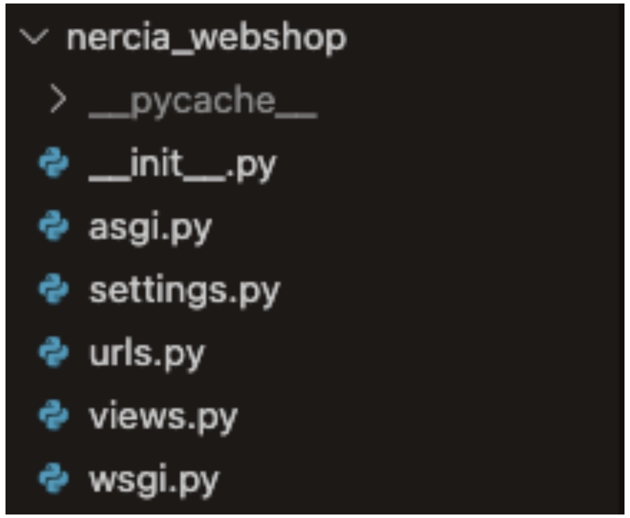
Python Validation

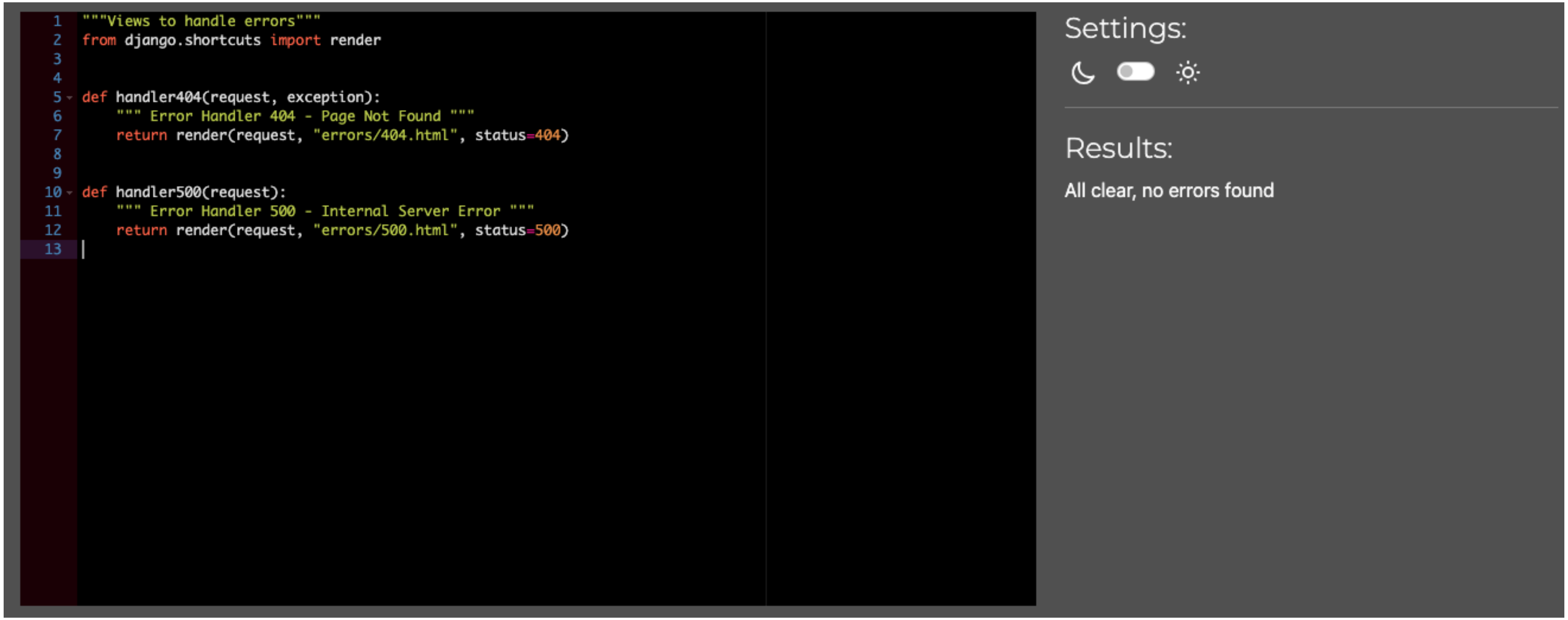
Project: nercia_webshop



```
nercia_webshop
  __pycache__
  __init__.py
  asgi.py
  settings.py
  urls.py
  views.py
  wsgi.py
```

Python Validation

Nercia_webshop/views.py



```
"""Views to handle errors"""
from django.shortcuts import render


def handler404(request, exception):
    """ Error Handler 404 - Page Not Found """
    return render(request, "errors/404.html", status=404)


def handler500(request):
    """ Error Handler 500 - Internal Server Error """
    return render(request, "errors/500.html", status=500)
```

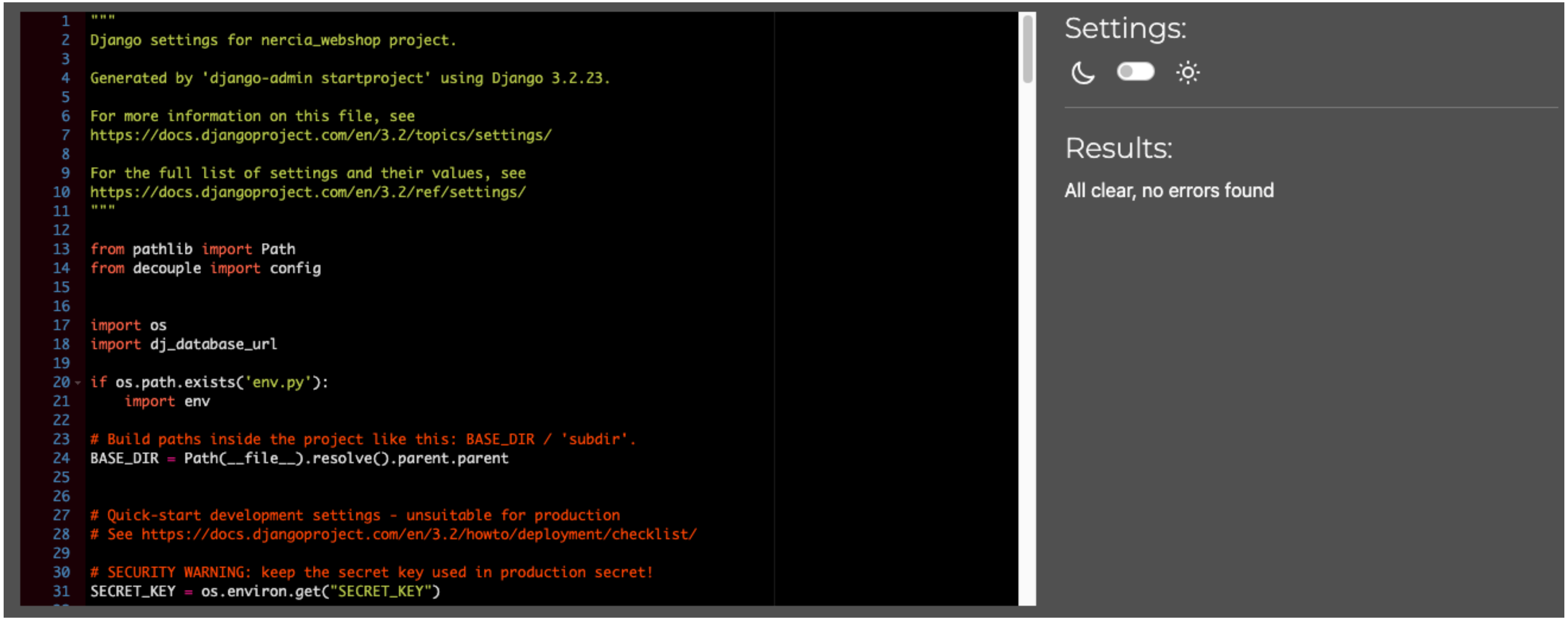
Settings:

Results:

All clear, no errors found

Python Validation

Nercia_webshop/settings.py



```
"""
Django settings for nercia_webshop project.

Generated by 'django-admin startproject' using Django 3.2.23.

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
from decouple import config


import os
import dj_database_url

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

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


# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/3.2/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = os.environ.get("SECRET_KEY")
```

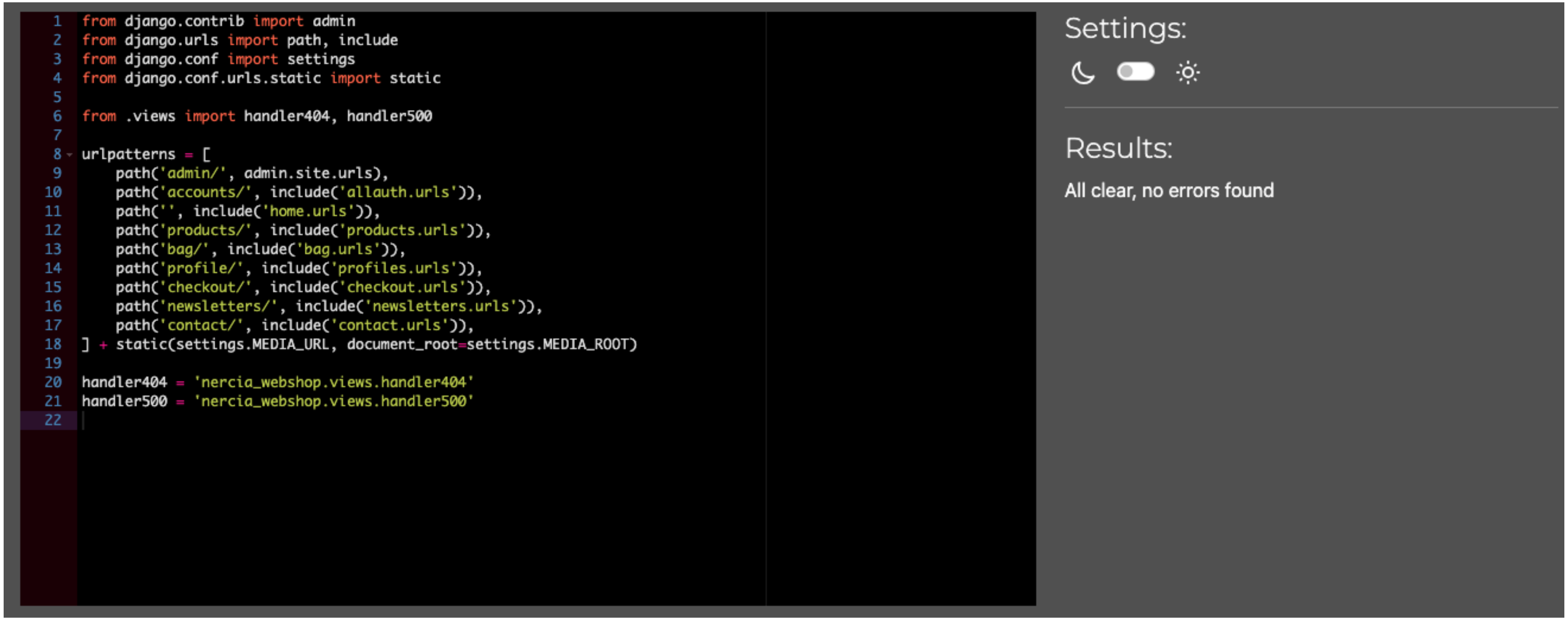
Settings:

Results:

All clear, no errors found

Python Validation

Nercia_webshop/urls.py



```python
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, handler500

urlpatterns = [
    path('admin/', admin.site.urls),
    path('accounts/', include('allauth.urls')),
    path('', include('home.urls')),
    path('products/', include('products.urls')),
    path('bag/', include('bag.urls')),
    path('profile/', include('profiles.urls')),
    path('checkout/', include('checkout.urls')),
    path('newsletters/', include('newsletters.urls')),
    path('contact/', include('contact.urls')),
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)

handler404 = 'nercia_webshop.views.handler404'
handler500 = 'nercia_webshop.views.handler500'
```

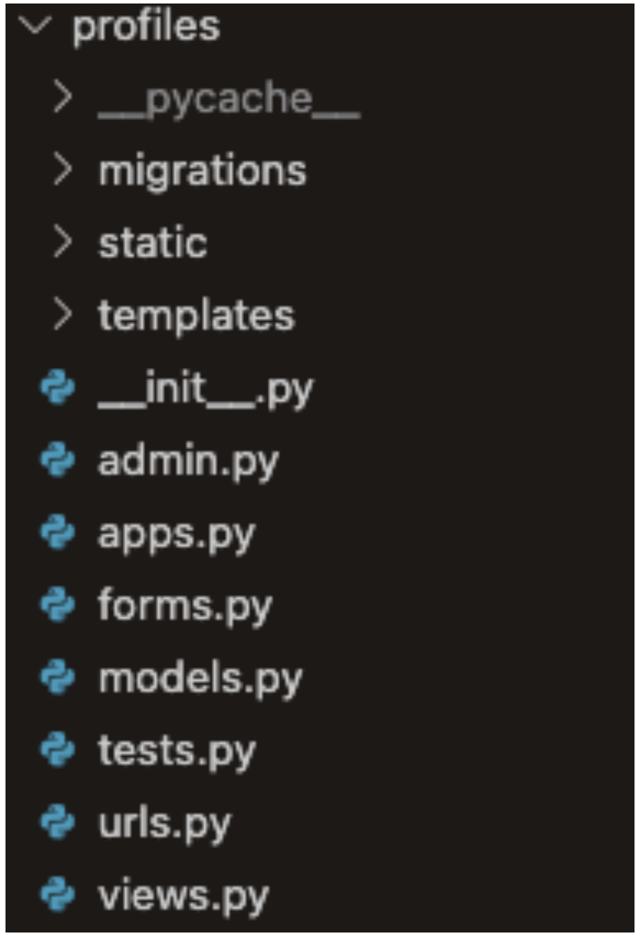
Settings:

Results:

All clear, no errors found

Python Validation

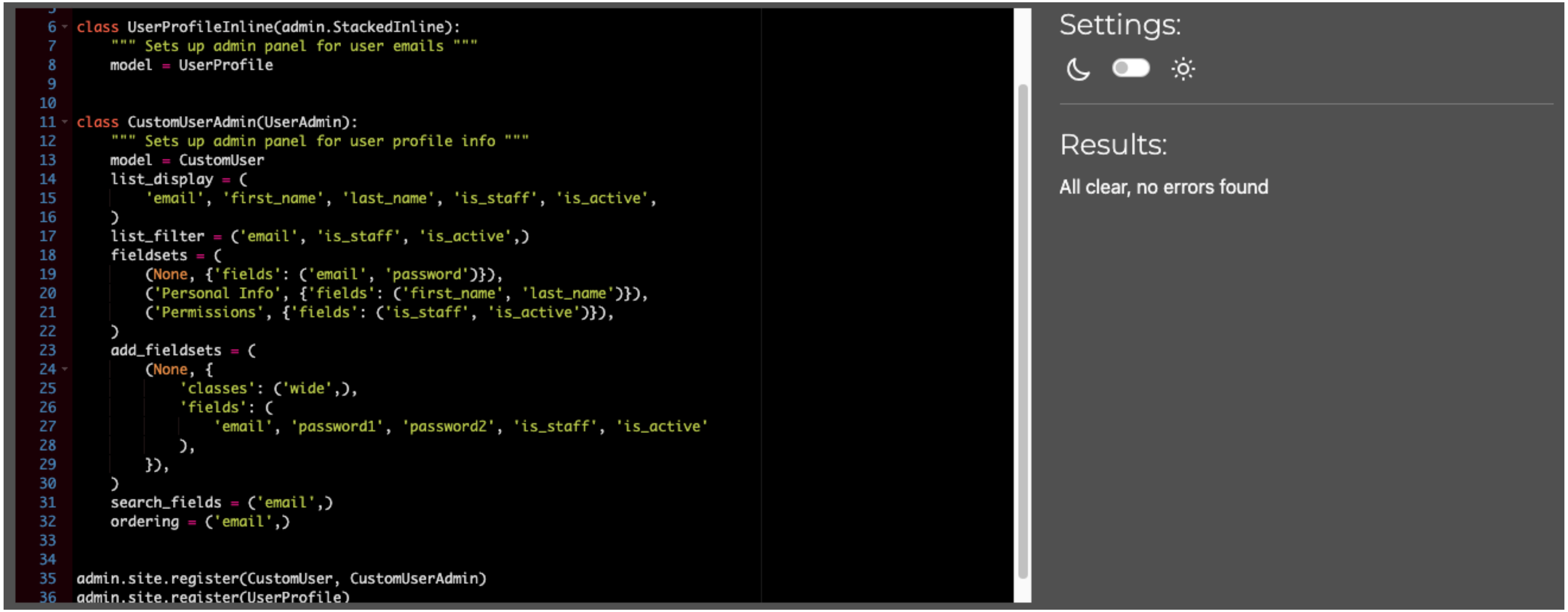
App: profiles



```
profiles
  __pycache__
  migrations
  static
  templates
  __init__.py
  admin.py
  apps.py
  forms.py
  models.py
  tests.py
  urls.py
  views.py
```

Python Validation

profiles/admin.py



```
class UserProfileInline(admin.StackedInline):
    """ Sets up admin panel for user emails """
    model = UserProfile


class CustomUserAdmin(UserAdmin):
    """ Sets up admin panel for user profile info """
    model = CustomUser
    list_display = (
        'email', 'first_name', 'last_name', 'is_staff', 'is_active',
    )
    list_filter = ('email', 'is_staff', 'is_active',)
    fieldsets = (
        (None, {'fields': ('email', 'password')}),
        ('Personal Info', {'fields': ('first_name', 'last_name')}),
        ('Permissions', {'fields': ('is_staff', 'is_active')}),
    )
    add_fieldsets = (
        (None, {
            'classes': ('wide',),
            'fields': (
                'email', 'password1', 'password2', 'is_staff', 'is_active'
            ),
        }),
    )
    search_fields = ('email',)
    ordering = ('email',)


admin.site.register(CustomUser, CustomUserAdmin)
admin.site.register(UserProfile)
```

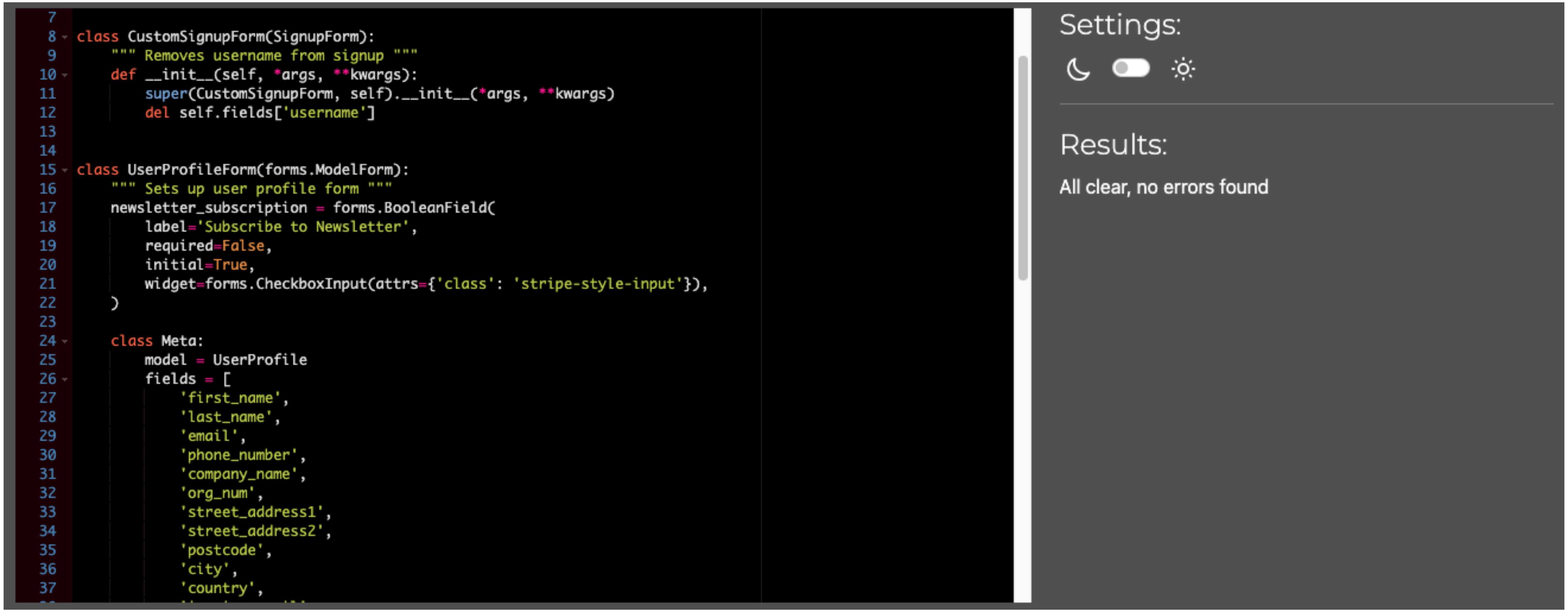
Settings:

Results:

All clear, no errors found

Python Validation

profiles/forms.py



```

class CustomSignupForm(SignupForm):
    """ Removes username from signup """
    def __init__(self, *args, **kwargs):
        super(CustomSignupForm, self).__init__(*args, **kwargs)
        del self.fields['username']


class UserProfileForm(forms.ModelForm):
    """ Sets up user profile form """
    newsletter_subscription = forms.BooleanField(
        label='Subscribe to Newsletter',
        required=False,
        initial=True,
        widget=forms.CheckboxInput(attrs={'class': 'stripe-style-input'}),
    )

    class Meta:
        model = UserProfile
        fields = [
            'first_name',
            'last_name',
            'email',
            'phone_number',
            'company_name',
            'org_num',
            'street_address1',
            'street_address2',
            'postcode',
            'city',
            'country',
```

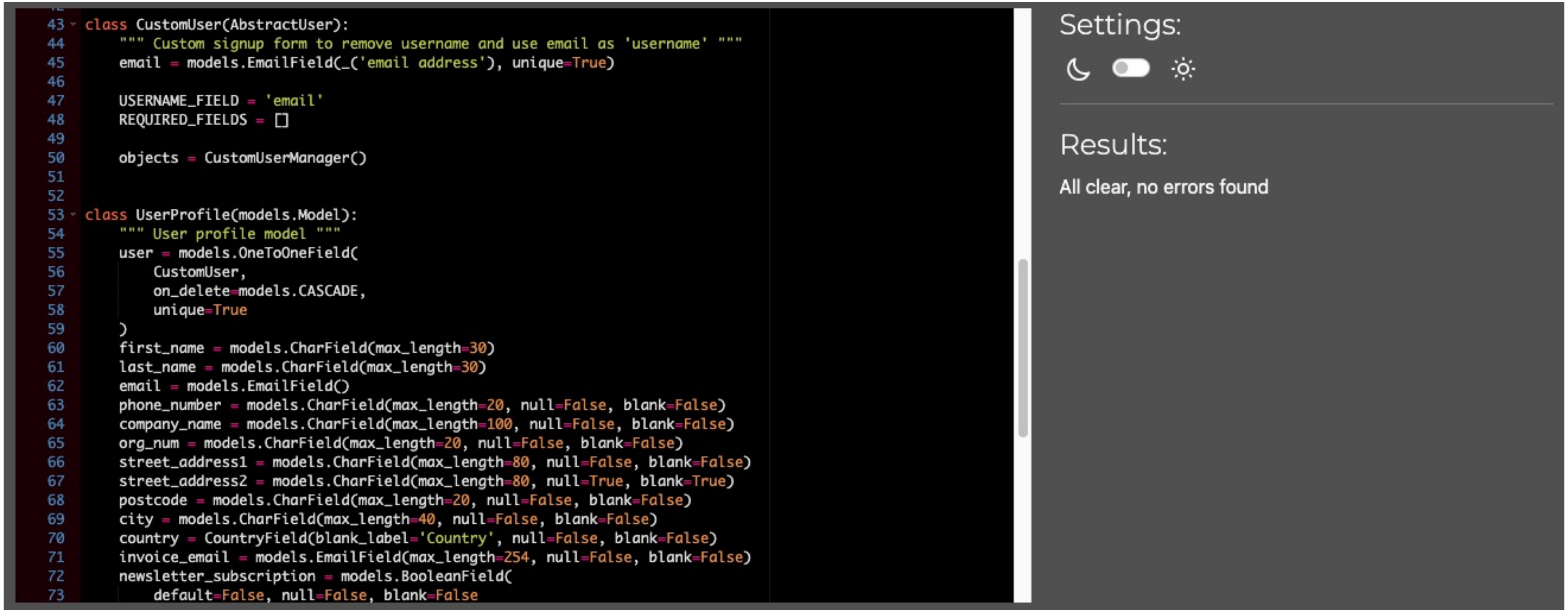
Settings:

Results:

All clear, no errors found

Python Validation

profiles/models.py



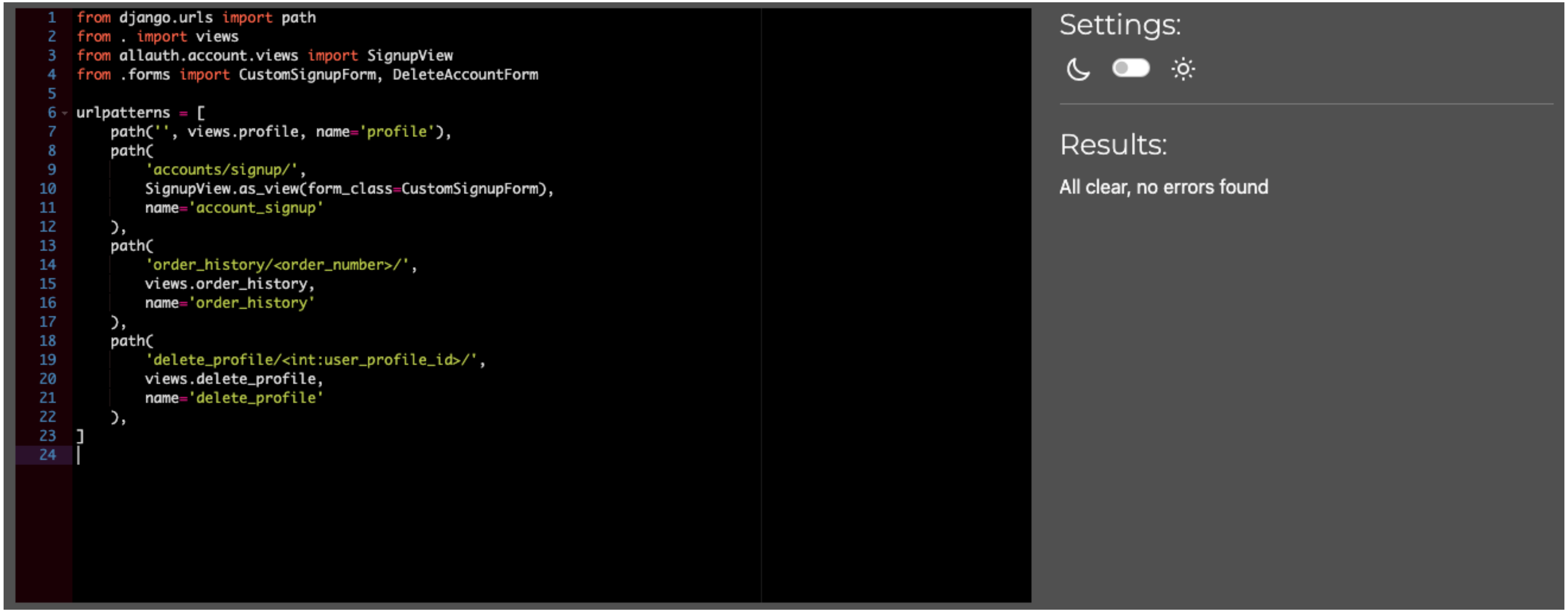
```python
class CustomUser(AbstractUser):
    """ Custom signup form to remove username and use email as 'username' """
    email = models.EmailField(_('email address'), unique=True)

    USERNAME_FIELD = 'email'
    REQUIRED_FIELDS = []

    objects = CustomUserManager()


class UserProfile(models.Model):
    """ User profile model """
    user = models.OneToOneField(
        CustomUser,
        on_delete=models.CASCADE,
        unique=True
    )
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=30)
    email = models.EmailField()
    phone_number = models.CharField(max_length=20, null=False, blank=False)
    company_name = models.CharField(max_length=100, null=False, blank=False)
    org_num = models.CharField(max_length=20, null=False, blank=False)
    street_address1 = models.CharField(max_length=80, null=False, blank=False)
    street_address2 = models.CharField(max_length=80, null=True, blank=True)
    postcode = models.CharField(max_length=20, null=False, blank=False)
    city = models.CharField(max_length=40, null=False, blank=False)
    country = CountryField(blank_label='Country', null=False, blank=False)
    invoice_email = models.EmailField(max_length=254, null=False, blank=False)
    newsletter_subscription = models.BooleanField(
        default=False, null=False, blank=False
```

Results:

All clear, no errors found

Python Validation

profiles/urls.py



```python
from django.urls import path
from . import views
from allauth.account.views import SignupView
from .forms import CustomSignupForm, DeleteAccountForm

urlpatterns = [
    path('', views.profile, name='profile'),
    path(
        'accounts/signup/',
        SignupView.as_view(form_class=CustomSignupForm),
        name='account_signup'
    ),
    path(
        'order_history/<order_number>/',
        views.order_history,
        name='order_history'
    ),
    path(
        'delete_profile/<int:user_profile_id>/',
        views.delete_profile,
        name='delete_profile'
    ),
]
```

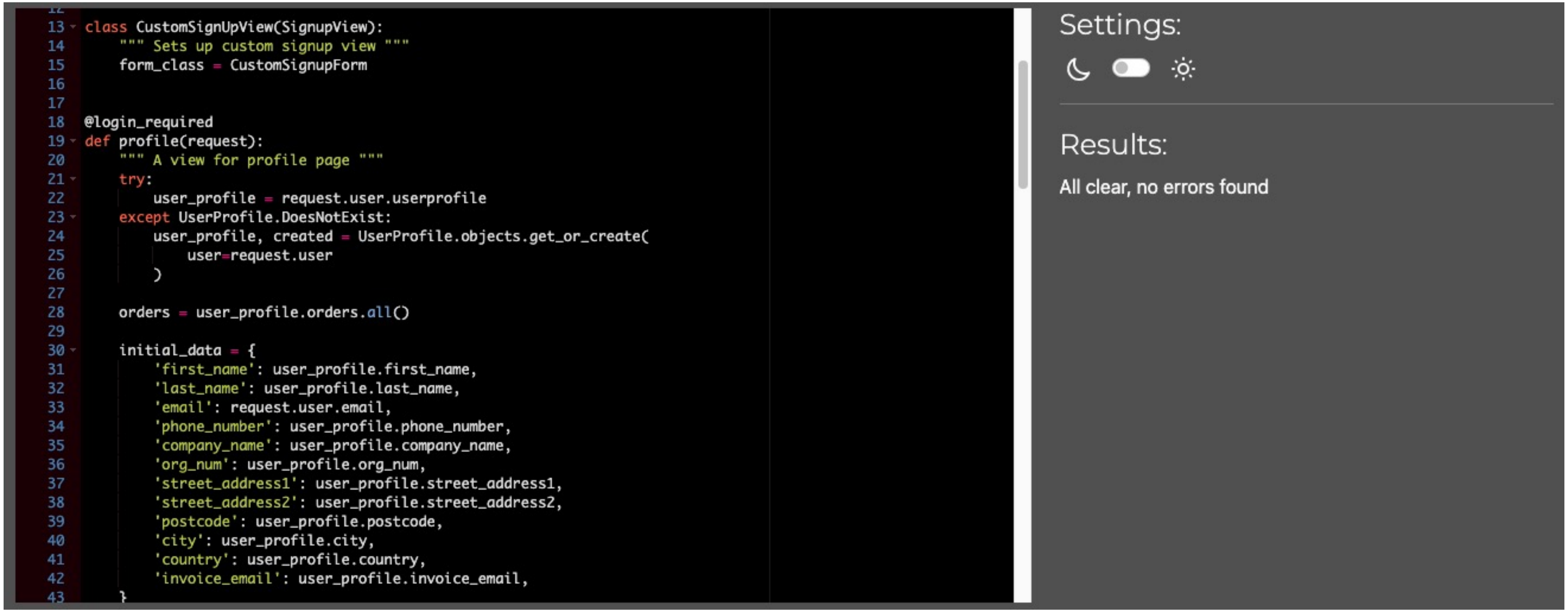
Settings:

Results:

All clear, no errors found

Python Validation

profiles/views.py



```
class CustomSignUpView(SignupView):
    """ Sets up custom signup view """
    form_class = CustomSignupForm


@login_required
def profile(request):
    """ A view for profile page """
    try:
        user_profile = request.user.userprofile
    except UserProfile.DoesNotExist:
        user_profile, created = UserProfile.objects.get_or_create(
            user=request.user
        )

    orders = user_profile.orders.all()

    initial_data = {
        'first_name': user_profile.first_name,
        'last_name': user_profile.last_name,
        'email': request.user.email,
        'phone_number': user_profile.phone_number,
        'company_name': user_profile.company_name,
        'org_num': user_profile.org_num,
        'street_address1': user_profile.street_address1,
        'street_address2': user_profile.street_address2,
        'postcode': user_profile.postcode,
        'city': user_profile.city,
        'country': user_profile.country,
        'invoice_email': user_profile.invoice_email,
    }
```

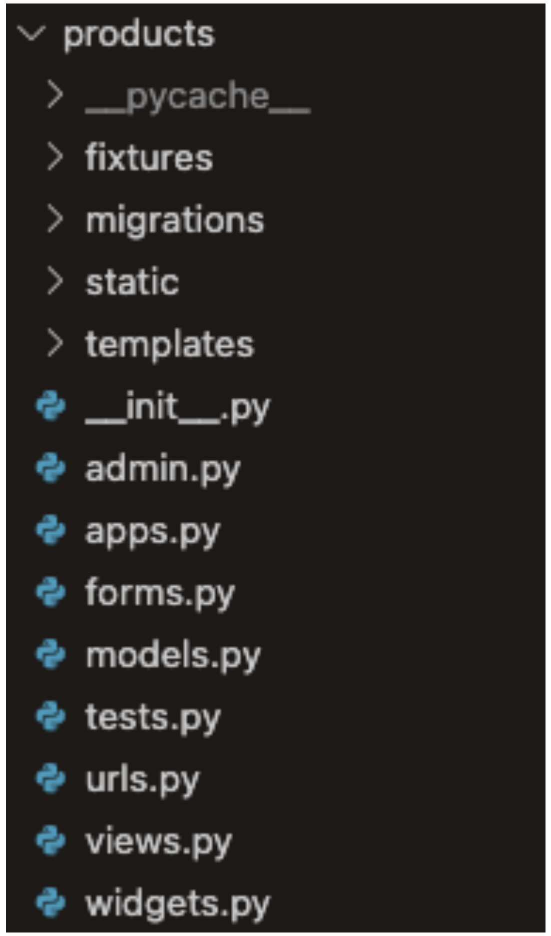
Settings:

Results:

All clear, no errors found

Python Validation

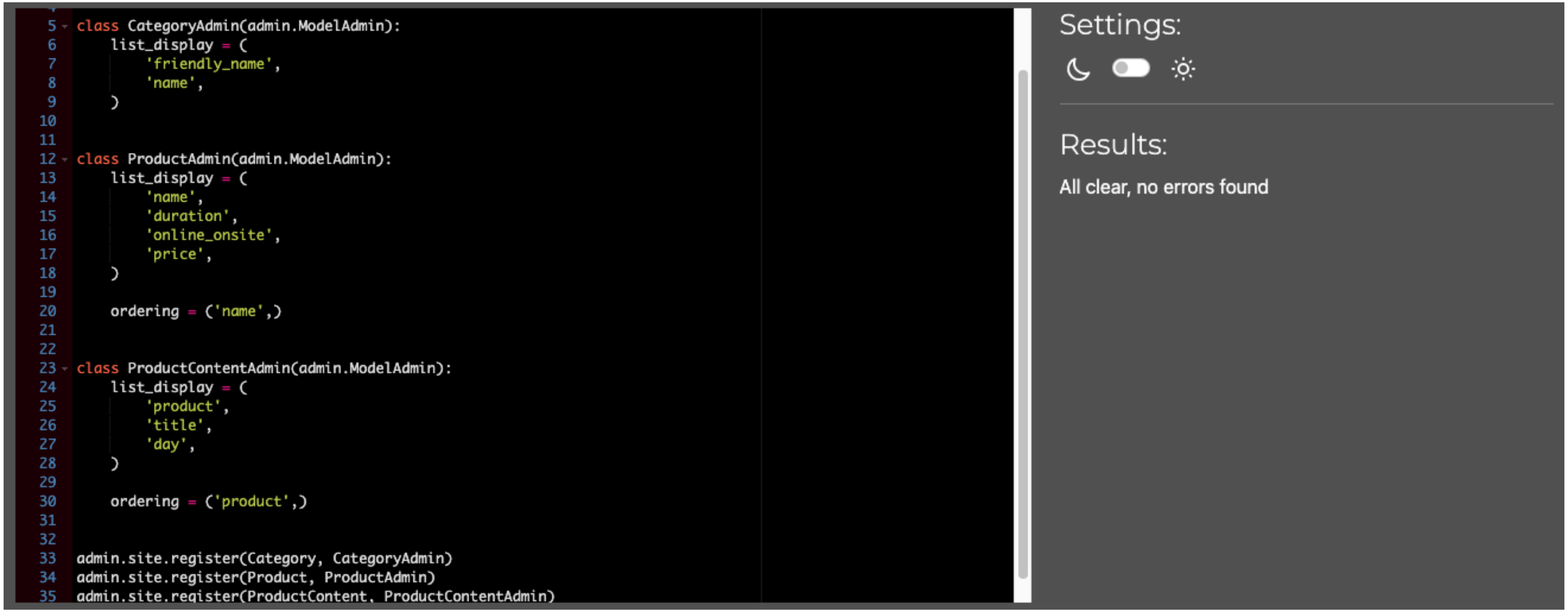
App: products



```
products
  __pycache__
  fixtures
  migrations
  static
  templates
  __init__.py
  admin.py
  apps.py
  forms.py
  models.py
  tests.py
  urls.py
  views.py
  widgets.py
```

Python Validation

products/admin.py



```
class CategoryAdmin(admin.ModelAdmin):
    list_display = (
        'friendly_name',
        'name',
    )


class ProductAdmin(admin.ModelAdmin):
    list_display = (
        'name',
        'duration',
        'online_onsite',
        'price',
    )

    ordering = ('name',)


class ProductContentAdmin(admin.ModelAdmin):
    list_display = (
        'product',
        'title',
        'day',
    )

    ordering = ('product',)


admin.site.register(Category, CategoryAdmin)
admin.site.register(Product, ProductAdmin)
admin.site.register(ProductContent, ProductContentAdmin)
```

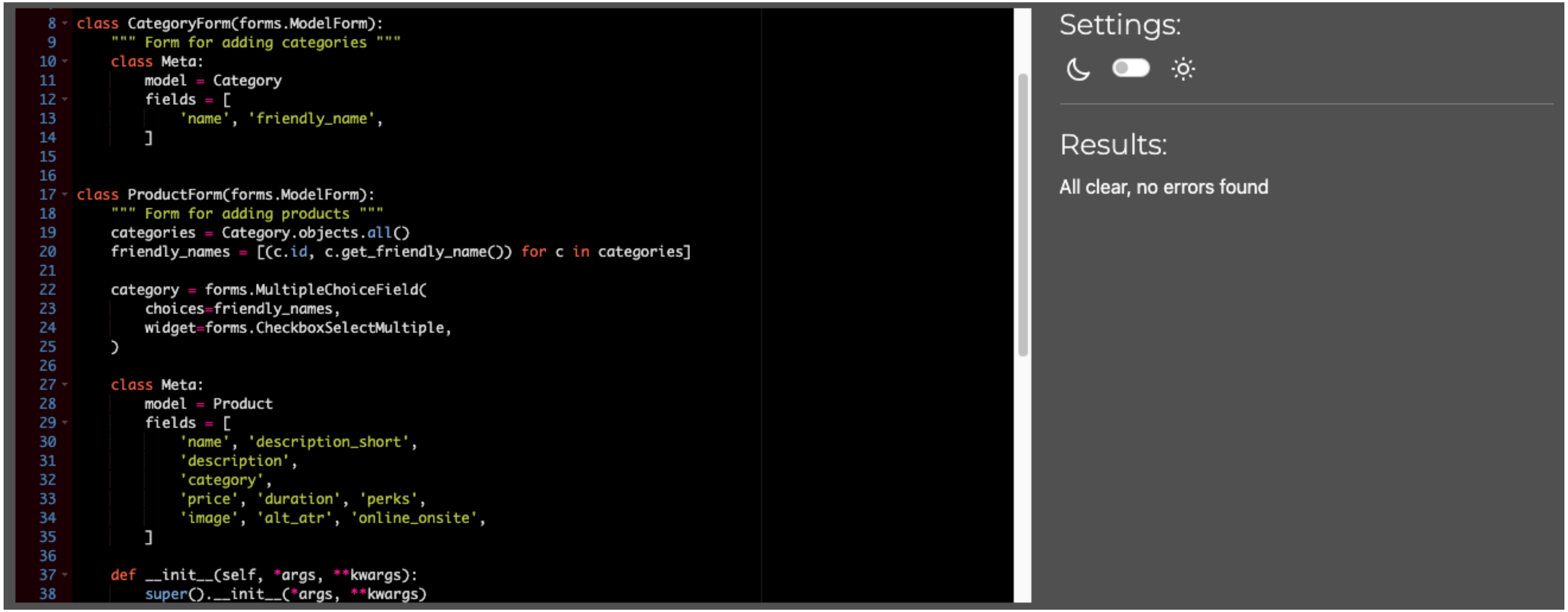
Settings:

Results:

All clear, no errors found

Python Validation

products/forms.py



```python
class CategoryForm(forms.ModelForm):
    """ Form for adding categories """
    class Meta:
        model = Category
        fields = [
            'name', 'friendly_name',
        ]


class ProductForm(forms.ModelForm):
    """ Form for adding products """
    categories = Category.objects.all()
    friendly_names = [(c.id, c.get_friendly_name()) for c in categories]

    category = forms.MultipleChoiceField(
        choices=friendly_names,
        widget=forms.CheckboxSelectMultiple,
    )

    class Meta:
        model = Product
        fields = [
            'name', 'description_short',
            'description',
            'category',
            'price', 'duration', 'perks',
            'image', 'alt_atr', 'online_onsite',
        ]

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
```

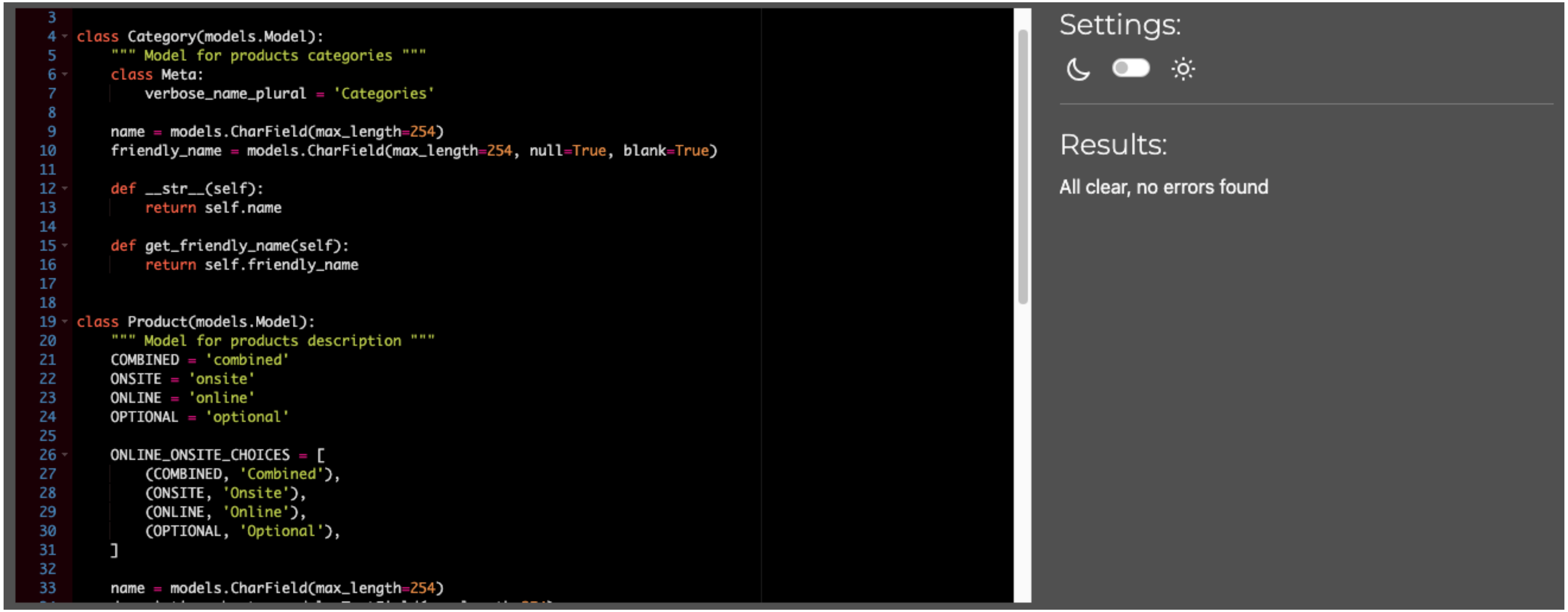
Settings:

Results:

All clear, no errors found

Python Validation

products/models.py



```python

class Category(models.Model):
    """ Model for products categories """
    class Meta:
        verbose_name_plural = 'Categories'

    name = models.CharField(max_length=254)
    friendly_name = models.CharField(max_length=254, null=True, blank=True)

    def __str__(self):
        return self.name

    def get_friendly_name(self):
        return self.friendly_name


class Product(models.Model):
    """ Model for products description """
    COMBINED = 'combined'
    ONSITE = 'onsite'
    ONLINE = 'online'
    OPTIONAL = 'optional'

    ONLINE_ONSITE_CHOICES = [
        (COMBINED, 'Combined'),
        (ONSITE, 'Onsite'),
        (ONLINE, 'Online'),
        (OPTIONAL, 'Optional'),
    ]

    name = models.CharField(max_length=254)
```

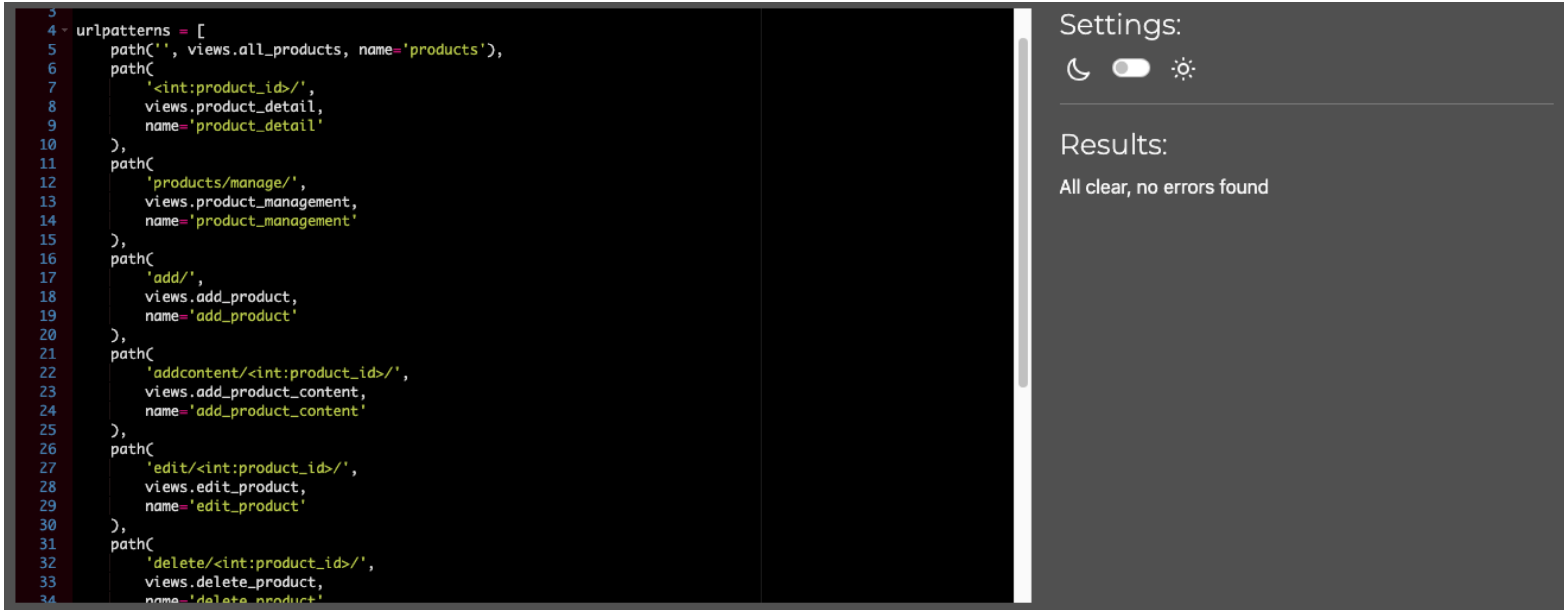
Settings:

Results:

All clear, no errors found

Python Validation

products/urls.py



```
urlpatterns = [
    path('', views.all_products, name='products'),
    path(
        '<int:product_id>/',
        views.product_detail,
        name='product_detail'
    ),
    path(
        'products/manage/',
        views.product_management,
        name='product_management'
    ),
    path(
        'add/',
        views.add_product,
        name='add_product'
    ),
    path(
        'addcontent/<int:product_id>/',
        views.add_product_content,
        name='add_product_content'
    ),
    path(
        'edit/<int:product_id>/',
        views.edit_product,
        name='edit_product'
    ),
    path(
        'delete/<int:product_id>/',
        views.delete_product,
        name='delete_product'
```

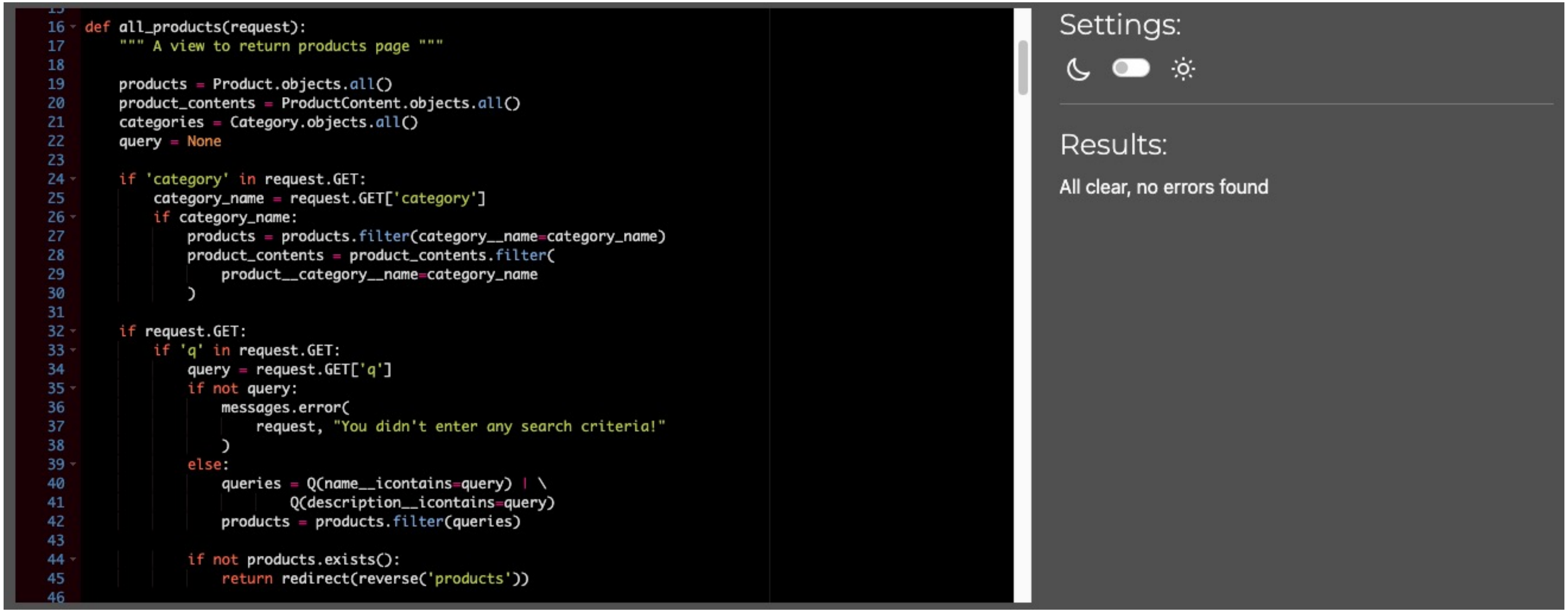
Settings:

Results:

All clear, no errors found

Python Validation

products/views.py



```python
def all_products(request):
    """ A view to return products page """

    products = Product.objects.all()
    product_contents = ProductContent.objects.all()
    categories = Category.objects.all()
    query = None

    if 'category' in request.GET:
        category_name = request.GET['category']
        if category_name:
            products = products.filter(category__name=category_name)
            product_contents = product_contents.filter(
                product__category__name=category_name
            )

    if request.GET:
        if 'q' in request.GET:
            query = request.GET['q']
            if not query:
                messages.error(
                    request, "You didn't enter any search criteria!"
                )
            else:
                queries = Q(name__icontains=query) | \
                          Q(description__icontains=query)
                products = products.filter(queries)

            if not products.exists():
                return redirect(reverse('products'))
```

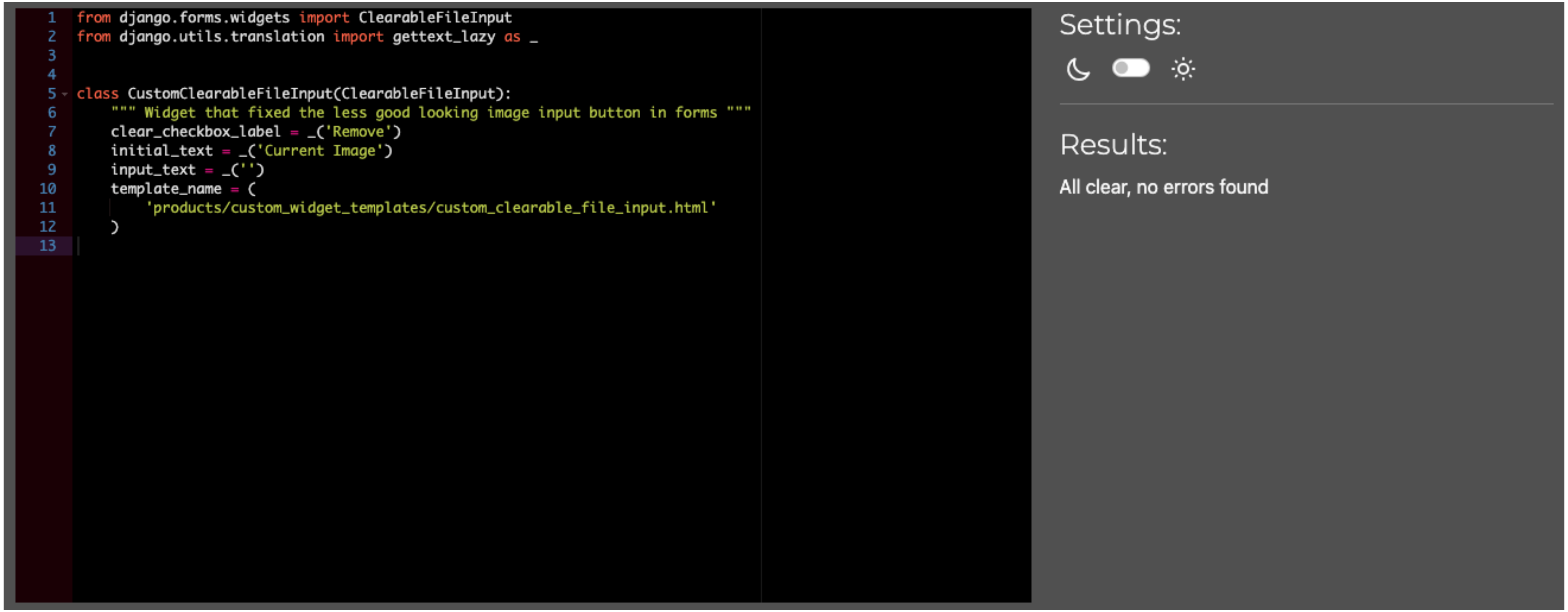
Settings:

Results:

All clear, no errors found

Python Validation

products/widgets.py



```python
from django.forms.widgets import ClearableFileInput
from django.utils.translation import gettext_lazy as _


class CustomClearableFileInput(ClearableFileInput):
    """ Widget that fixed the less good looking image input button in forms """
    clear_checkbox_label = _('Remove')
    initial_text = _('Current Image')
    input_text = _('')
    template_name = (
        'products/custom_widget_templates/custom_clearable_file_input.html'
    )
```

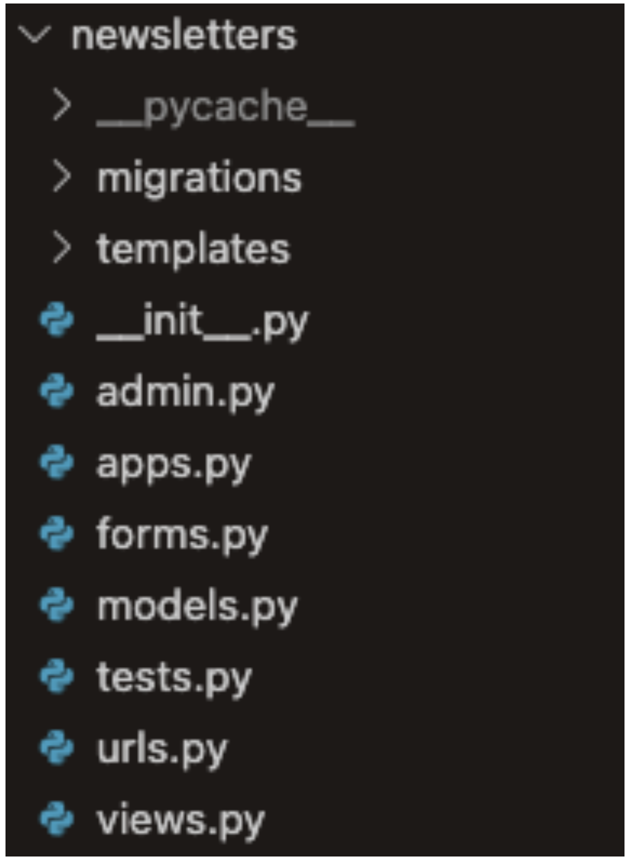
Settings:

Results:

All clear, no errors found

Python Validation

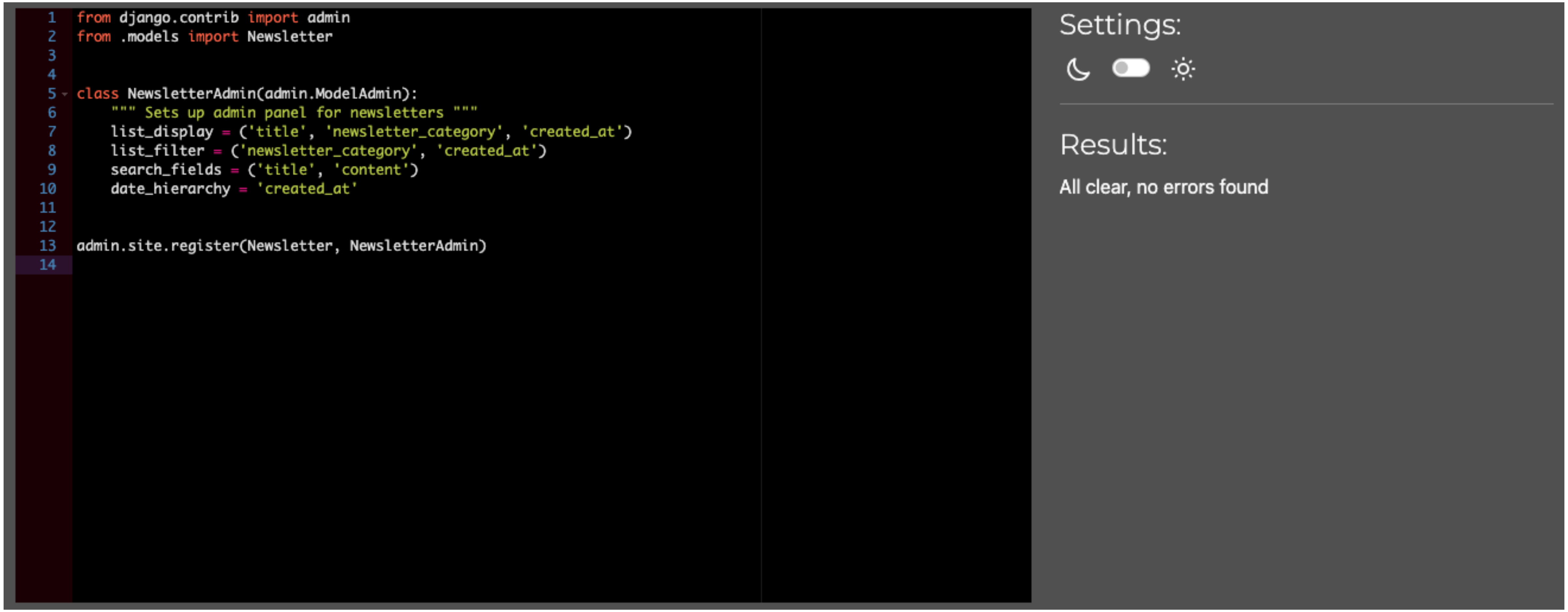
App: newsletters



```
newsletters
  __pycache__
  migrations
  templates
  __init__.py
  admin.py
  apps.py
  forms.py
  models.py
  tests.py
  urls.py
  views.py
```

Python Validation

newsletters/admin.py



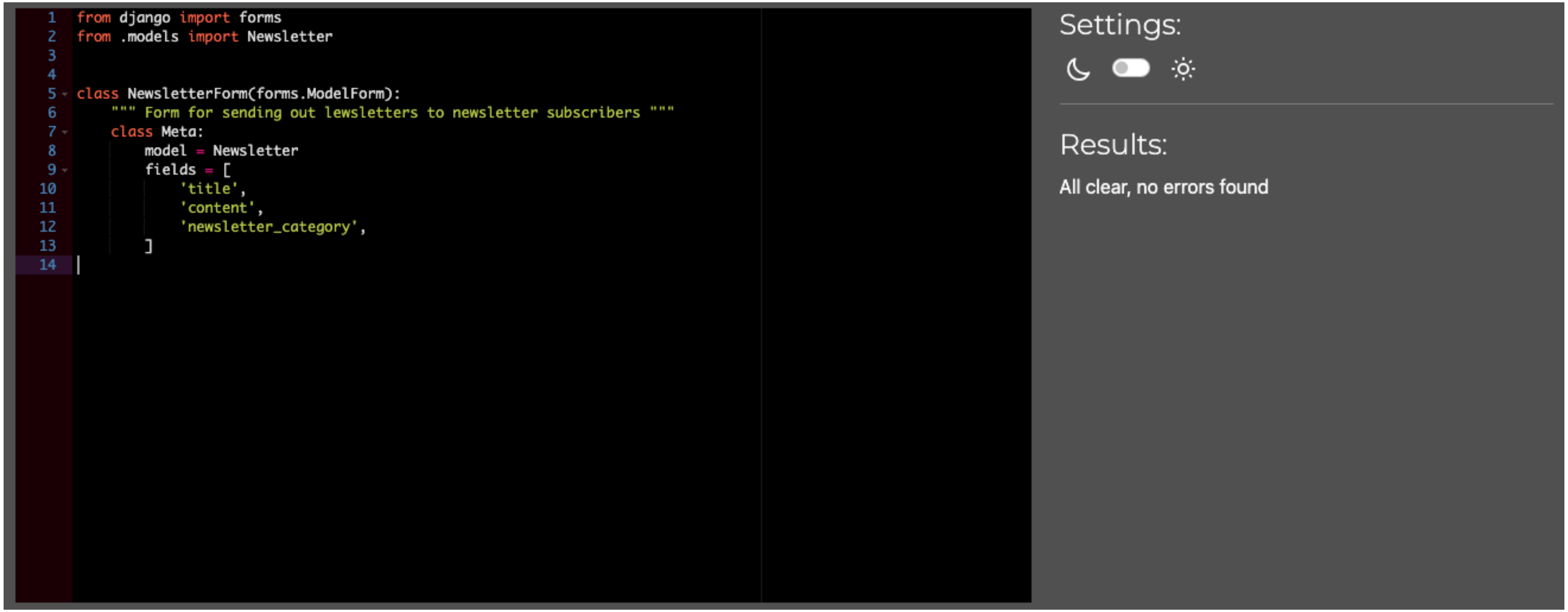
```python
from django.contrib import admin
from .models import Newsletter


class NewsletterAdmin(admin.ModelAdmin):
    """ Sets up admin panel for newsletters """
    list_display = ('title', 'newsletter_category', 'created_at')
    list_filter = ('newsletter_category', 'created_at')
    search_fields = ('title', 'content')
    date_hierarchy = 'created_at'


admin.site.register(Newsletter, NewsletterAdmin)
```

Results:

All clear, no errors found

Python Validation

newsletters/forms.py



```
from django import forms
from .models import Newsletter


class NewsletterForm(forms.ModelForm):
    """ Form for sending out lewsletters to newsletter subscribers """
    class Meta:
        model = Newsletter
        fields = [
            'title',
            'content',
            'newsletter_category',
        ]

```

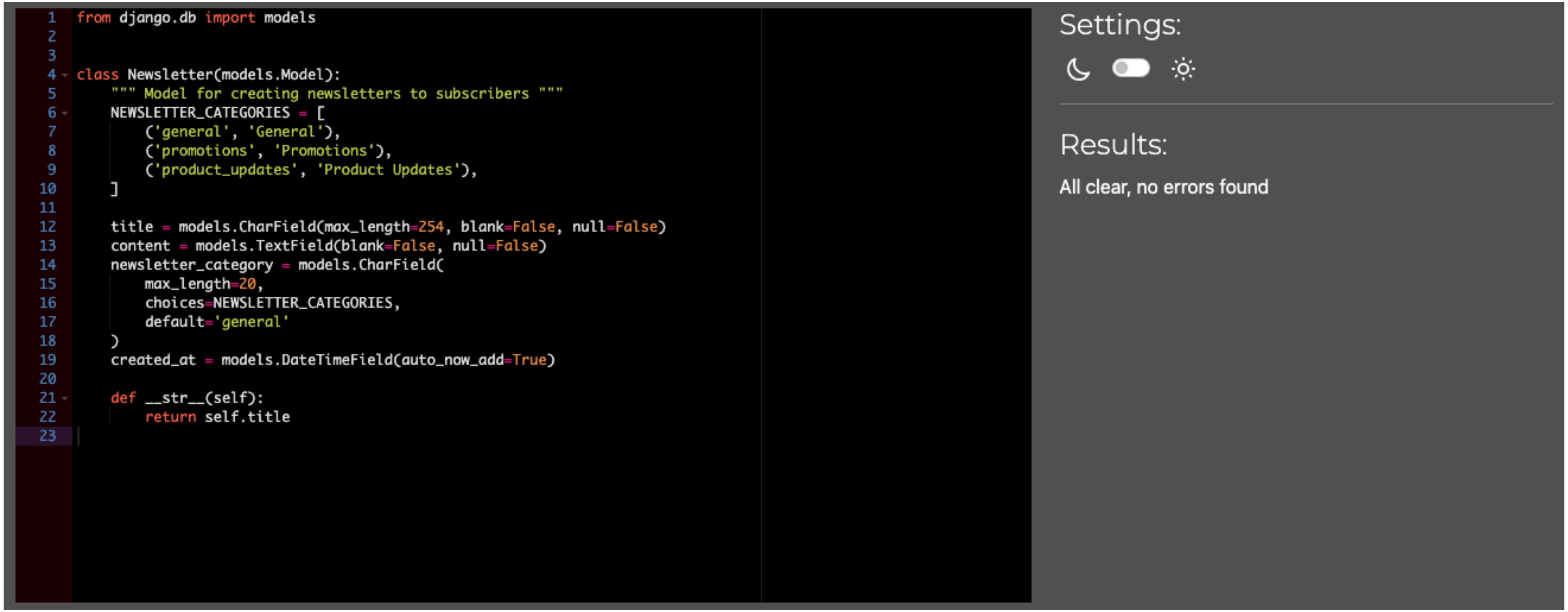
Settings:

Results:

All clear, no errors found

Python Validation

newsletters/models.py



```
from django.db import models


class Newsletter(models.Model):
    """ Model for creating newsletters to subscribers """
    NEWSLETTER_CATEGORIES = [
        ('general', 'General'),
        ('promotions', 'Promotions'),
        ('product_updates', 'Product Updates'),
    ]

    title = models.CharField(max_length=254, blank=False, null=False)
    content = models.TextField(blank=False, null=False)
    newsletter_category = models.CharField(
        max_length=20,
        choices=NEWSLETTER_CATEGORIES,
        default='general'
    )
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.title

```

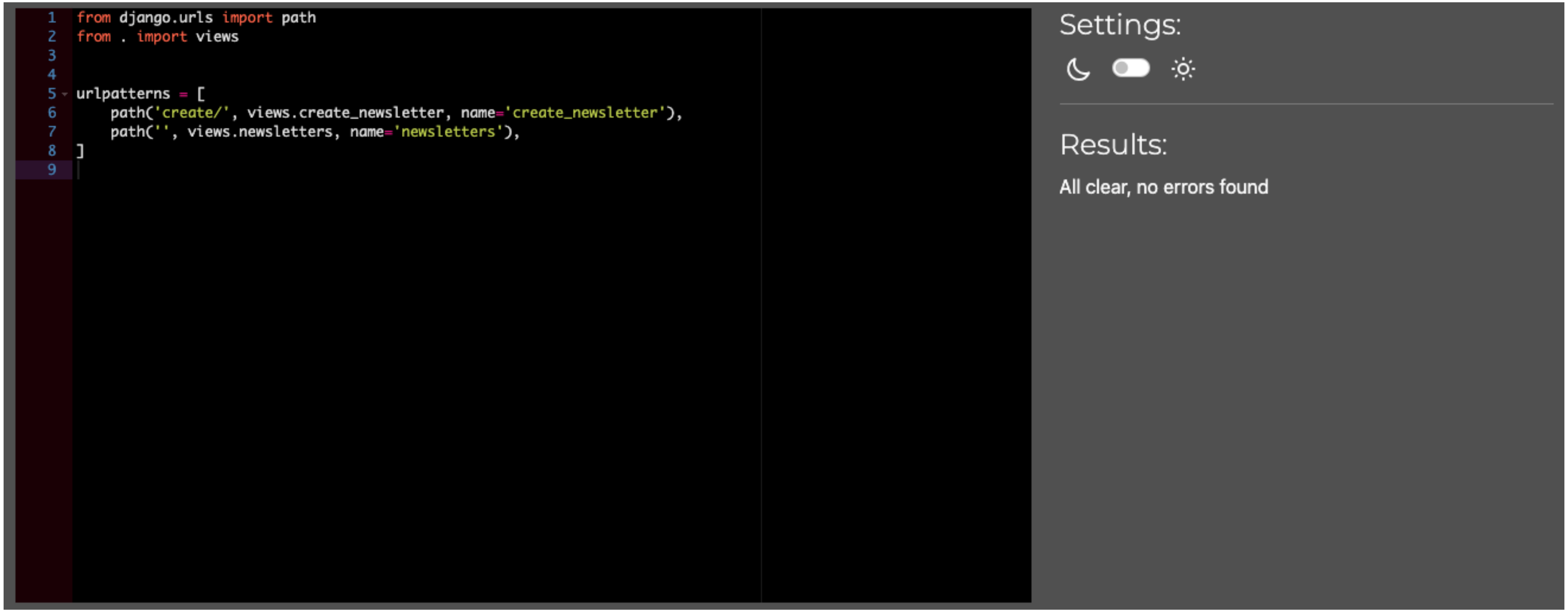
Settings:

Results:

All clear, no errors found

Python Validation

newsletters/urls.py



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


urlpatterns = [
    path('create/', views.create_newsletter, name='create_newsletter'),
    path('', views.newsletters, name='newsletters'),
]
```

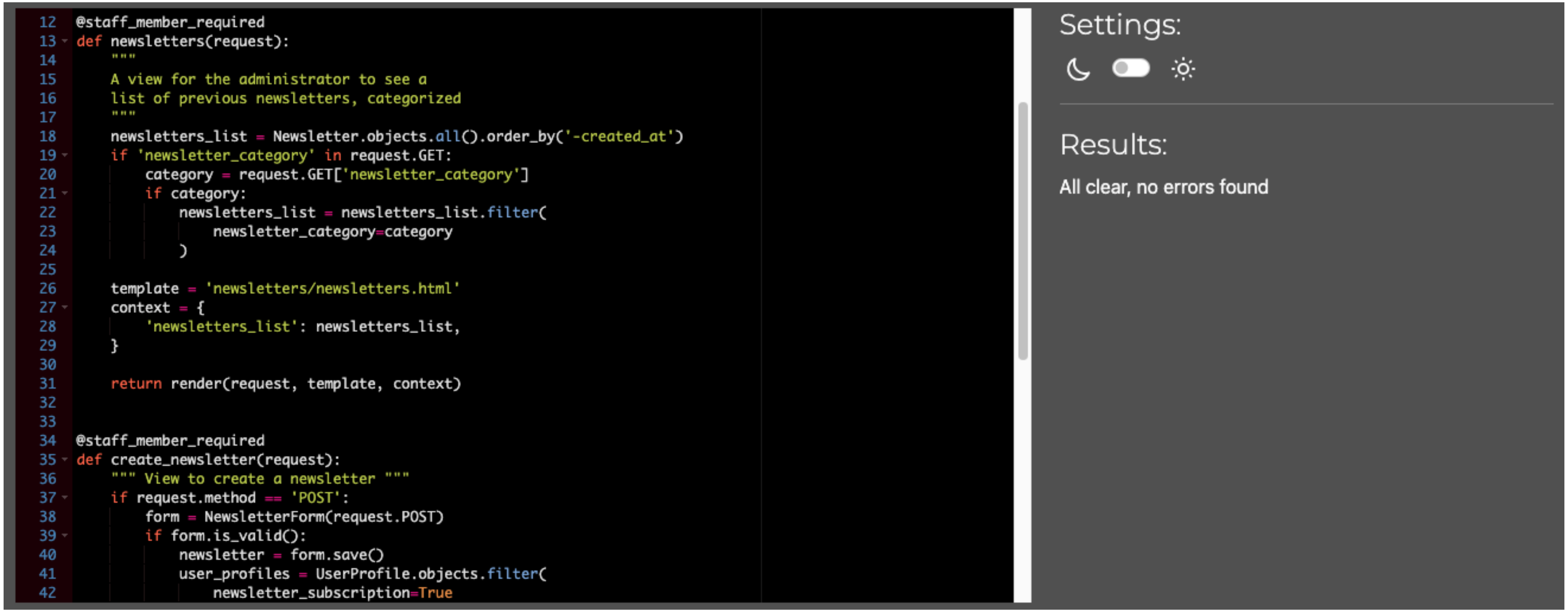
Settings:

Results:

All clear, no errors found

Python Validation

newsletters/views.py



```python
@staff_member_required
def newsletters(request):
    """
    A view for the administrator to see a
    list of previous newsletters, categorized
    """
    newsletters_list = Newsletter.objects.all().order_by('-created_at')
    if 'newsletter_category' in request.GET:
        category = request.GET['newsletter_category']
        if category:
            newsletters_list = newsletters_list.filter(
                newsletter_category=category
            )

    template = 'newsletters/newsletters.html'
    context = {
        'newsletters_list': newsletters_list,
    }

    return render(request, template, context)


@staff_member_required
def create_newsletter(request):
    """ View to create a newsletter """
    if request.method == 'POST':
        form = NewsletterForm(request.POST)
        if form.is_valid():
            newsletter = form.save()
            user_profiles = UserProfile.objects.filter(
                newsletter_subscription=True
```

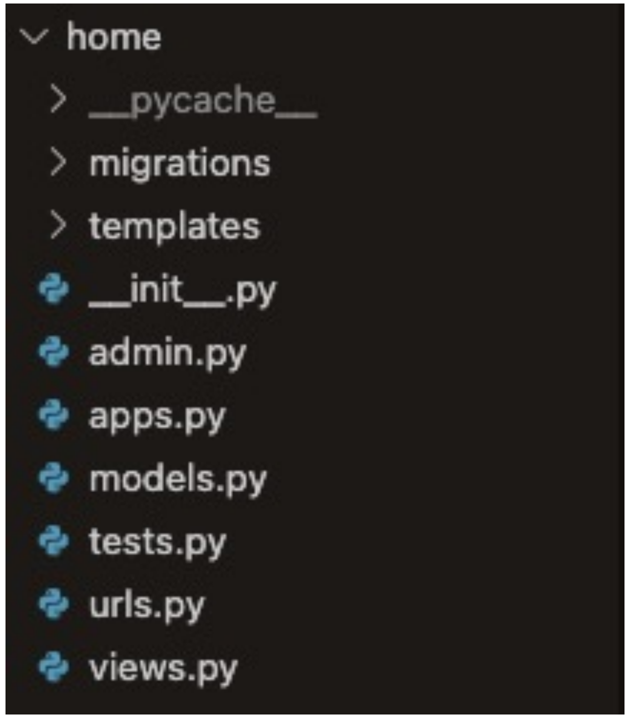
Settings:

Results:

All clear, no errors found

Python Validation

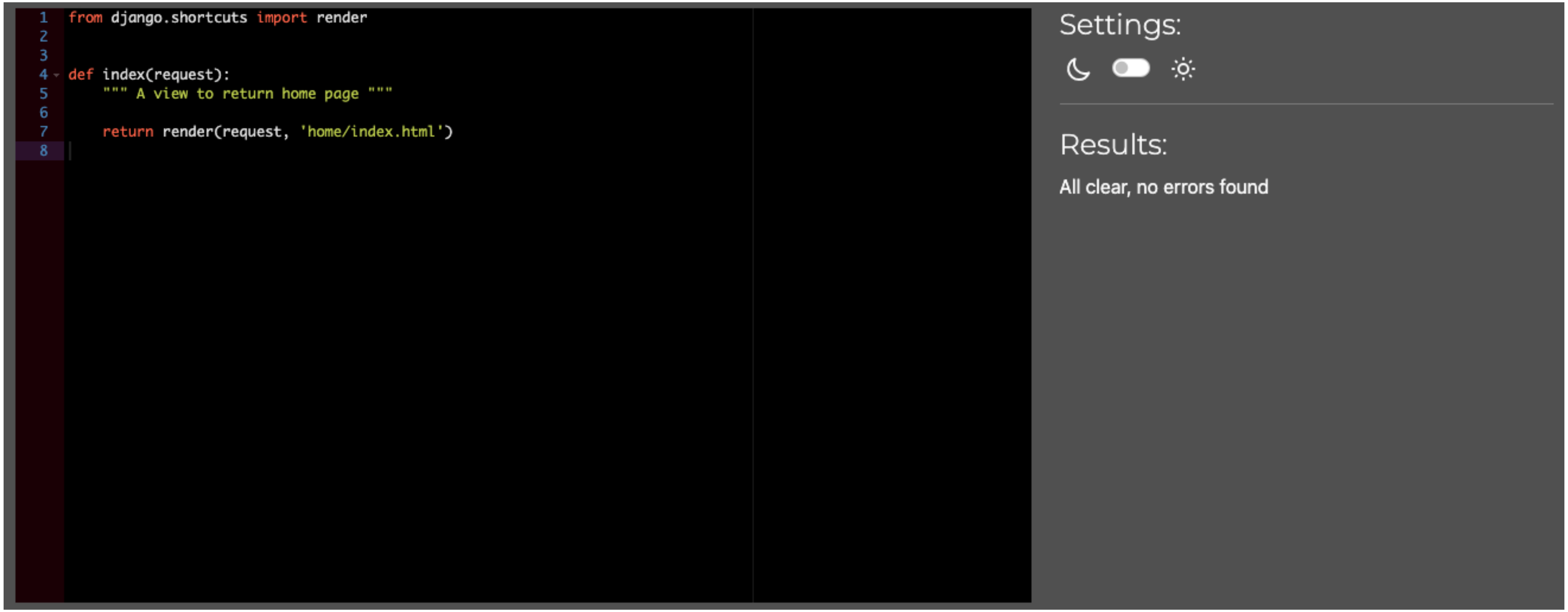
App: home



```
home
  > __pycache__
  > migrations
  > templates
  __init__.py
  admin.py
  apps.py
  models.py
  tests.py
  urls.py
  views.py
```

Python Validation

home/views.py



```python
from django.shortcuts import render


def index(request):
    """ A view to return home page """

    return render(request, 'home/index.html')

```

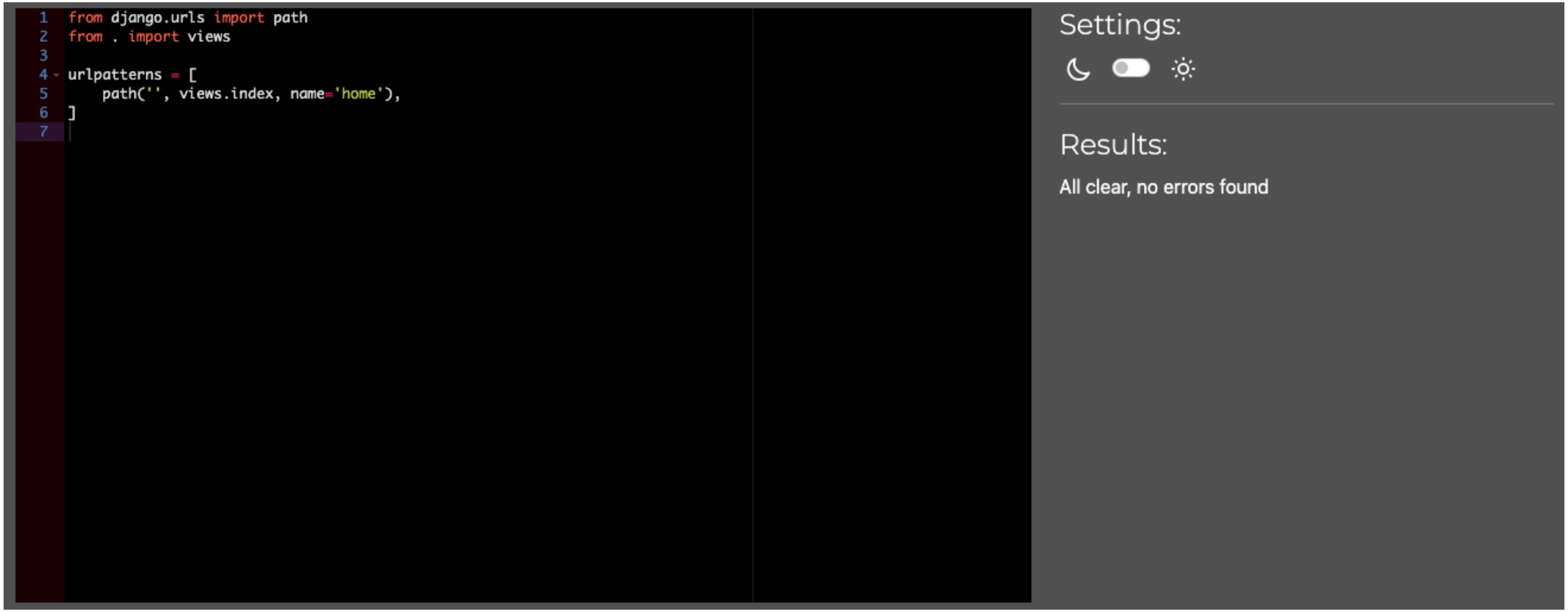
Settings:

Results:

All clear, no errors found

Python Validation

newsletters/urls.py



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

urlpatterns = [
    path('', views.index, name='home'),
]
```

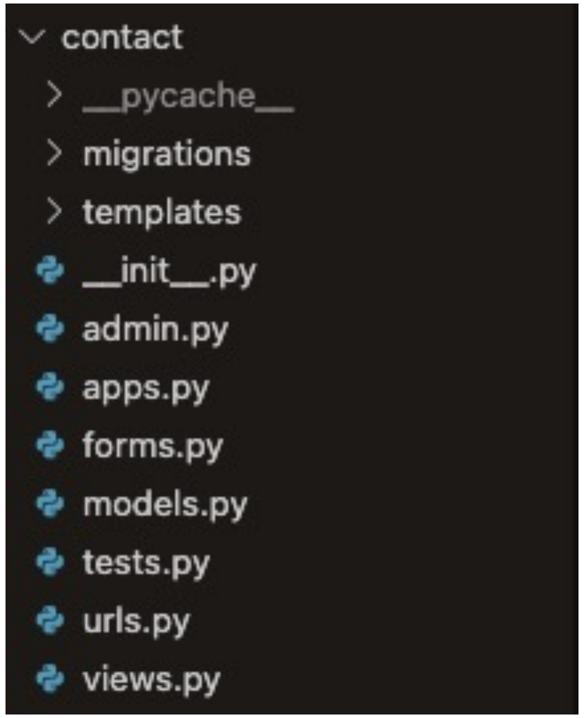
Settings:

Results:

All clear, no errors found

Python Validation

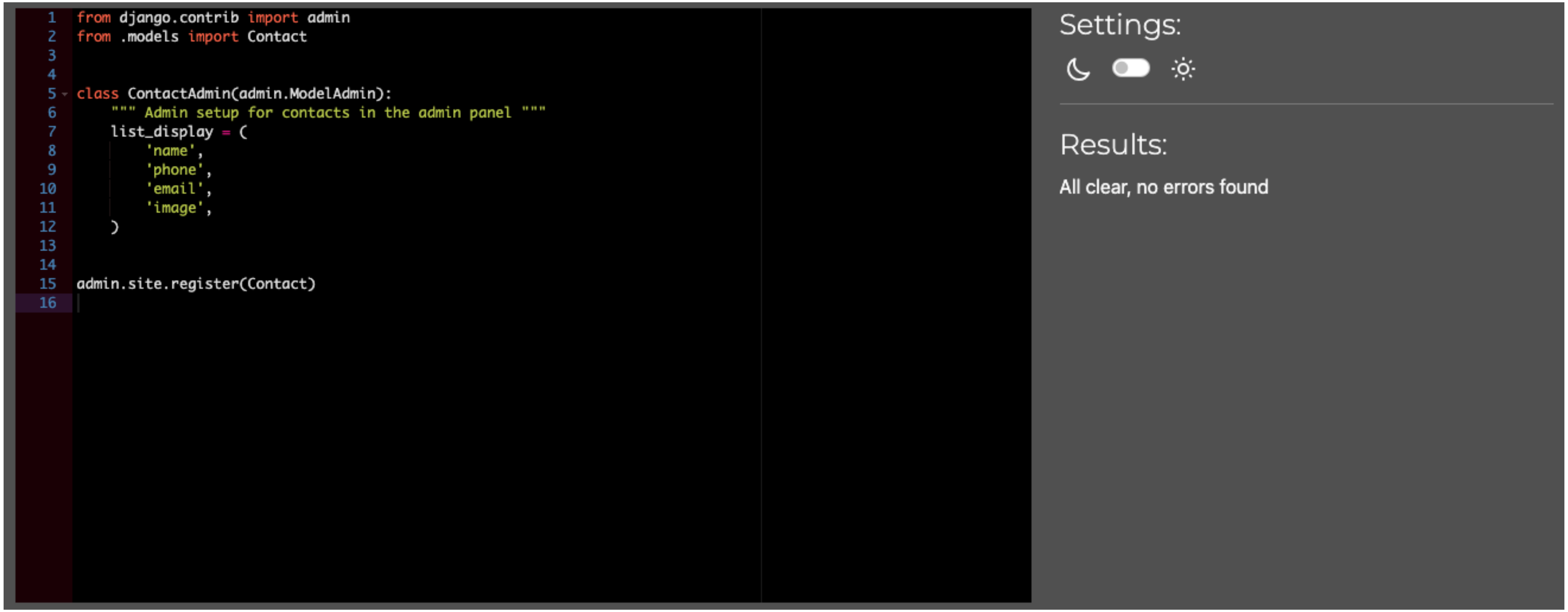
App: contact



```
contact
  __pycache__
  migrations
  templates
  __init__.py
  admin.py
  apps.py
  forms.py
  models.py
  tests.py
  urls.py
  views.py
```

Python Validation

contact/admin.py



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


class ContactAdmin(admin.ModelAdmin):
    """ Admin setup for contacts in the admin panel """
    list_display = (
        'name',
        'phone',
        'email',
        'image',
    )


admin.site.register(Contact)

```

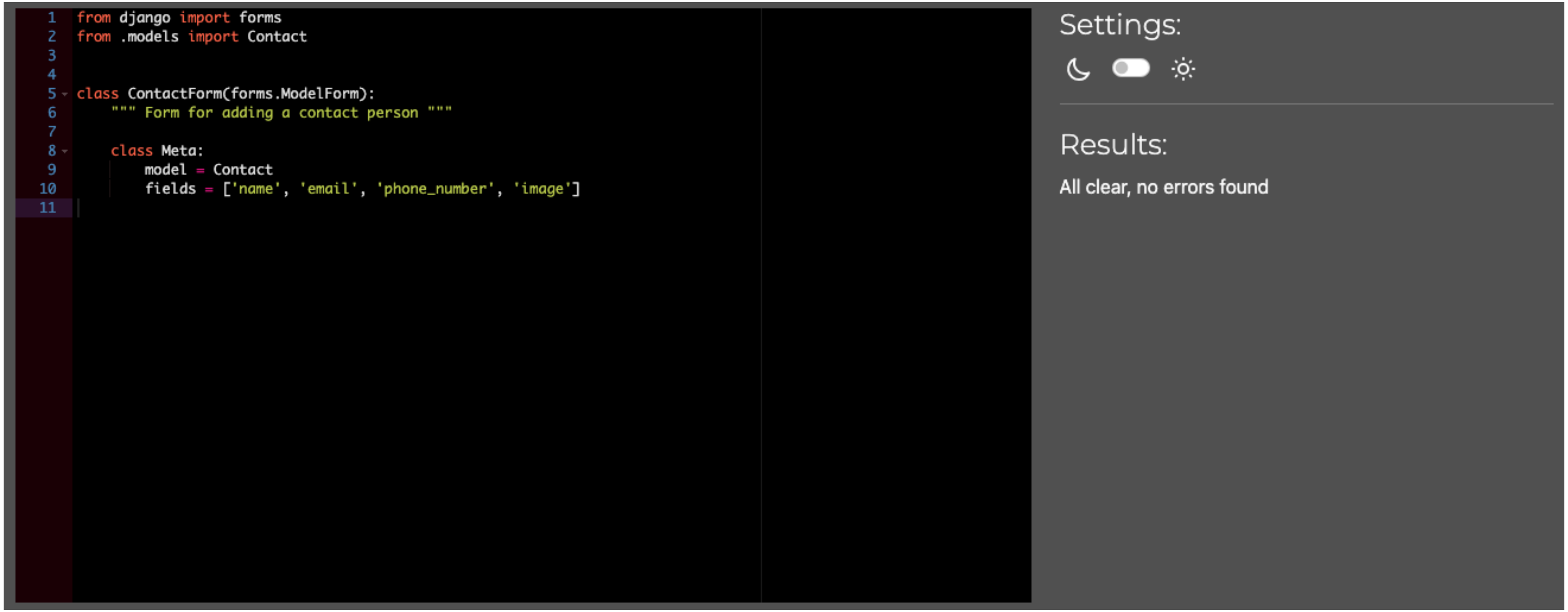
Settings:

Results:

All clear, no errors found

Python Validation

contact/forms.py



```
from django import forms
from .models import Contact


class ContactForm(forms.ModelForm):
    """ Form for adding a contact person """

    class Meta:
        model = Contact
        fields = ['name', 'email', 'phone_number', 'image']
```

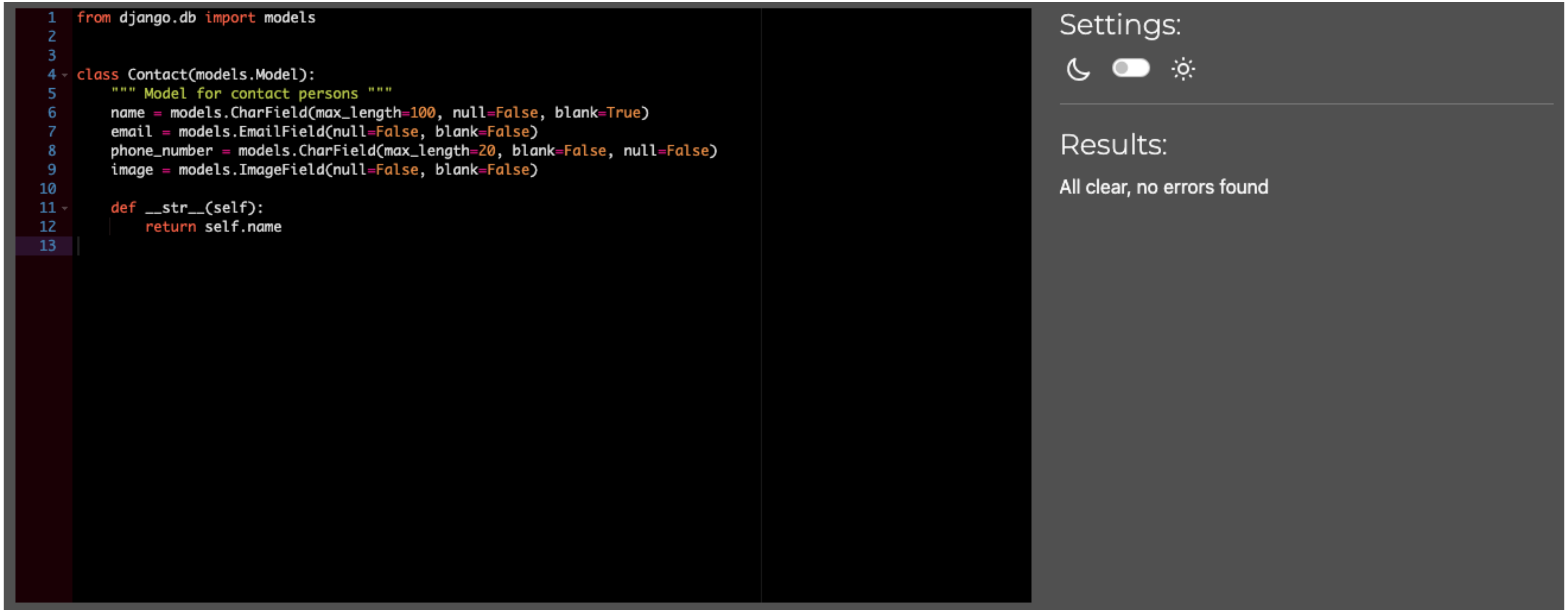
Settings:

Results:

All clear, no errors found

Python Validation

contact/models.py



```python
from django.db import models


class Contact(models.Model):
    """ Model for contact persons """
    name = models.CharField(max_length=100, null=False, blank=True)
    email = models.EmailField(null=False, blank=False)
    phone_number = models.CharField(max_length=20, blank=False, null=False)
    image = models.ImageField(null=False, blank=False)

    def __str__(self):
        return self.name
```

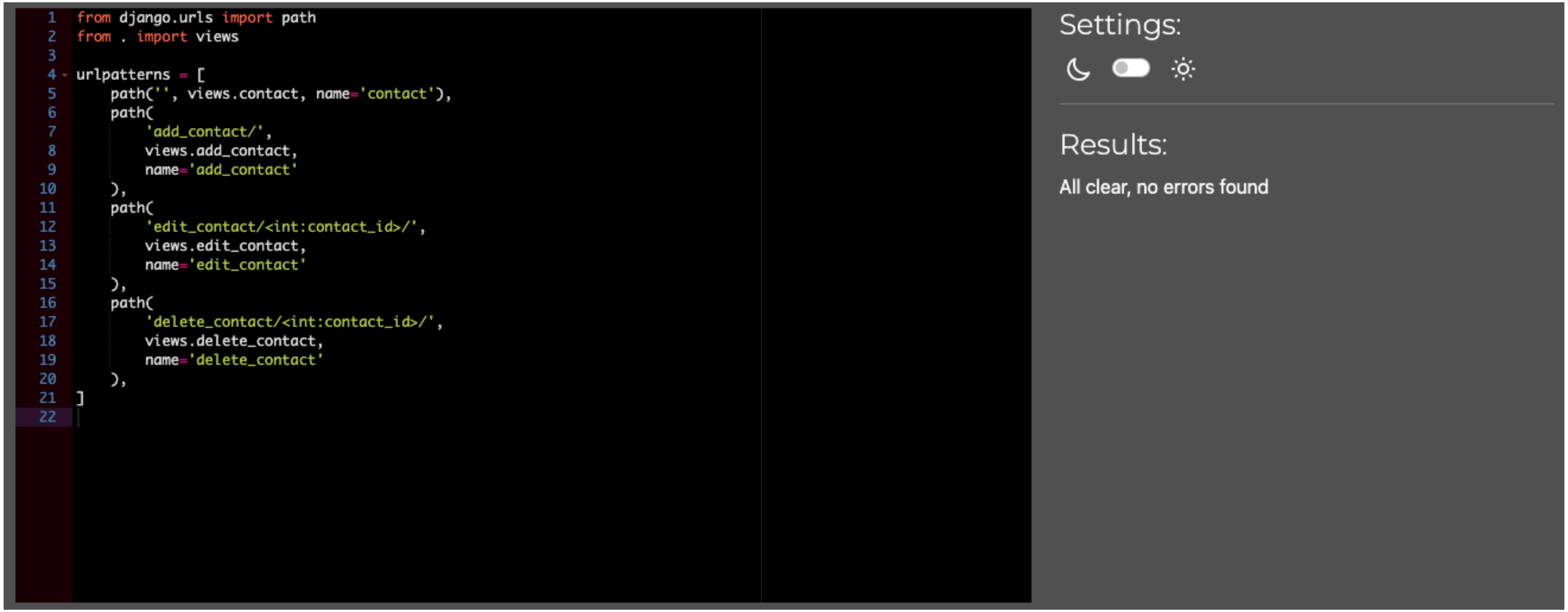
Settings:

Results:

All clear, no errors found

Python Validation

contact/urls.py



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

urlpatterns = [
    path('', views.contact, name='contact'),
    path(
        'add_contact/',
        views.add_contact,
        name='add_contact'
    ),
    path(
        'edit_contact/<int:contact_id>/',
        views.edit_contact,
        name='edit_contact'
    ),
    path(
        'delete_contact/<int:contact_id>/',
        views.delete_contact,
        name='delete_contact'
    ),
]
```

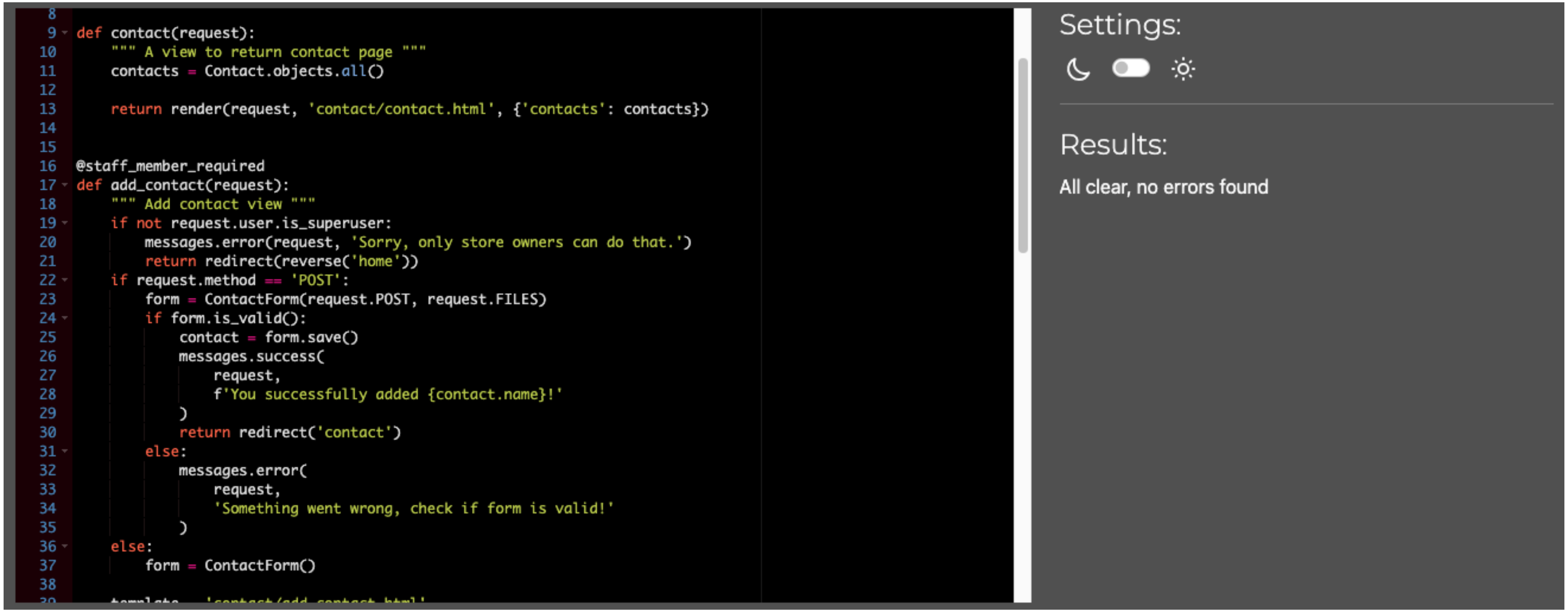
Settings:

Results:

All clear, no errors found

Python Validation

contact/views.py



```python

def contact(request):
    """ A view to return contact page """
    contacts = Contact.objects.all()

    return render(request, 'contact/contact.html', {'contacts': contacts})


@staff_member_required
def add_contact(request):
    """ Add contact view """
    if not request.user.is_superuser:
        messages.error(request, 'Sorry, only store owners can do that.')
        return redirect(reverse('home'))
    if request.method == 'POST':
        form = ContactForm(request.POST, request.FILES)
        if form.is_valid():
            contact = form.save()
            messages.success(
                request,
                f'You successfully added {contact.name}!'
            )
            return redirect('contact')
        else:
            messages.error(
                request,
                'Something went wrong, check if form is valid!'
            )
    else:
        form = ContactForm()

    template = 'contact/add_contact.html'
```

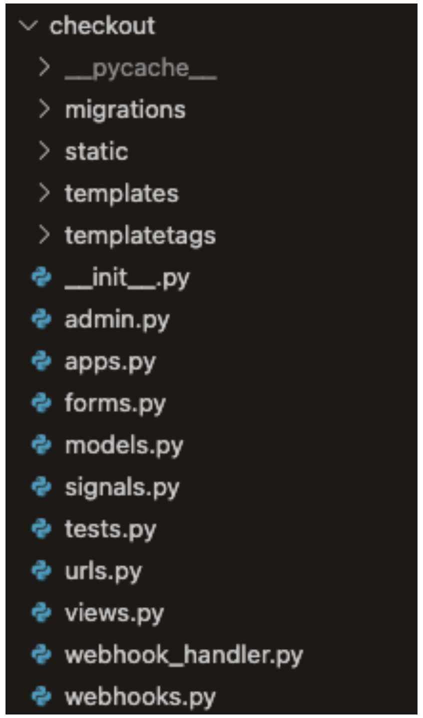
Settings:

Results:

All clear, no errors found

Python Validation

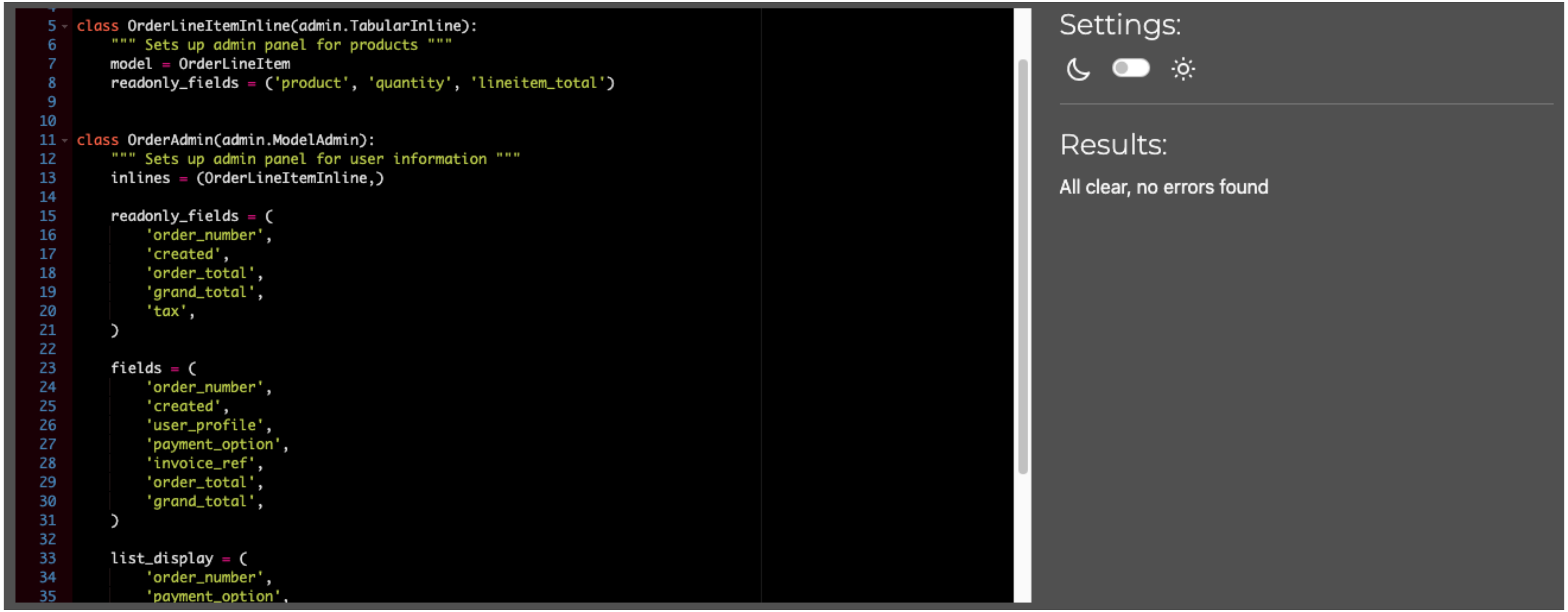
App: checkout



```
checkout
  __pycache__
  migrations
  static
  templates
  templatetags
  __init__.py
  admin.py
  apps.py
  forms.py
  models.py
  signals.py
  tests.py
  urls.py
  views.py
  webhook_handler.py
  webhooks.py
```

Python Validation

checkout/admin.py



```python
class OrderLineItemInline(admin.TabularInline):
    """ Sets up admin panel for products """
    model = OrderLineItem
    readonly_fields = ('product', 'quantity', 'lineitem_total')


class OrderAdmin(admin.ModelAdmin):
    """ Sets up admin panel for user information """
    inlines = (OrderLineItemInline,)

    readonly_fields = (
        'order_number',
        'created',
        'order_total',
        'grand_total',
        'tax',
    )

    fields = (
        'order_number',
        'created',
        'user_profile',
        'payment_option',
        'invoice_ref',
        'order_total',
        'grand_total',
    )

    list_display = (
        'order_number',
        'payment_option',
```

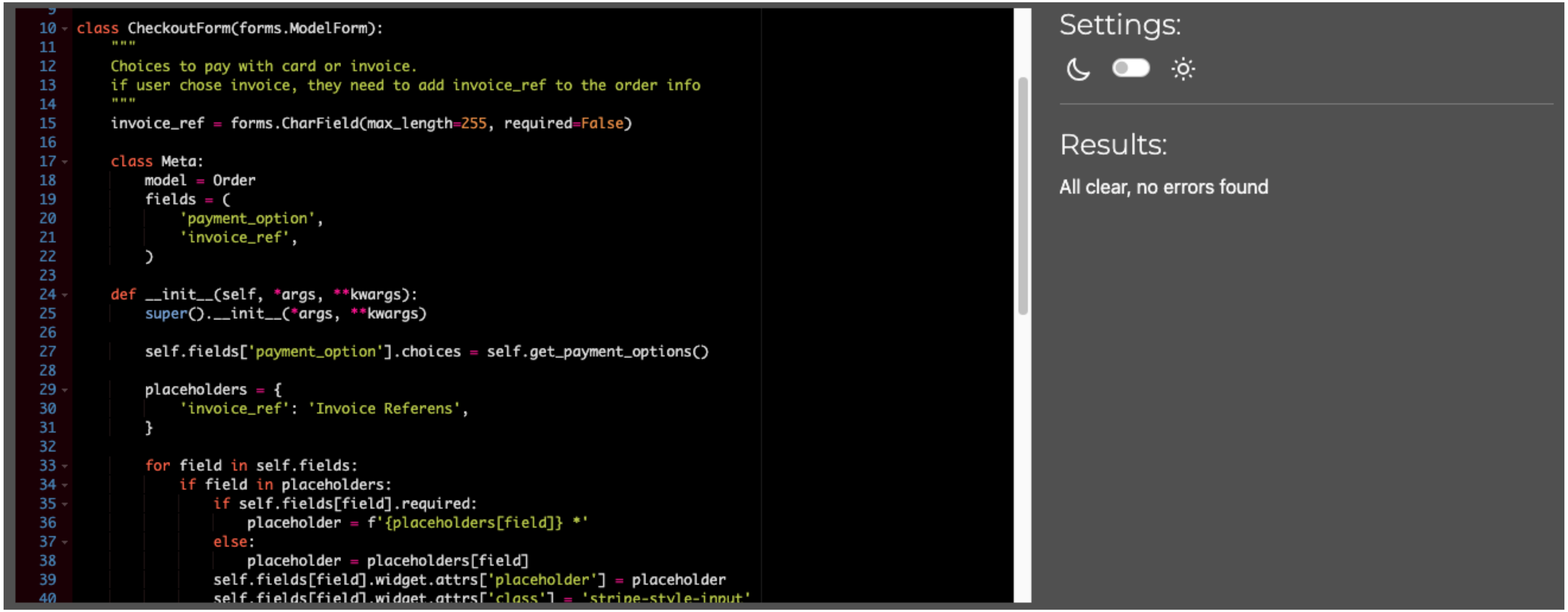
Settings:

Results:

All clear, no errors found

Python Validation

checkout/forms.py



```
class CheckoutForm(forms.ModelForm):
    """
    Choices to pay with card or invoice.
    if user chose invoice, they need to add invoice_ref to the order info
    """
    invoice_ref = forms.CharField(max_length=255, required=False)

    class Meta:
        model = Order
        fields = (
            'payment_option',
            'invoice_ref',
        )

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)

        self.fields['payment_option'].choices = self.get_payment_options()

        placeholders = {
            'invoice_ref': 'Invoice Referens',
        }

        for field in self.fields:
            if field in placeholders:
                if self.fields[field].required:
                    placeholder = f'{placeholders[field]} *'
                else:
                    placeholder = placeholders[field]
                self.fields[field].widget.attrs['placeholder'] = placeholder
                self.fields[field].widget.attrs['class'] = 'stripe-style-input'
```

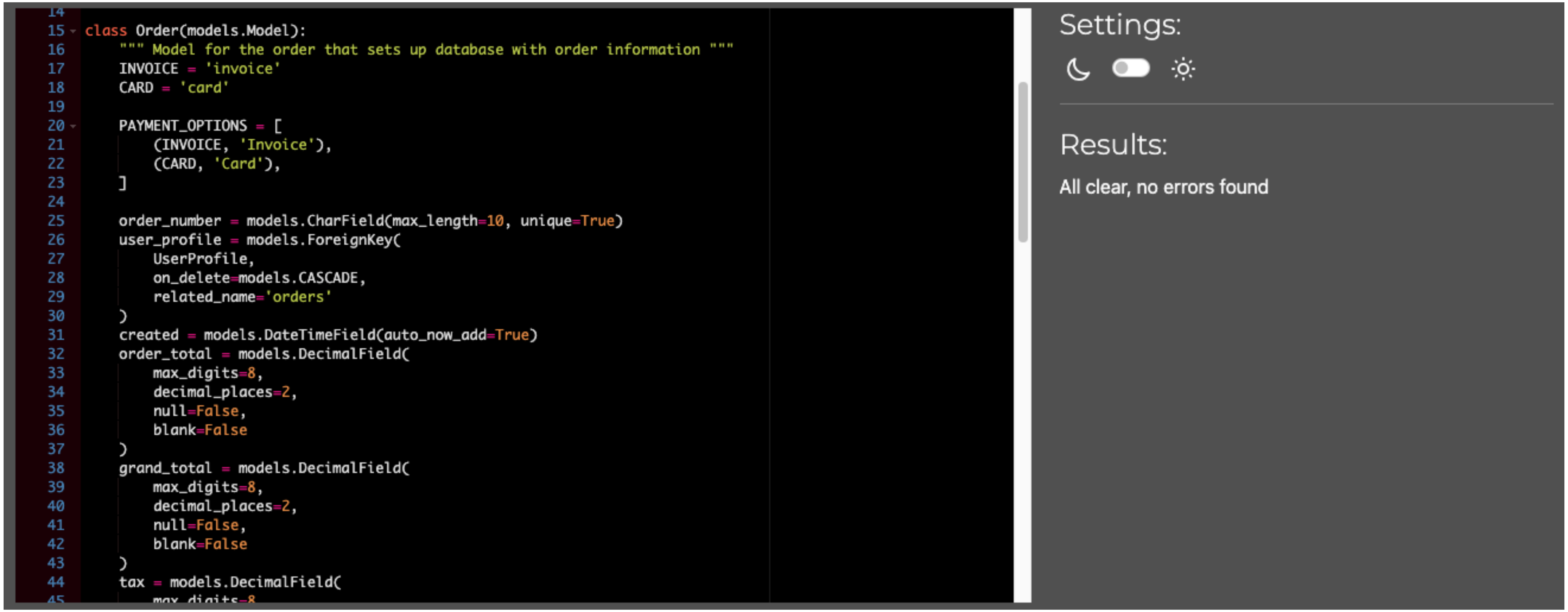
Settings:

Results:

All clear, no errors found

Python Validation

checkout/models.py



```
class Order(models.Model):
    """ Model for the order that sets up database with order information """
    INVOICE = 'invoice'
    CARD = 'card'

    PAYMENT_OPTIONS = [
        (INVOICE, 'Invoice'),
        (CARD, 'Card'),
    ]

    order_number = models.CharField(max_length=10, unique=True)
    user_profile = models.ForeignKey(
        UserProfile,
        on_delete=models.CASCADE,
        related_name='orders'
    )
    created = models.DateTimeField(auto_now_add=True)
    order_total = models.DecimalField(
        max_digits=8,
        decimal_places=2,
        null=False,
        blank=False
    )
    grand_total = models.DecimalField(
        max_digits=8,
        decimal_places=2,
        null=False,
        blank=False
    )
    tax = models.DecimalField(
        max_digits=8
```

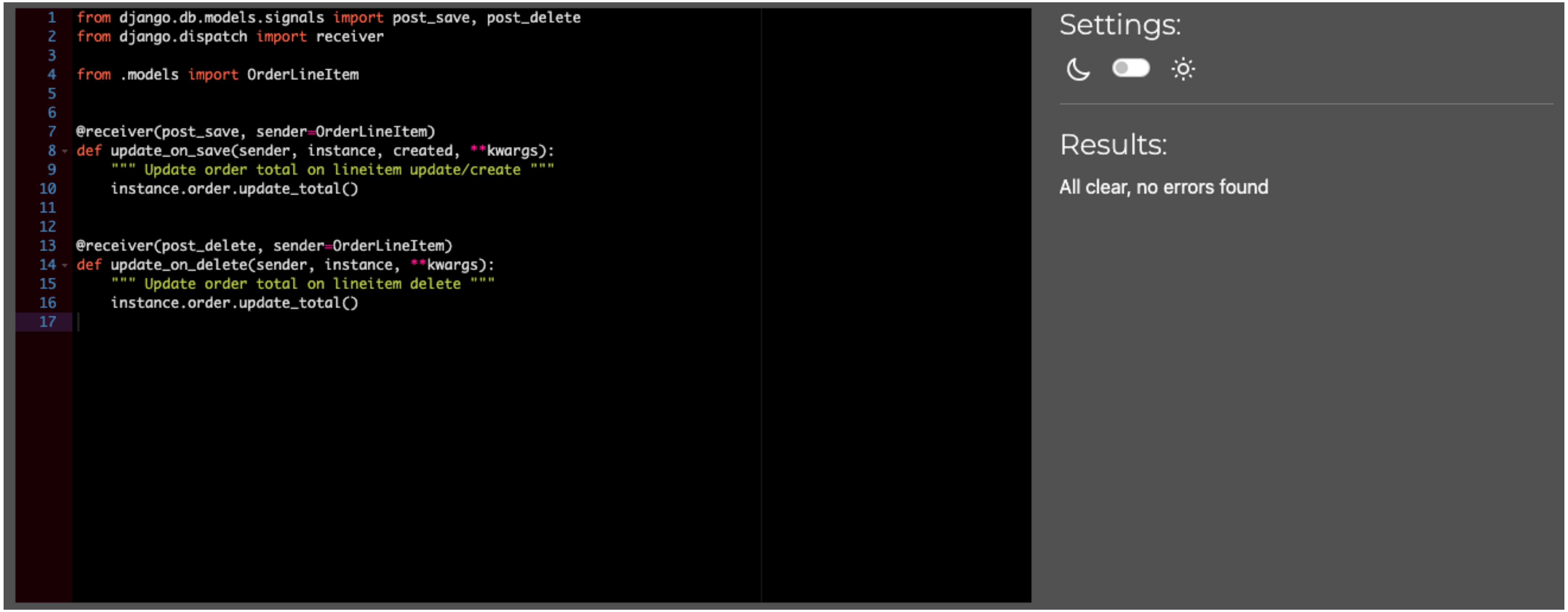
Settings:

Results:

All clear, no errors found

Python Validation

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()

```

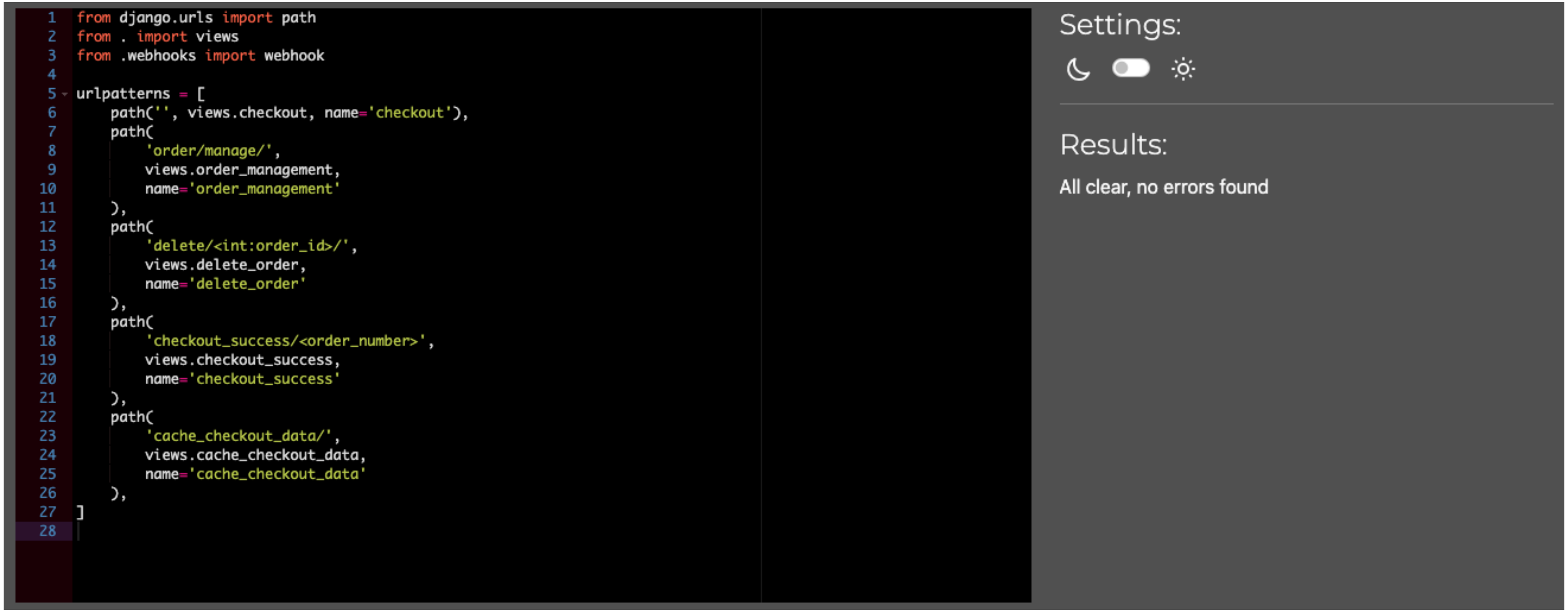
Settings:

Results:

All clear, no errors found

Python Validation

checkout/urls.py



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

urlpatterns = [
    path('', views.checkout, name='checkout'),
    path(
        'order/manage/',
        views.order_management,
        name='order_management'
    ),
    path(
        'delete/<int:order_id>/',
        views.delete_order,
        name='delete_order'
    ),
    path(
        'checkout_success/<order_number>',
        views.checkout_success,
        name='checkout_success'
    ),
    path(
        'cache_checkout_data/',
        views.cache_checkout_data,
        name='cache_checkout_data'
    ),
]
```

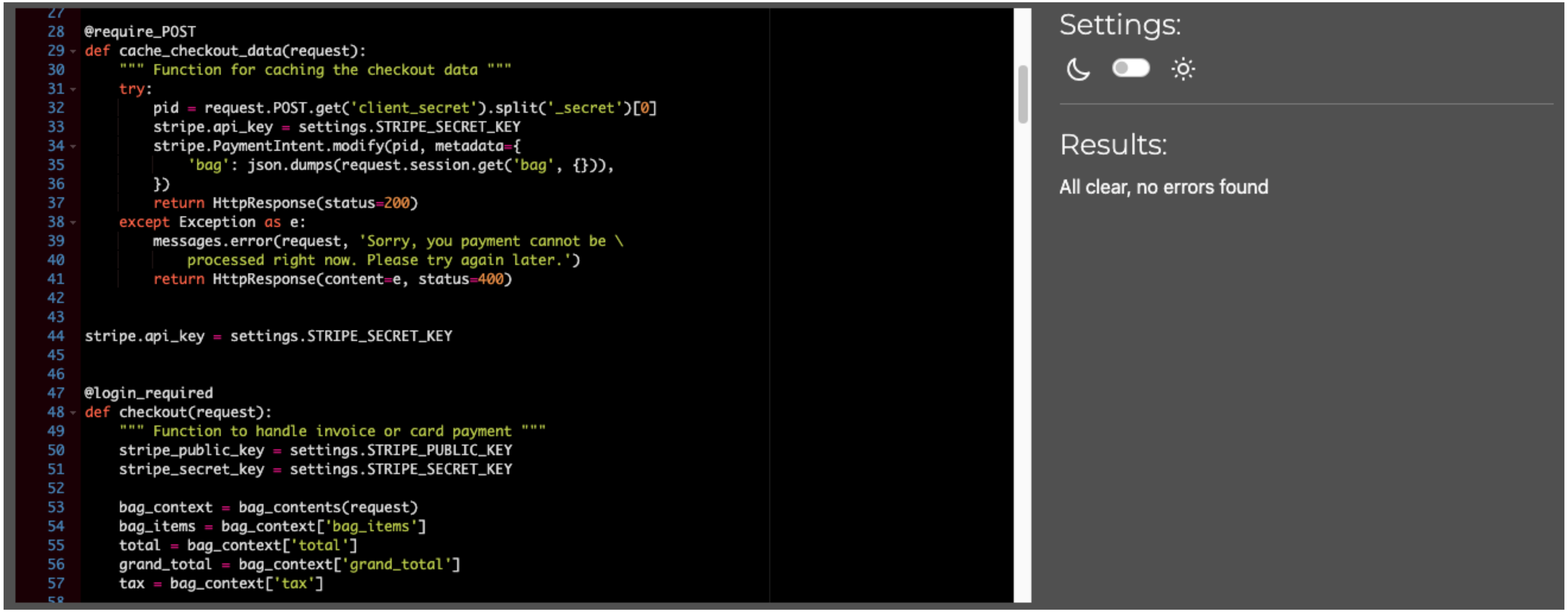
Settings:

Results:

All clear, no errors found

Python Validation

checkout/views.py



```python
@require_POST
def cache_checkout_data(request):
    """ Function for caching the checkout data """
    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', {})),
        })
        return HttpResponse(status=200)
    except Exception as e:
        messages.error(request, 'Sorry, you payment cannot be \
            processed right now. Please try again later.')
        return HttpResponse(content=e, status=400)


stripe.api_key = settings.STRIPE_SECRET_KEY


@login_required
def checkout(request):
    """ Function to handle invoice or card payment """
    stripe_public_key = settings.STRIPE_PUBLIC_KEY
    stripe_secret_key = settings.STRIPE_SECRET_KEY

    bag_context = bag_contents(request)
    bag_items = bag_context['bag_items']
    total = bag_context['total']
    grand_total = bag_context['grand_total']
    tax = bag_context['tax']
```

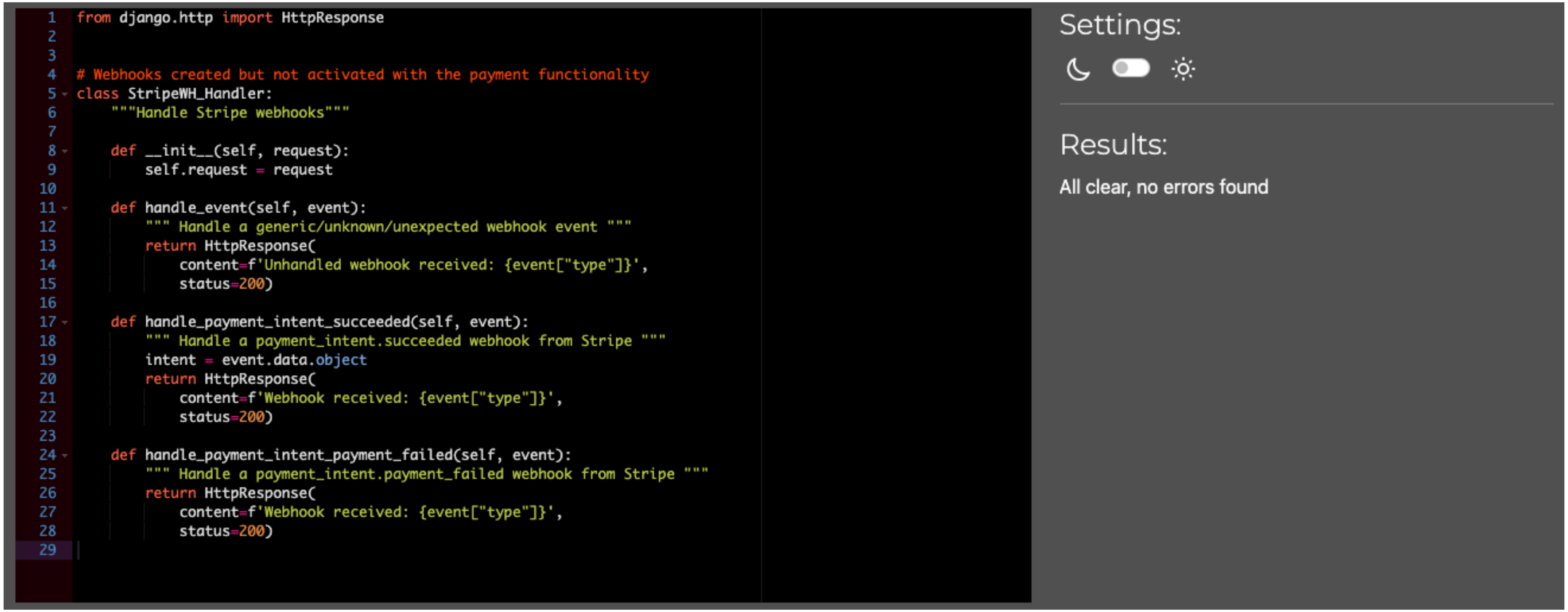
Settings:

Results:

All clear, no errors found

Python Validation

checkout/webhook_handler.py



```
from django.http import HttpResponse


# Webhooks created but not activated with the payment functionality
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 a payment_intent.succeeded webhook from Stripe """
        intent = event.data.object
        return HttpResponse(
            content=f'Webhook received: {event["type"]}',
            status=200)

    def handle_payment_intent_payment_failed(self, event):
        """ Handle a payment_intent.payment_failed webhook from Stripe """
        return HttpResponse(
            content=f'Webhook received: {event["type"]}',
            status=200)
```

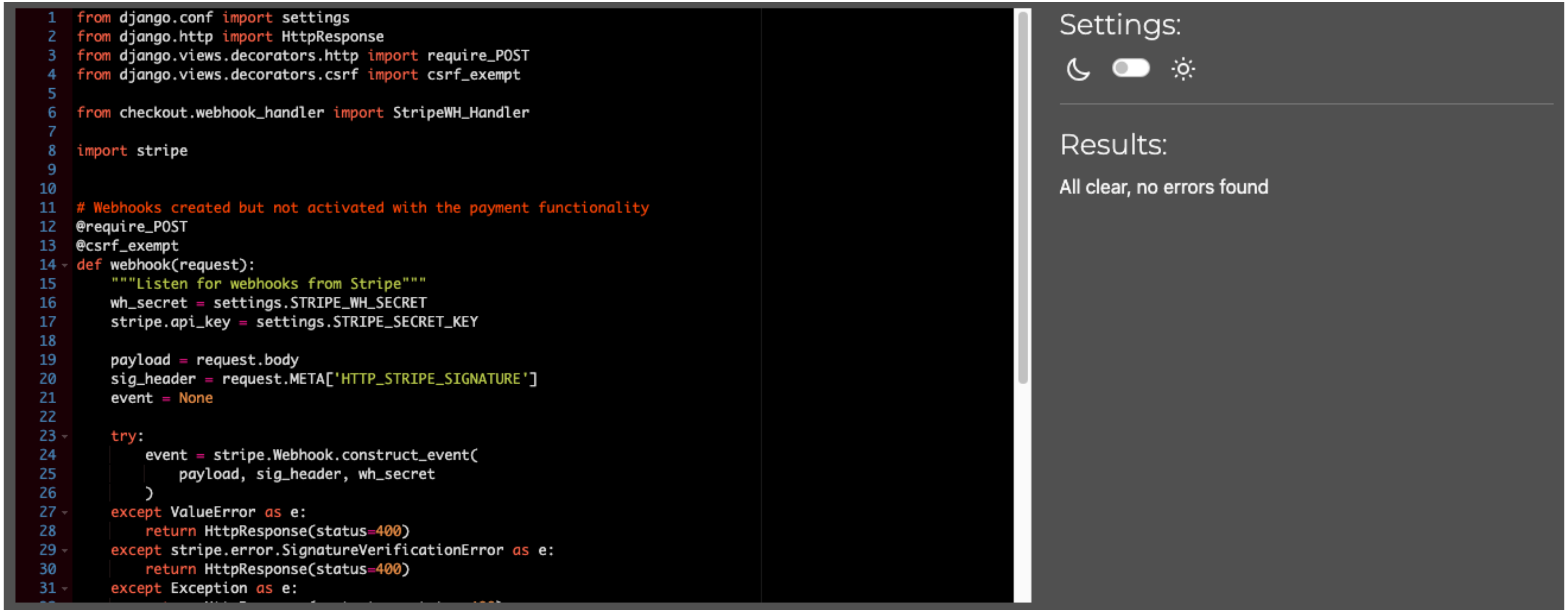
Settings:

Results:

All clear, no errors found

Python Validation

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


# Webhooks created but not activated with the payment functionality
@require_POST
@csrf_exempt
def webhook(request):
    """Listen for webhooks from Stripe"""
    wh_secret = settings.STRIPE_WH_SECRET
    stripe.api_key = settings.STRIPE_SECRET_KEY

    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:
        return HttpResponse(status=400)
    except stripe.error.SignatureVerificationError as e:
        return HttpResponse(status=400)
    except Exception as e:
```

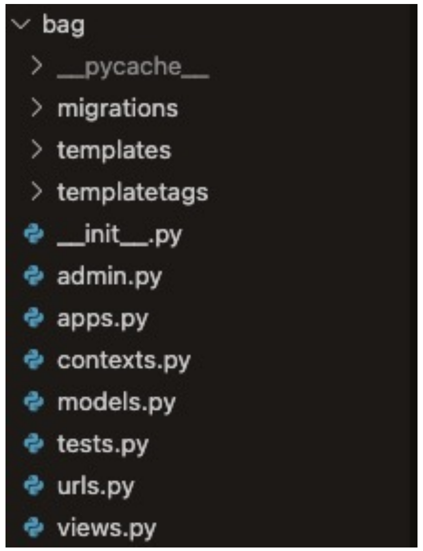
Settings:

Results:

All clear, no errors found

Python Validation

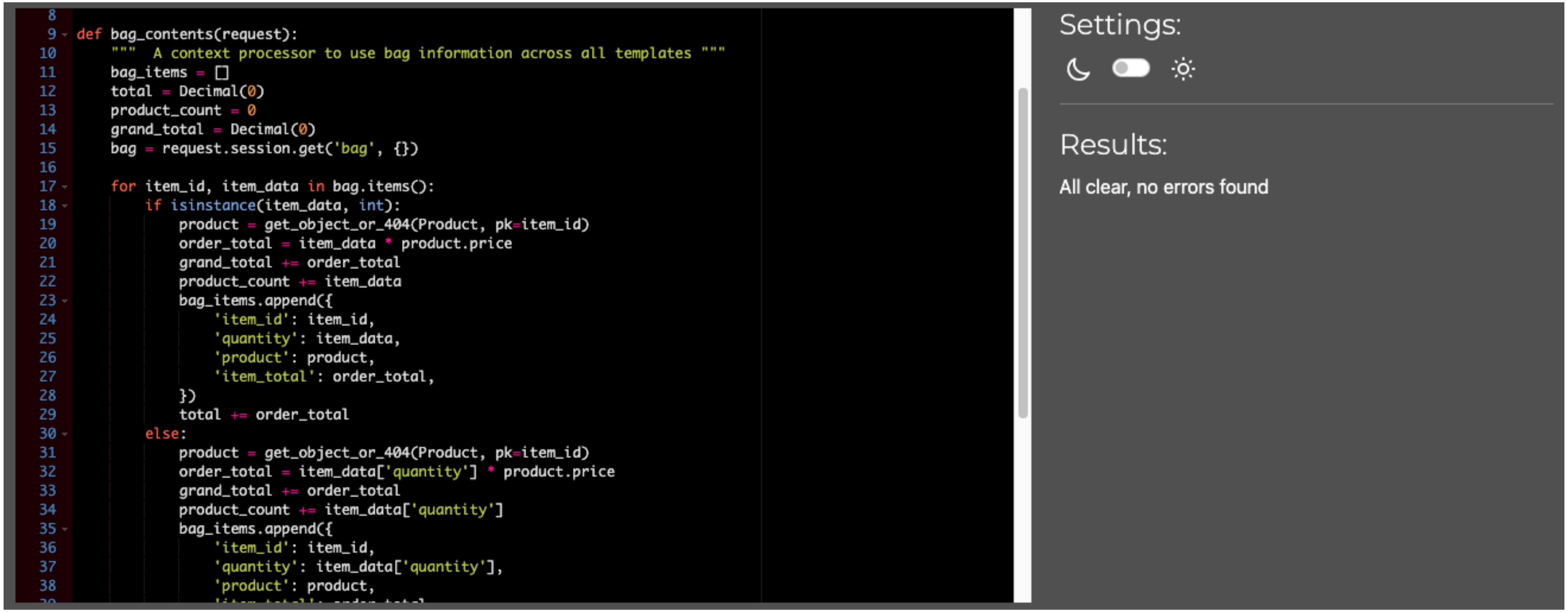
App: bag



```
bag
  __pycache__
  migrations
  templates
  templatetags
  __init__.py
  admin.py
  apps.py
  contexts.py
  models.py
  tests.py
  urls.py
  views.py
```

Python Validation

bag/contexts.py



```python

def bag_contents(request):
    """  A context processor to use bag information across all templates """
    bag_items = []
    total = Decimal(0)
    product_count = 0
    grand_total = Decimal(0)
    bag = request.session.get('bag', {})

    for item_id, item_data in bag.items():
        if isinstance(item_data, int):
            product = get_object_or_404(Product, pk=item_id)
            order_total = item_data * product.price
            grand_total += order_total
            product_count += item_data
            bag_items.append({
                'item_id': item_id,
                'quantity': item_data,
                'product': product,
                'item_total': order_total,
            })
            total += order_total
        else:
            product = get_object_or_404(Product, pk=item_id)
            order_total = item_data['quantity'] * product.price
            grand_total += order_total
            product_count += item_data['quantity']
            bag_items.append({
                'item_id': item_id,
                'quantity': item_data['quantity'],
                'product': product,
```

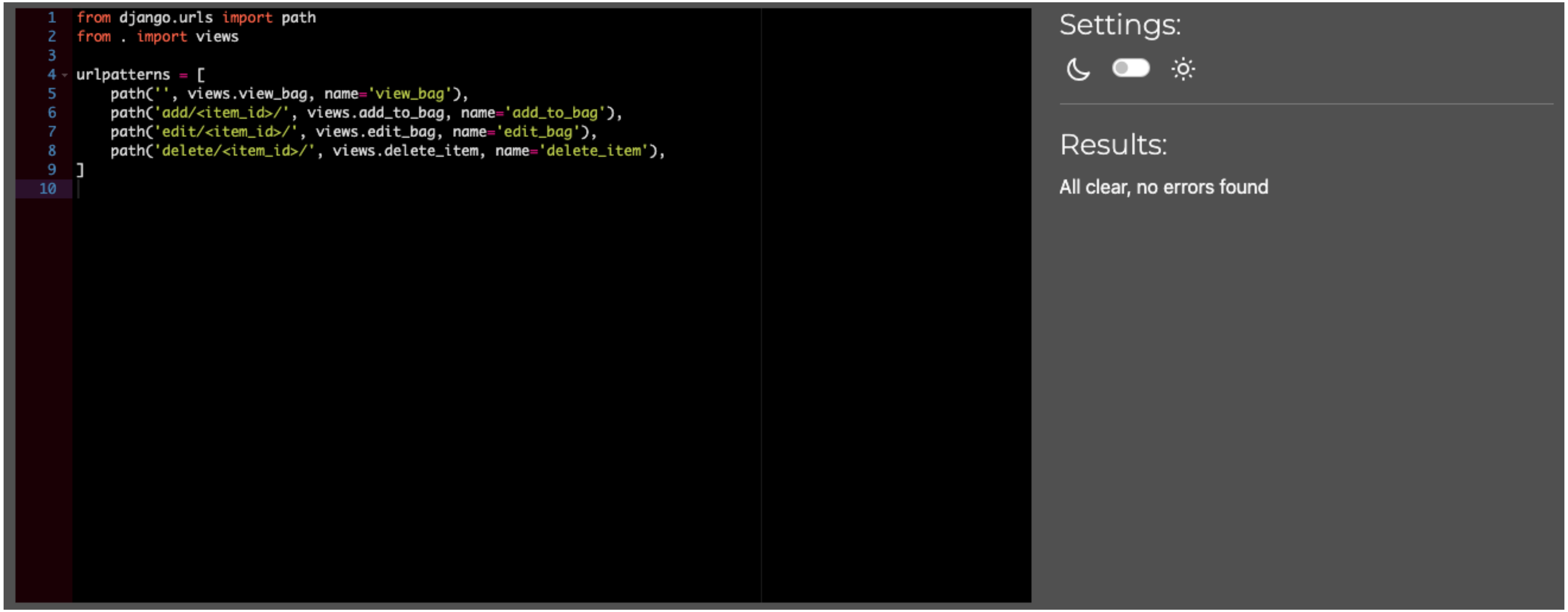
Settings:

Results:

All clear, no errors found

Python Validation

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('edit/<item_id>/', views.edit_bag, name='edit_bag'),
    path('delete/<item_id>/', views.delete_item, name='delete_item'),
]
```

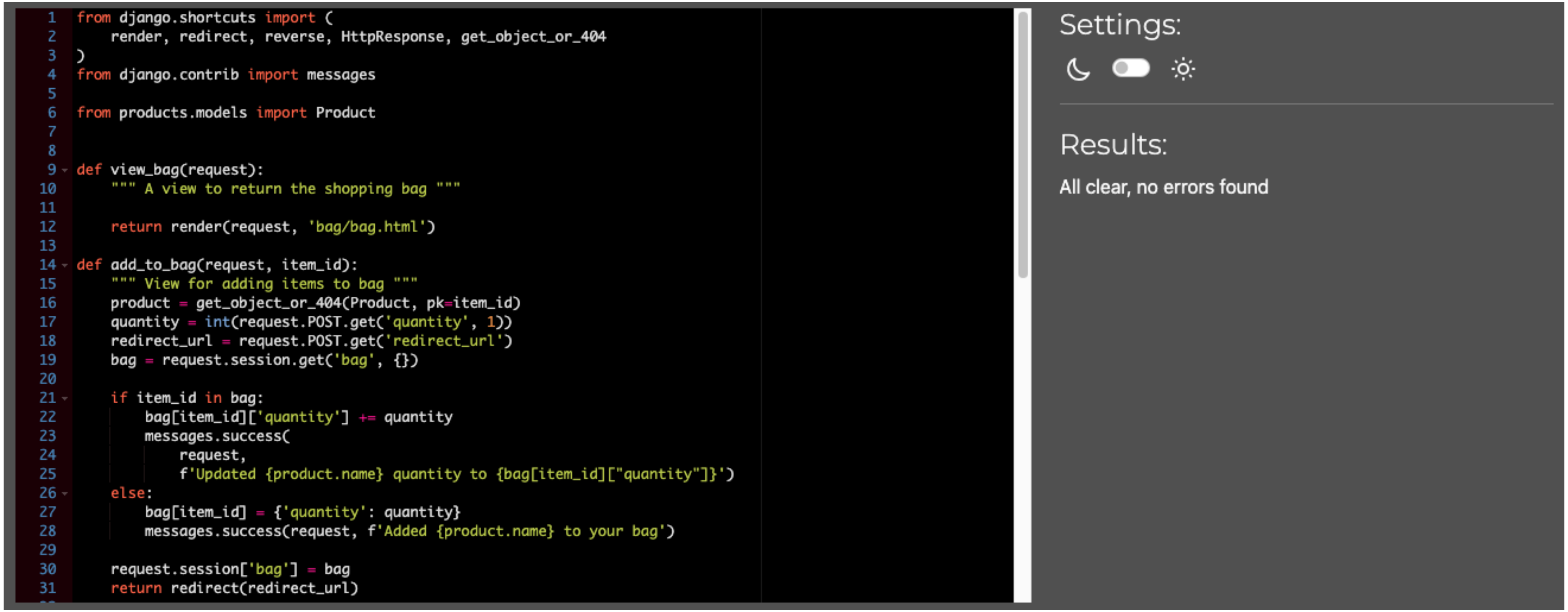
Settings:

Results:

All clear, no errors found

Python Validation

bag/views.py



```python
from django.shortcuts import (
    render, redirect, reverse, HttpResponse, get_object_or_404
)
from django.contrib import messages

from products.models import Product


def view_bag(request):
    """ A view to return the shopping bag """

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

def add_to_bag(request, item_id):
    """ View for adding items to bag """
    product = get_object_or_404(Product, pk=item_id)
    quantity = int(request.POST.get('quantity', 1))
    redirect_url = request.POST.get('redirect_url')
    bag = request.session.get('bag', {})

    if item_id in bag:
        bag[item_id]['quantity'] += quantity
        messages.success(
            request,
            f'Updated {product.name} quantity to {bag[item_id]["quantity"]}')
    else:
        bag[item_id] = {'quantity': quantity}
        messages.success(request, f'Added {product.name} to your bag')

    request.session['bag'] = bag
    return redirect(redirect_url)
```

Settings:

Results:

All clear, no errors found