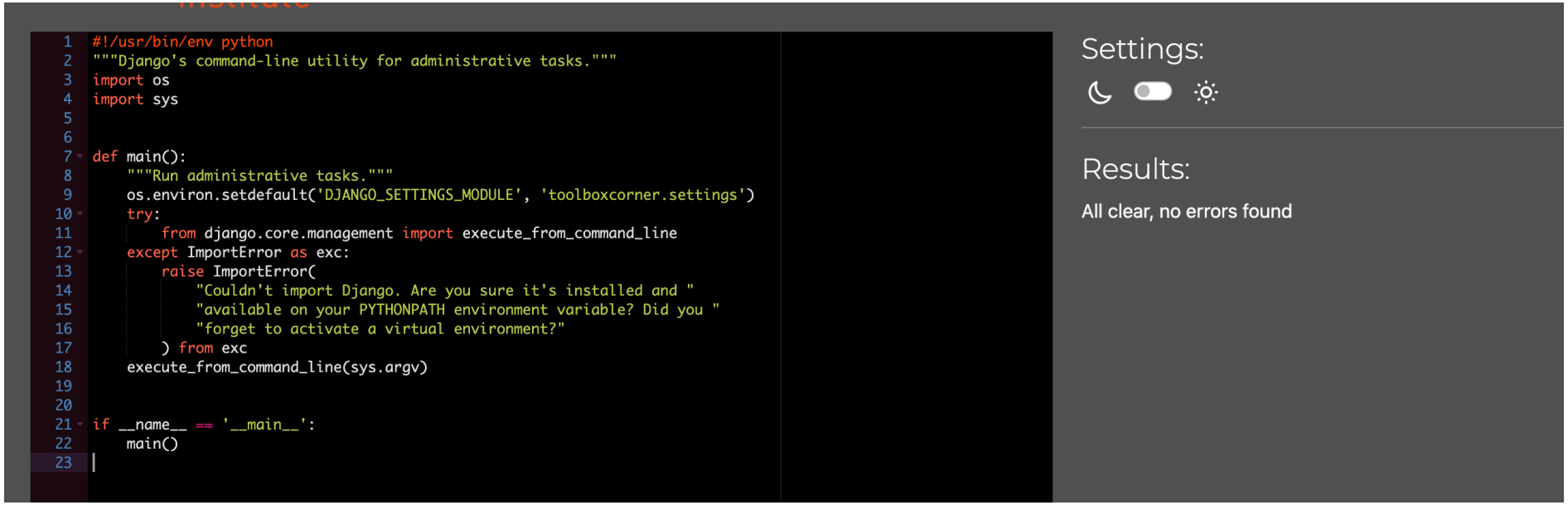
Manage.py


#!/usr/bin/env python
"""Django's command-line utility for administrative tasks."""
import os
import sys
def main():
"""Run administrative tasks."""
os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'toolboxcorner.settings')
try:
from django.core.management import execute_from_command_line
except ImportError as exc:
raise ImportError(
"Couldn't import Django. Are you sure it's installed and "
"available on your PYTHONPATH environment variable? Did you "
"forget to activate a virtual environment?"
) from exc
execute_from_command_line(sys.argv)
if __name__ == '__main__':
main()
Settings:
Results:
All clear, no errors found

Website – Views.py

```
            for form in image_formset:
                if form.cleaned_data.get('DELETE'):
                    if form.instance.pk:
                        form.instance.delete()
                elif form.cleaned_data.get('image'):
                    image_instance = form.save(commit=False)
                    image_instance.product = product
                    image_instance.save()

            messages.success(request, 'Product updated successfully!')
            return redirect(reverse('product_detail', args=[product.id]))
        else:
            messages.error(request, 'Please correct the errors below.')
    else:
        form = ProductForm(instance=product)
        image_formset = ProductImageFormSet(queryset=product.images.all())

    return render(
        request, 'website/edit_product.html',
        {'form': form, 'image_formset': image_formset, 'product': product}
    )


@staff_member_required
def delete_product(request, product_id):
    """
    Allows staff members to delete products.
    Handles both GET (display confirmation)
    and POST (process deletion) requests.
    """
    product = get_object_or_404(Product, id=product_id)
```

Settings:

Results:

All clear, no errors found

Website – urls.py

```python
from django.urls import path
from . import views
from django.urls import path, include
from .views import edit_product, search_results

urlpatterns = [
    path('', views.welcome, name='welcome'),
    path('products', views.home, name='home'),
    path('products/category/<int:category_id>/',
         views.category_products, name='category_products'),
    path('product/<int:product_id>/', views.product_detail,
         name='product_detail'),
    path('product/add/', views.add_product, name='add_product'),
    path('edit/<int:product_id>/', edit_product, name='edit_product'),
    path('delete/<int:product_id>/',
         views.delete_product, name='delete_product'),
    path('bag/', include('bag.urls')),
    path('search/', search_results, name='search_results'),
]
```

Website – Models.py

```python
class Product(models.Model):
    """Model representing products."""
    name = models.CharField(max_length=100)
    description = models.TextField(max_length=400)
    price = models.DecimalField(max_digits=10, decimal_places=2)
    has_sizes = models.BooleanField(default=False, null=True, blank=True)
    created_by = models.ForeignKey(User, null=True, blank=True,
                                   on_delete=models.SET_NULL)
    created_at = models.DateTimeField(auto_now_add=True)
    modified_at = models.DateTimeField(auto_now=True)
    category = models.ForeignKey(Category,
                                 null=True, blank=True,
                                 on_delete=models.SET_NULL,
                                 related_name='products')

    def __str__(self):
        """String representation of the product."""
        return self.name


class ProductImage(models.Model):
    """Model representing product images."""
    product = models.ForeignKey(Product, on_delete=models.CASCADE,
                                related_name='images')
    image = CloudinaryField('image', default='placeholder')
    alt = models.CharField(max_length=50, blank=True, null=True)

    def __str__(self):
        """String representation of the product image."""
        return f"Image for {self.product.name}"

```

Settings:

Results:

All clear, no errors found

Website – forms.py

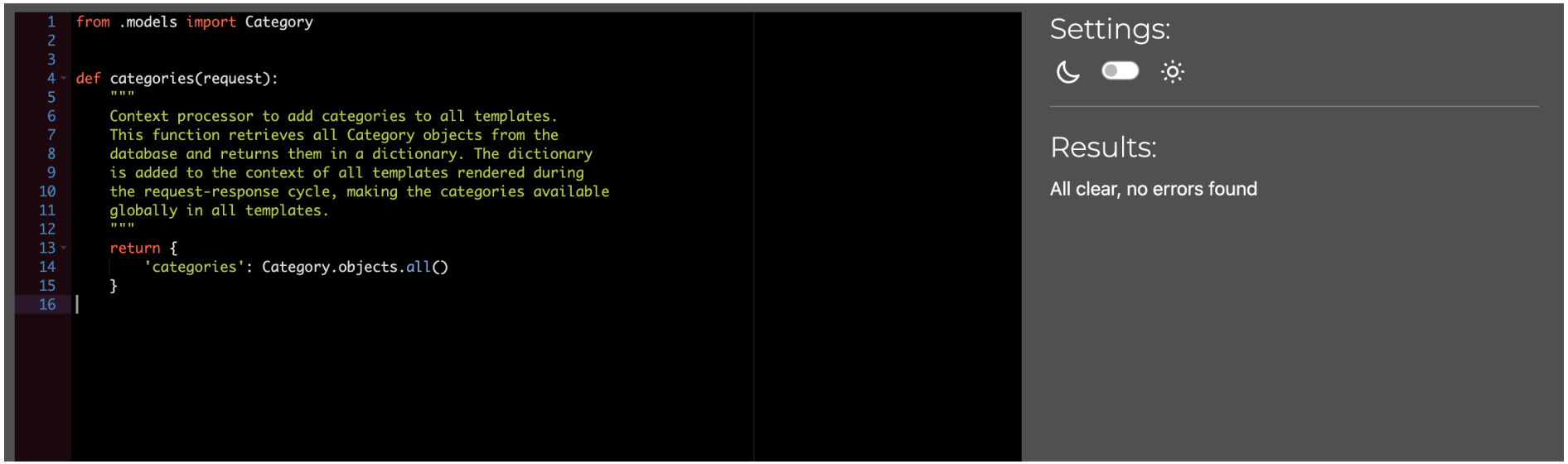
```python
from django import forms
from .models import Product, ProductImage, Category
from django.forms import modelformset_factory


class ProductForm(forms.ModelForm):
    """
    Form for creating and updating Product instances.
    """
    class Meta:
        model = Product
        fields = ['name', 'description', 'price', 'has_sizes', 'category']


class ProductImageForm(forms.ModelForm):
    """
    Form for creating and updating ProductImage instances.
    """
    class Meta:
        model = ProductImage
        fields = ['image', 'alt']

# Define the formset for ProductImage


ProductImageFormSet = modelformset_factory(ProductImage,
                                           form=ProductImageForm,
                                           extra=1, can_delete=True)

```

Settings:

Results:

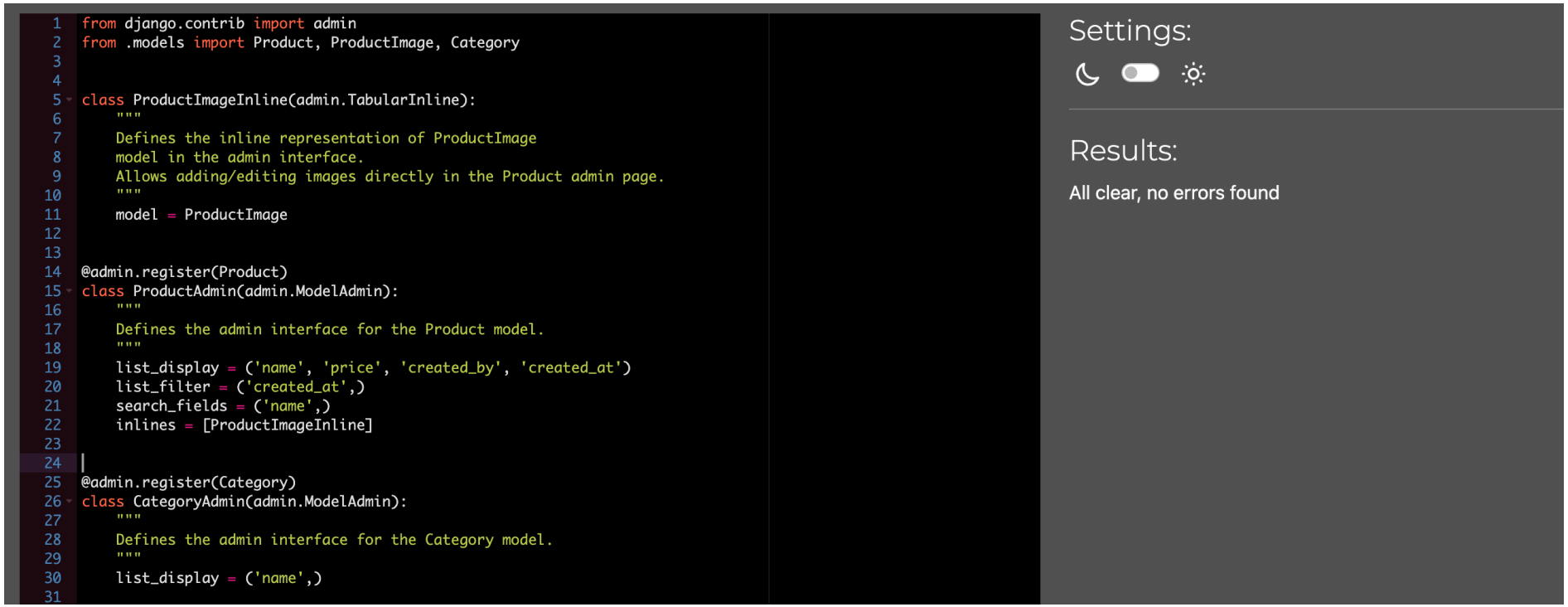
All clear, no errors found

Website – contexts.py



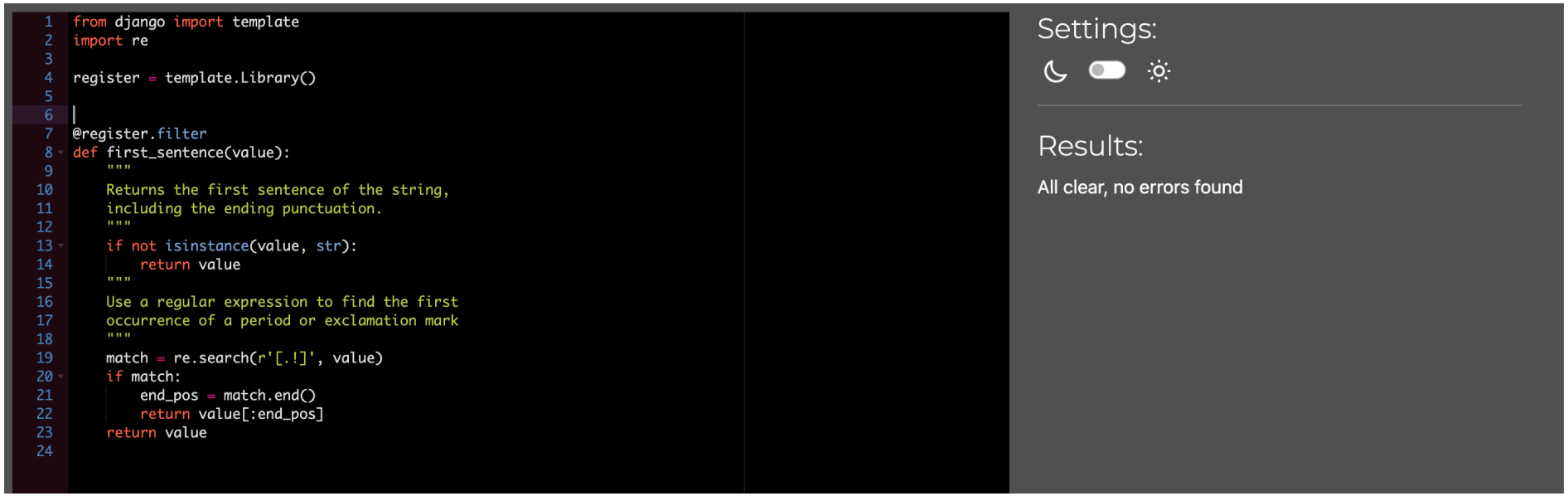
Website – apps.py

Website – admin.py

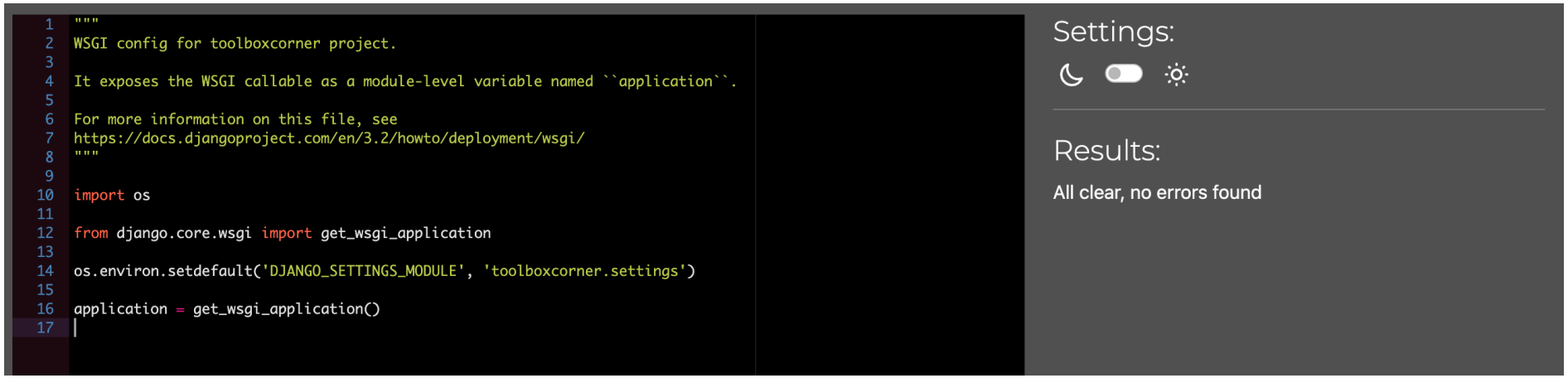


```python
from django.contrib import admin
from .models import Product, ProductImage, Category


class ProductImageInline(admin.TabularInline):
    """
    Defines the inline representation of ProductImage
    model in the admin interface.
    Allows adding/editing images directly in the Product admin page.
    """
    model = ProductImage


@admin.register(Product)
class ProductAdmin(admin.ModelAdmin):
    """
    Defines the admin interface for the Product model.
    """
    list_display = ('name', 'price', 'created_by', 'created_at')
    list_filter = ('created_at',)
    search_fields = ('name',)
    inlines = [ProductImageInline]

@admin.register(Category)
class CategoryAdmin(admin.ModelAdmin):
    """
    Defines the admin interface for the Category model.
    """
    list_display = ('name',)

```

Website – templatetags – custom_filters.py



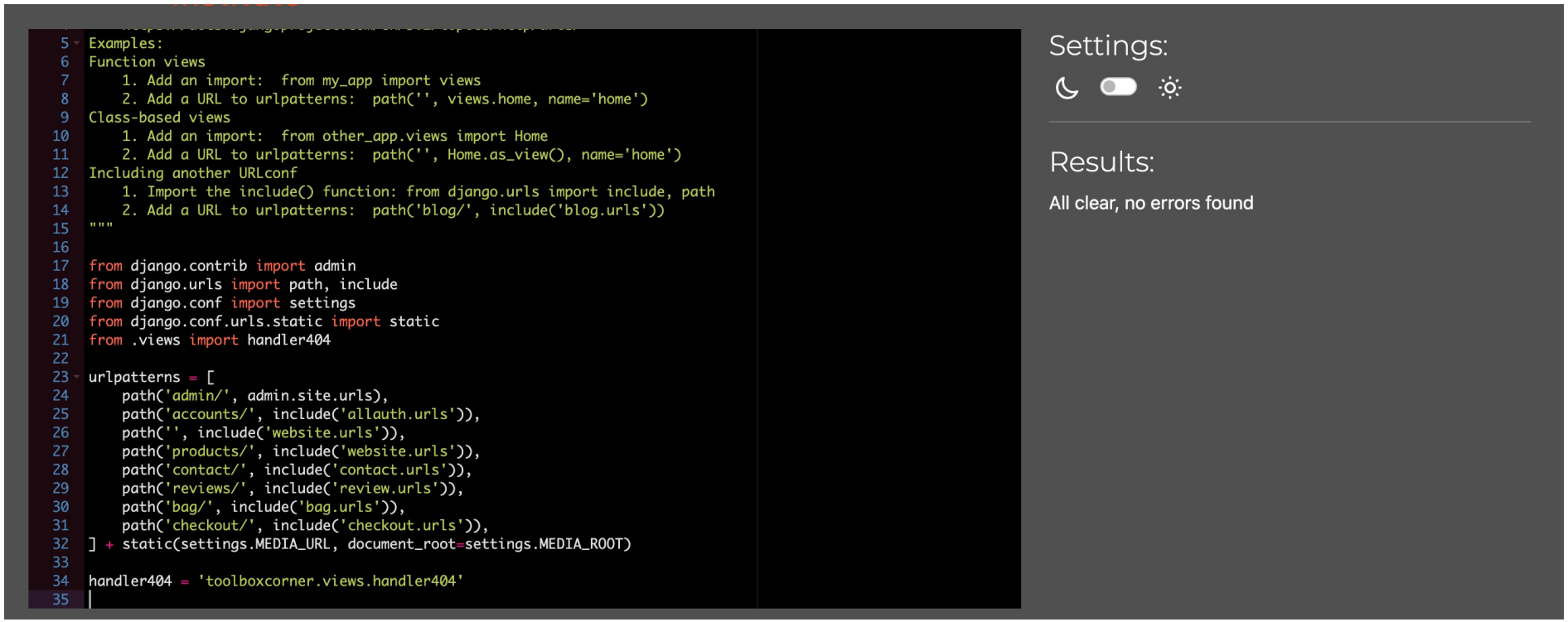
Tooboxcorner – wsgi.py



```python
"""
WSGI config for toolboxcorner project.

It exposes the WSGI callable as a module-level variable named ``application``.

For more information on this file, see
https://docs.djangoproject.com/en/3.2/howto/deployment/wsgi/
"""

import os

from django.core.wsgi import get_wsgi_application

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'toolboxcorner.settings')

application = get_wsgi_application()
```

Toolboxcorner – views.py

Toolboxcorner – urls.py



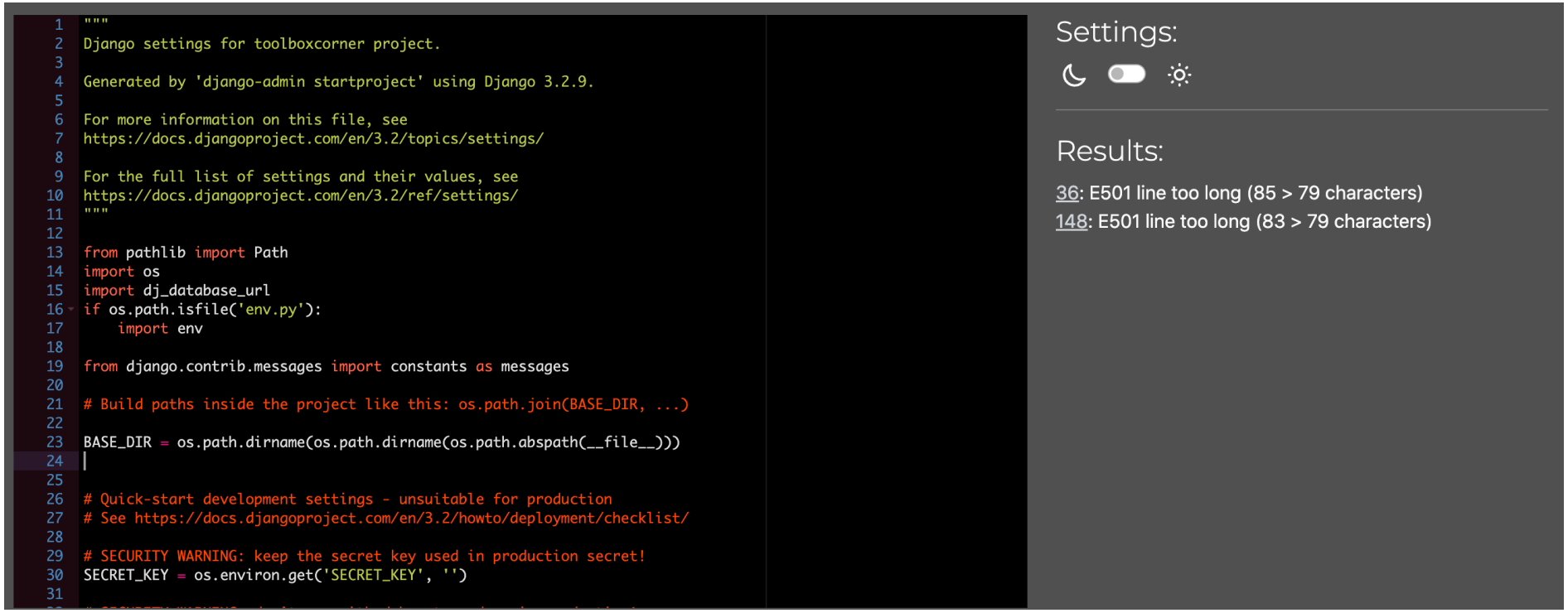
```
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including 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 django.conf import settings
from django.conf.urls.static import static
from .views import handler404

urlpatterns = [
    path('admin/', admin.site.urls),
    path('accounts/', include('allauth.urls')),
    path('', include('website.urls')),
    path('products/', include('website.urls')),
    path('contact/', include('contact.urls')),
    path('reviews/', include('review.urls')),
    path('bag/', include('bag.urls')),
    path('checkout/', include('checkout.urls')),
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)

handler404 = 'toolboxcorner.views.handler404'
```

Settings:

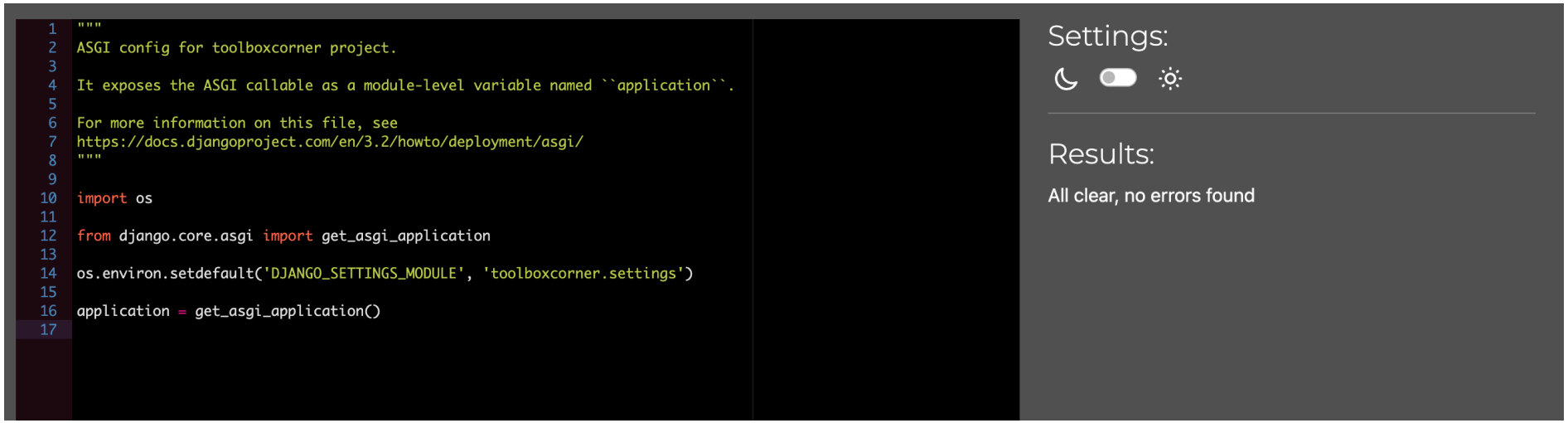
Results:

All clear, no errors found

Toolboxcorner – settings.py



Toolboxcorner – asgi.py



```
"""
ASGI config for toolboxcorner project.

It exposes the ASGI callable as a module-level variable named ``application``.

For more information on this file, see
https://docs.djangoproject.com/en/3.2/howto/deployment/asgi/
"""

import os

from django.core.asgi import get_asgi_application

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'toolboxcorner.settings')

application = get_asgi_application()
```

Review - views.py

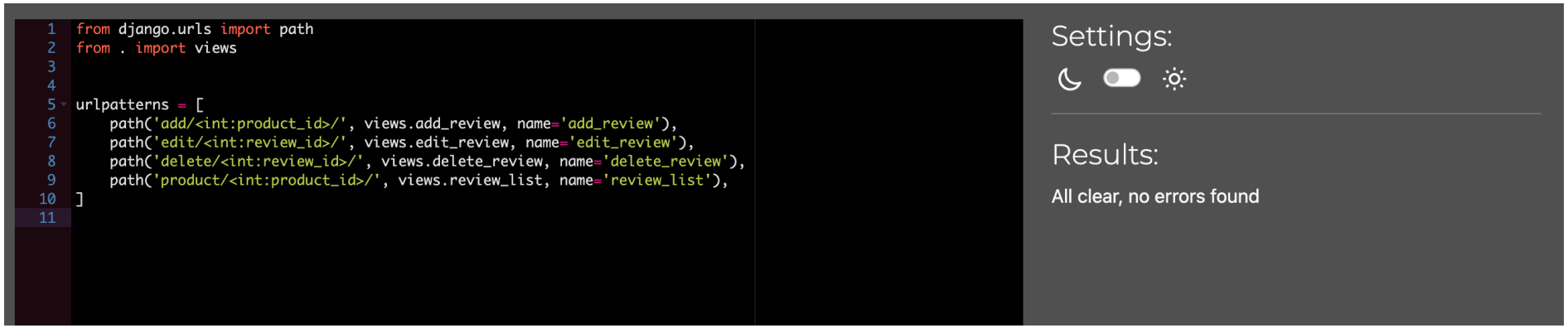
```python
from django.shortcuts import render, get_object_or_404, redirect
from django.contrib.auth.decorators import login_required
from django.http import HttpResponseForbidden
from .models import Review
from .forms import ReviewForm
from website.models import Product
from django.db.models import Avg


@login_required
def add_review(request, product_id):
    """
    Handles the submission of a new review for a given product.
    """
    product = get_object_or_404(Product, id=product_id)
    if request.method == 'POST':
        form = ReviewForm(request.POST)
        if form.is_valid():
            review = form.save(commit=False)
            review.product = product
            review.user = request.user
            review.save()
            return redirect('product_detail', product.id)
    else:
        form = ReviewForm()
    return render(request,
                  'review/add_review.html',
                  {'form': form, 'product': product})


@login_required
```

Settings:

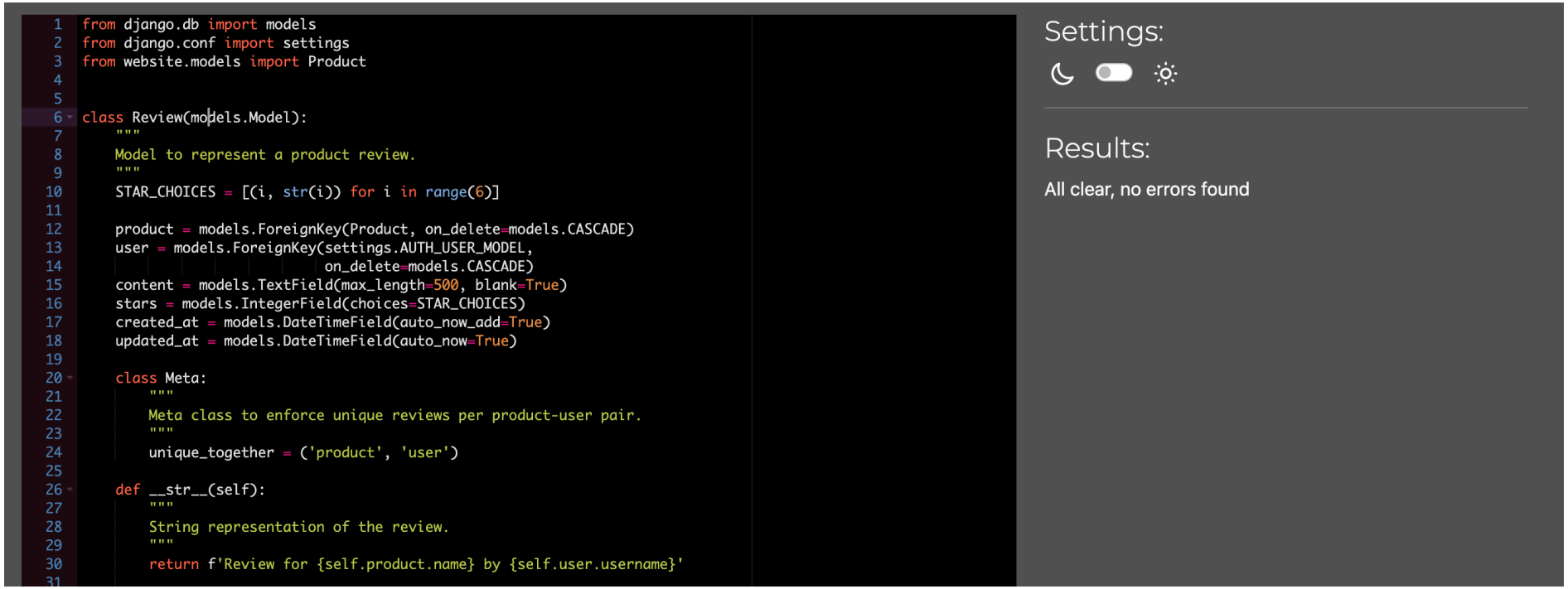
Results:

All clear, no errors found

Review – urls.py

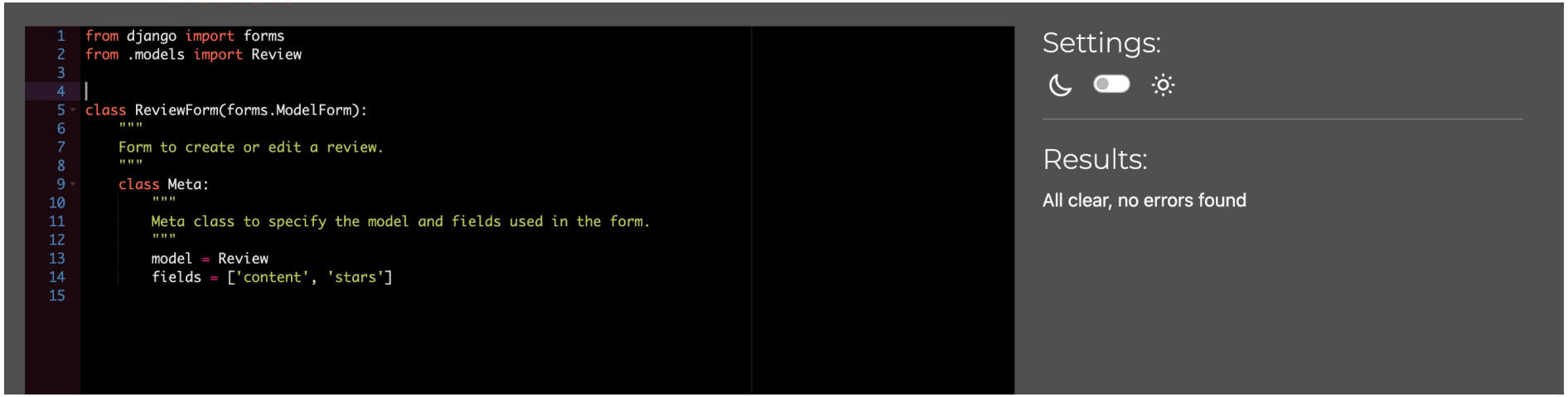


Review – models.py



```python
from django.db import models
from django.conf import settings
from website.models import Product


class Review(models.Model):
    """
    Model to represent a product review.
    """
    STAR_CHOICES = [(i, str(i)) for i in range(6)]

    product = models.ForeignKey(Product, on_delete=models.CASCADE)
    user = models.ForeignKey(settings.AUTH_USER_MODEL,
                             on_delete=models.CASCADE)
    content = models.TextField(max_length=500, blank=True)
    stars = models.IntegerField(choices=STAR_CHOICES)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    class Meta:
        """
        Meta class to enforce unique reviews per product-user pair.
        """
        unique_together = ('product', 'user')

    def __str__(self):
        """
        String representation of the review.
        """
        return f'Review for {self.product.name} by {self.user.username}'
```

Review – forms.py



Review – apps.py

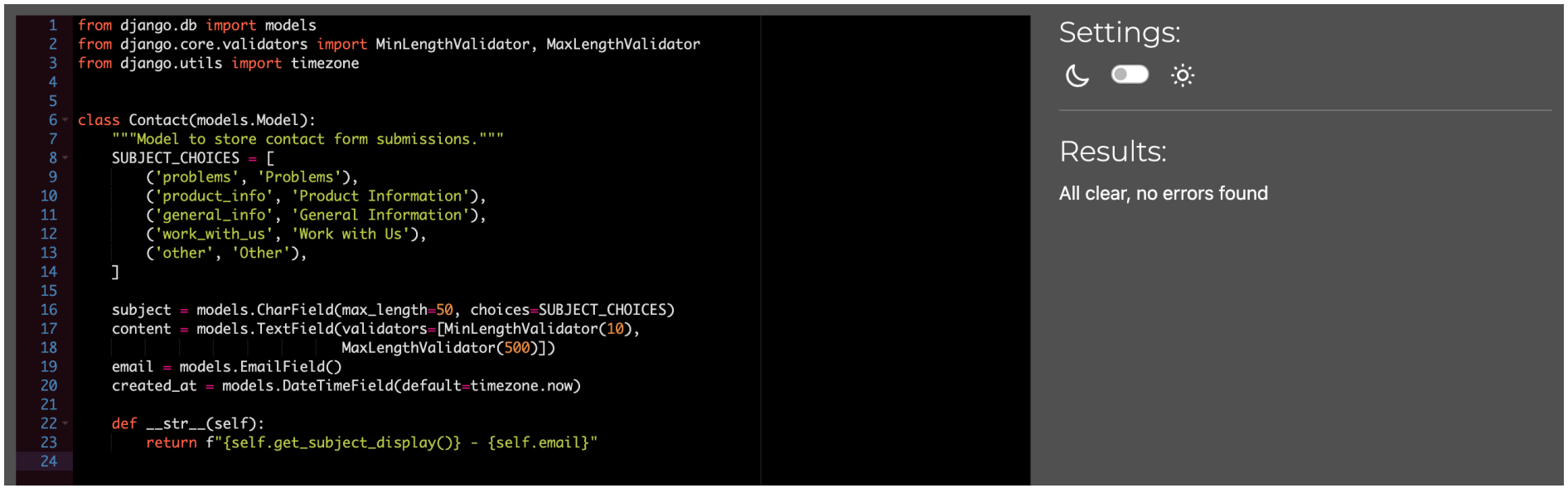
Review - admin.py

```python
from django.contrib import admin
from .models import Review


@admin.register(Review)
class ReviewAdmin(admin.ModelAdmin):
    """Admin configuration for the Review model."""

    list_display = ('product', 'user', 'stars', 'created_at')
    list_filter = ('stars', 'created_at')
    search_fields = ('content',)

```

Contact – views.py

Contact – urls.py

Contact – models.py



```python
from django.db import models
from django.core.validators import MinLengthValidator, MaxLengthValidator
from django.utils import timezone


class Contact(models.Model):
    """Model to store contact form submissions."""
    SUBJECT_CHOICES = [
        ('problems', 'Problems'),
        ('product_info', 'Product Information'),
        ('general_info', 'General Information'),
        ('work_with_us', 'Work with Us'),
        ('other', 'Other'),
    ]

    subject = models.CharField(max_length=50, choices=SUBJECT_CHOICES)
    content = models.TextField(validators=[MinLengthValidator(10),
                                           MaxLengthValidator(500)])
    email = models.EmailField()
    created_at = models.DateTimeField(default=timezone.now)

    def __str__(self):
        return f"{self.get_subject_display()} - {self.email}"

```

Settings:

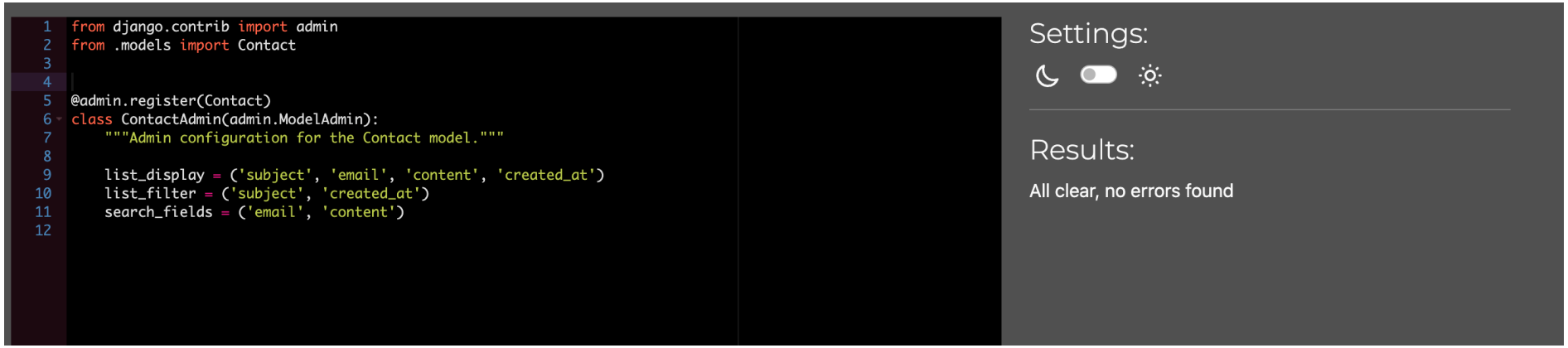
Results:

All clear, no errors found

Contact – forms.py

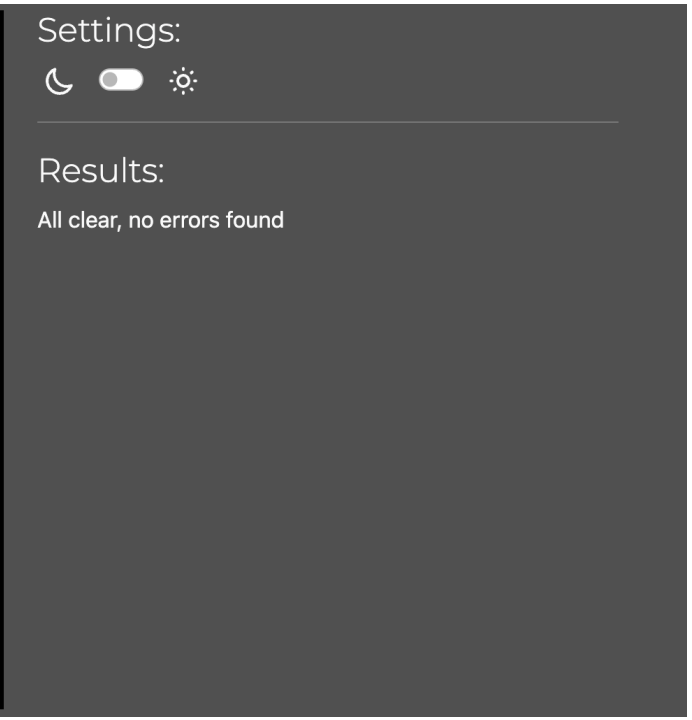
Contact – apps.py

Contact – admin.py



Checkout – webhooks.py

```python
from django.conf import settings
from django.http import HttpResponse
from django.views.decorators.http import require_POST
from django.views.decorators.csrf import csrf_exempt

from checkout.webhook_handler import StripeWH_Handler

import stripe


@require_POST
@csrf_exempt
def webhook(request):
    """Listen for webhooks from Stripe"""
    # Setup
    wh_secret = settings.STRIPE_WH_SECRET
    stripe.api_key = settings.STRIPE_SECRET_KEY

    # Get the webhook data and verify its signature
    payload = request.body
    sig_header = request.META['HTTP_STRIPE_SIGNATURE']
    event = None

    try:
        event = stripe.Webhook.construct_event(
        payload, sig_header, wh_secret
        )
    except ValueError as e:
        # Invalid payload
        return HttpResponse(status=400)
    except stripe.error.SignatureVerificationError as e:
```



Checkout – webhook_handler.py

```python
from django.http import HttpResponse

from .models import Order, OrderLineItem
from website.models import Product

import json
import time
import stripe


class StripeWH_Handler:
    """Handle Stripe webhooks"""

    def __init__(self, request):
        self.request = request

    def handle_event(self, event):
        """
        Handle a generic/unknown/unexpected webhook event
        """
        return HttpResponse(
            content=f'Unhandled webhook received: {event["type"]}',
            status=200
        )

    def handle_payment_intent_succeeded(self, event):
        """
        Handle the payment_intent.succeeded webhook from Stripe
        """
        intent = event.data.object
        pid = intent.id
```

Settings:

Results:

106: E501 line too long (81 > 79 characters)

Checkout – views.py

```python
from django.shortcuts import render, redirect, reverse
from django.shortcuts import get_object_or_404, HttpResponse
from django.views.decorators.http import require_POST
from django.contrib import messages
from django.conf import settings
import logging

from .forms import OrderForm
from .models import Order, OrderLineItem
from website.models import Product
from bag.contexts import bag_contents

import stripe
import json


@require_POST
def cache_checkout_data(request):
    """
    Cache checkout data to associate with Stripe PaymentIntent.
    """
    try:
        pid = request.POST.get('client_secret').split('_secret')[0]
        stripe.api_key = settings.STRIPE_SECRET_KEY
        stripe.PaymentIntent.modify(pid, metadata={
            'bag': json.dumps(request.session.get('bag', {})),
            'save_info': request.POST.get('save_info'),
            'username': request.user,
        })
        return HttpResponse(status=200)
    except Exception as e:
```

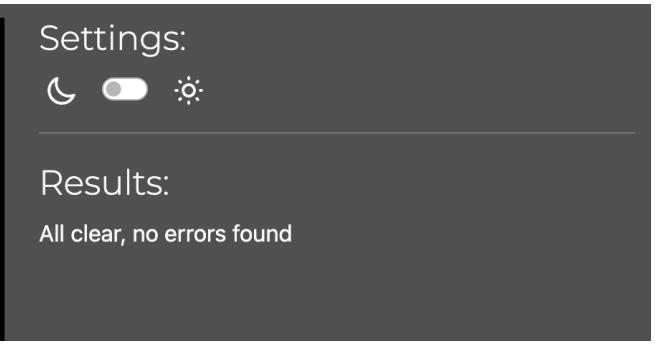
Settings:

Results:

78: E501 line too long (81 > 79 characters)

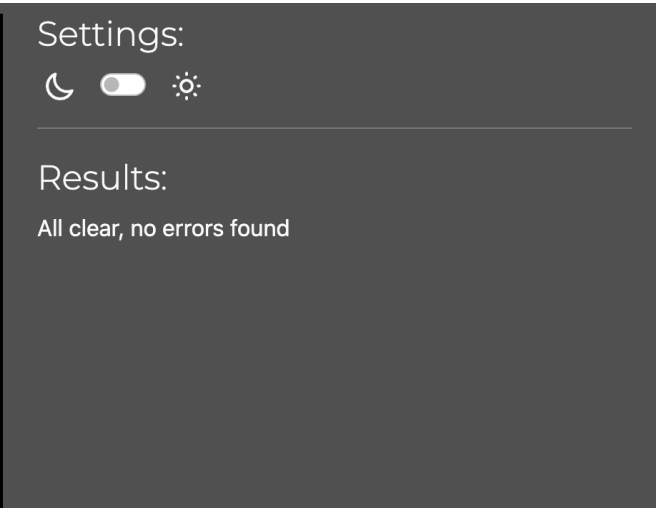
Checkout - urls.py

```python
from django.urls import path
from . import views
from .webhooks import webhook

urlpatterns = [
    path('', views.checkout, name='checkout'),
    path('checkout_success/<order_number>', views.checkout_success,
         name='checkout_success'),
    path('cache_checkout_data/', views.cache_checkout_data,
         name='cache_checkout_data'),
    path('wh/', webhook, name='webhook'),
]
```

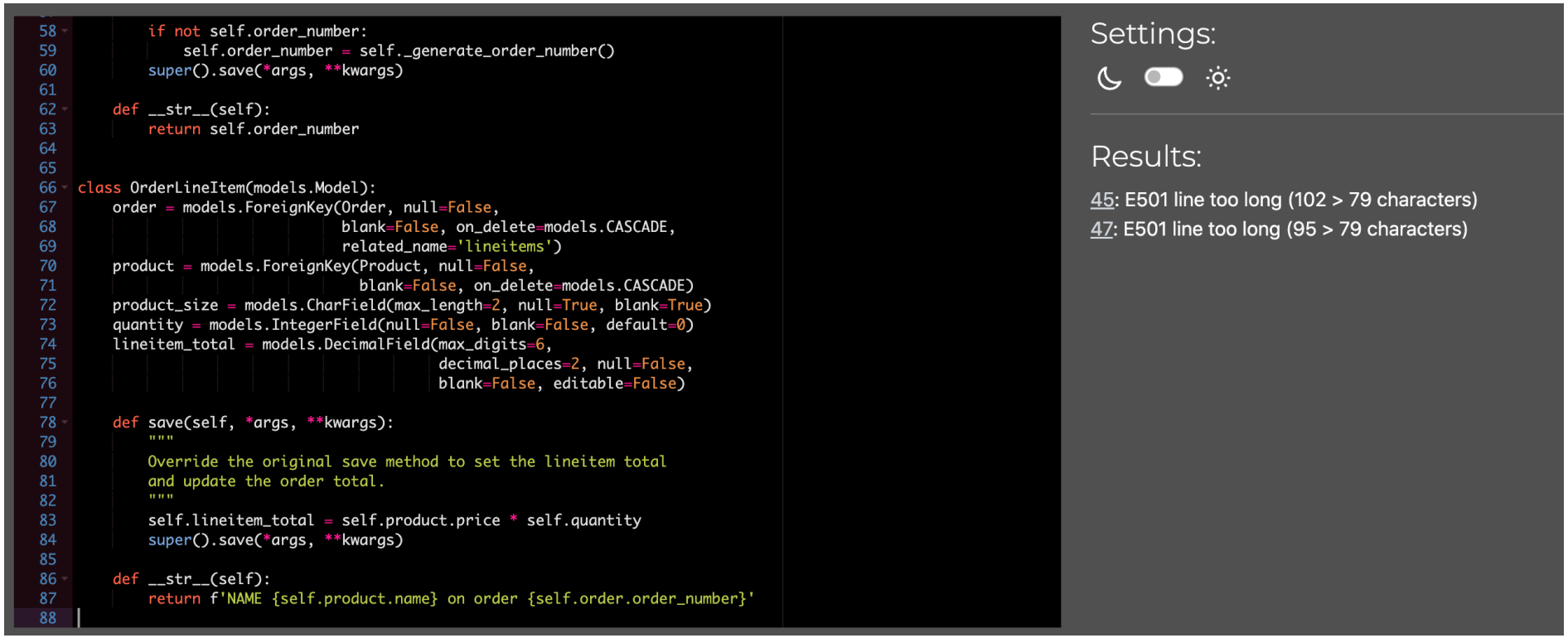


Checkout – signals.py

```python
from django.db.models.signals import post_save, post_delete
from django.dispatch import receiver

from .models import OrderLineItem


@receiver(post_save, sender=OrderLineItem)
def update_on_save(sender, instance, created, **kwargs):
    """
    Update order total on lineitem update/create
    """
    instance.order.update_total()


@receiver(post_delete, sender=OrderLineItem)
def update_on_delete(sender, instance, **kwargs):
    """
    Update order total on lineitem delete
    """
    instance.order.update_total()
```



Checkout – models.py



```
        if not self.order_number:
            self.order_number = self._generate_order_number()
        super().save(*args, **kwargs)

    def __str__(self):
        return self.order_number


class OrderLineItem(models.Model):
    order = models.ForeignKey(Order, null=False,
                              blank=False, on_delete=models.CASCADE,
                              related_name='lineitems')
    product = models.ForeignKey(Product, null=False,
                                blank=False, on_delete=models.CASCADE)
    product_size = models.CharField(max_length=2, null=True, blank=True)
    quantity = models.IntegerField(null=False, blank=False, default=0)
    lineitem_total = models.DecimalField(max_digits=6,
                                         decimal_places=2, null=False,
                                         blank=False, editable=False)

    def save(self, *args, **kwargs):
        """
        Override the original save method to set the lineitem total
        and update the order total.
        """
        self.lineitem_total = self.product.price * self.quantity
        super().save(*args, **kwargs)

    def __str__(self):
        return f'NAME {self.product.name} on order {self.order.order_number}'
```

Settings:

Results:

45: E501 line too long (102 > 79 characters)
47: E501 line too long (95 > 79 characters)

Checkout – forms.py

Checkout – apps.py

Checkout – admin.py

```python

class OrderLineItemAdminInline(admin.TabularInline):
    model = OrderLineItem
    readonly_fields = ('lineitem_total',)


class OrderAdmin(admin.ModelAdmin):
    inlines = (OrderLineItemAdminInline,)

    readonly_fields = ('order_number', 'date',
                       'delivery_cost', 'order_total',
                       'grand_total', 'original_bag',
                       'stripe_pid')

    fields = ('order_number', 'date', 'full_name',
              'email', 'phone_number', 'country',
              'postcode', 'town_or_city', 'street_address1',
              'street_address2', 'county', 'delivery_cost',
              'order_total', 'grand_total', 'original_bag',
              'stripe_pid')

    list_display = ('order_number', 'date', 'full_name',
                    'order_total', 'delivery_cost',
                    'grand_total',)

    ordering = ('-date',)


admin.site.register(Order, OrderAdmin)

```

Settings:

Results:

All clear, no errors found

Bag – views.py

```python
from django.shortcuts import render, redirect, reverse
from django.shortcuts import HttpResponse, get_object_or_404
from django.contrib import messages
from website.models import Product


def view_bag(request):
    """ A view that renders the bag contents page """

    return render(request, 'bag/bag.html')


def add_to_bag(request, item_id):
    """ Add a quantity of the specified product to the shopping bag """
    product = get_object_or_404(Product, pk=item_id)
    quantity = int(request.POST.get('quantity'))
    redirect_url = request.POST.get('redirect_url')
    size = None
    if 'product_size' in request.POST:
        size = request.POST['product_size']
    bag = request.session.get('bag', {})

    if size:
        if item_id in list(bag.keys()):
            if size in bag[item_id]['items_by_size'].keys():
                bag[item_id]['items_by_size'][size] += quantity
            else:
                bag[item_id]['items_by_size'][size] = quantity
        else:
            bag[item_id] = {'items_by_size': {size: quantity}}
    else:
```

Settings:

Results:

56: E501 line too long (87 > 79 characters)
62: E501 line too long (87 > 79 characters)
93: E501 line too long (87 > 79 characters)

Bag – urls.py

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

urlpatterns = [
    path('', views.view_bag, name='view_bag'),
    path('add/<item_id>/', views.add_to_bag, name='add_to_bag'),
    path('adjust/<item_id>/', views.adjust_bag, name='adjust_bag'),
    path('remove/<item_id>/', views.remove_from_bag, name='remove_from_bag'),
]
```

Settings:

Results:

All clear, no errors found

Bag – contexts.py

```
            for size, quantity in item_data['items_by_size'].items():
                total += quantity * product.price
                product_count += quantity
                bag_items.append({
                    'item_id': item_id,
                    'quantity': quantity,
                    'product': product,
                    'size': size,
                })

    if total < settings.FREE_DELIVERY_THRESHOLD:
        delivery = total * Decimal(settings.STANDARD_DELIVERY_PERCENTAGE / 100)
        free_delivery_delta = settings.FREE_DELIVERY_THRESHOLD - total
    else:
        delivery = 0
        free_delivery_delta = 0

    grand_total = delivery + total

    context = {
        'bag_items': bag_items,
        'total': total,
        'product_count': product_count,
        'delivery': delivery,
        'free_delivery_delta': free_delivery_delta,
        'free_delivery_threshold': settings.FREE_DELIVERY_THRESHOLD,
        'grand_total': grand_total,
    }

    return context
```

Settings:

Results:

All clear, no errors found

Bag – apps.py


1 from django.apps import AppConfig
2
3
4 class BagConfig(AppConfig):
5 default_auto_field = 'django.db.models.BigAutoField'
6 name = 'bag'
7
Settings:
Results:
All clear, no errors found

Bag – templatetags – bag_tools.py