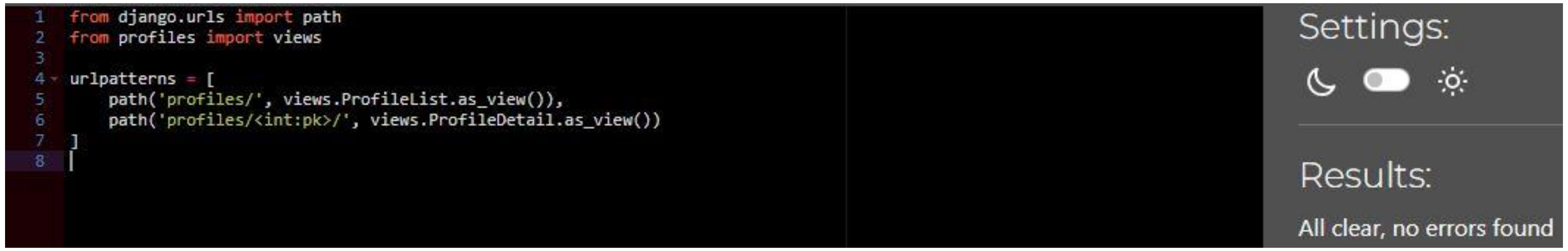
All clear, no errors found



```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'
        ]
```

Results:

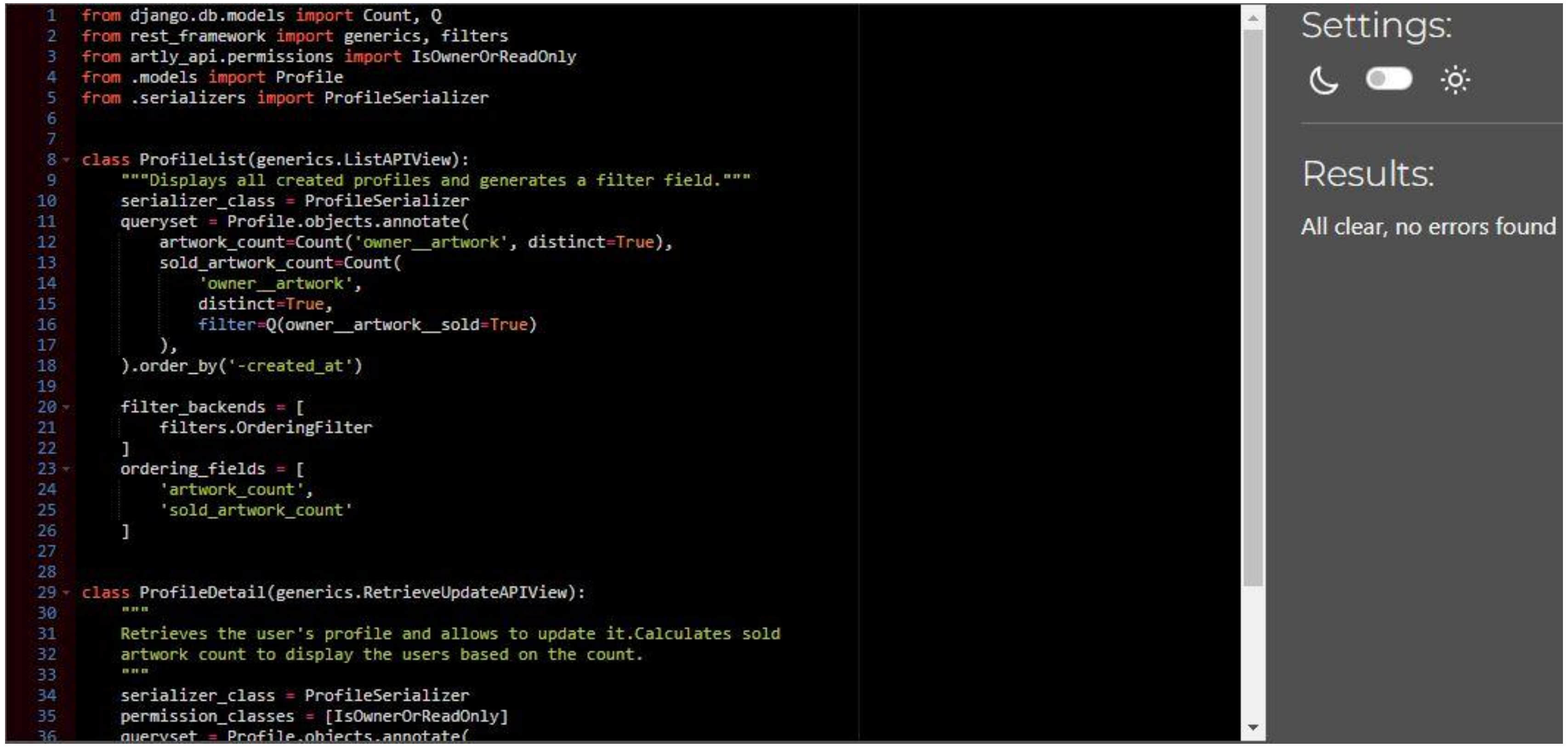
All clear, no errors found



```
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