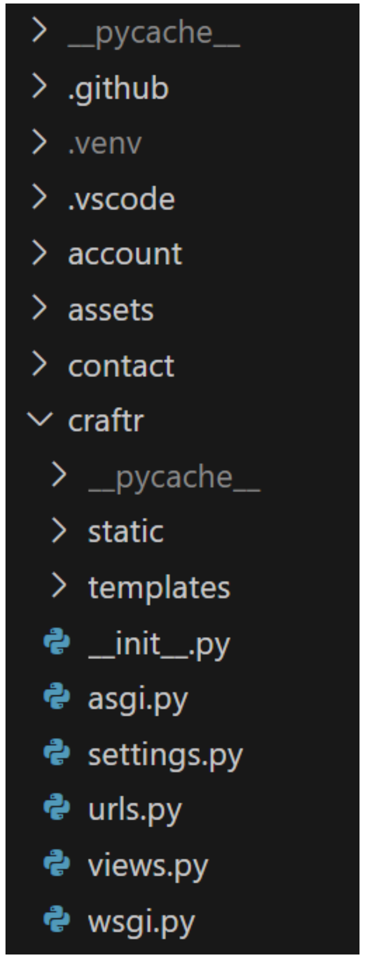
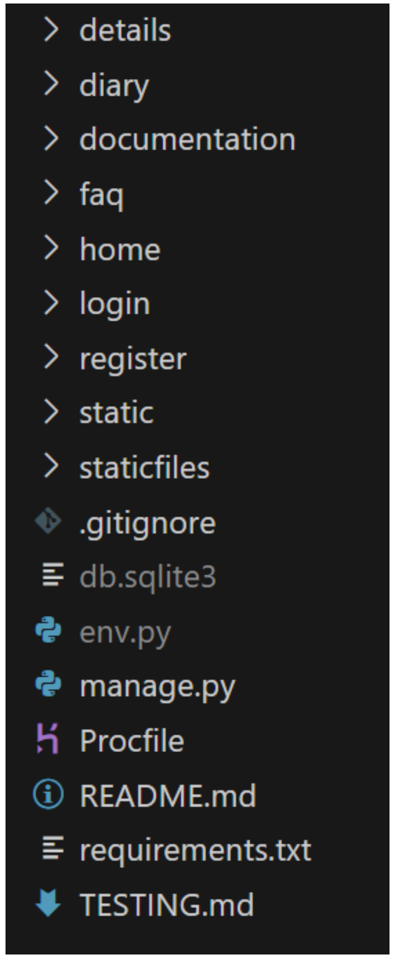
Project - Craftr



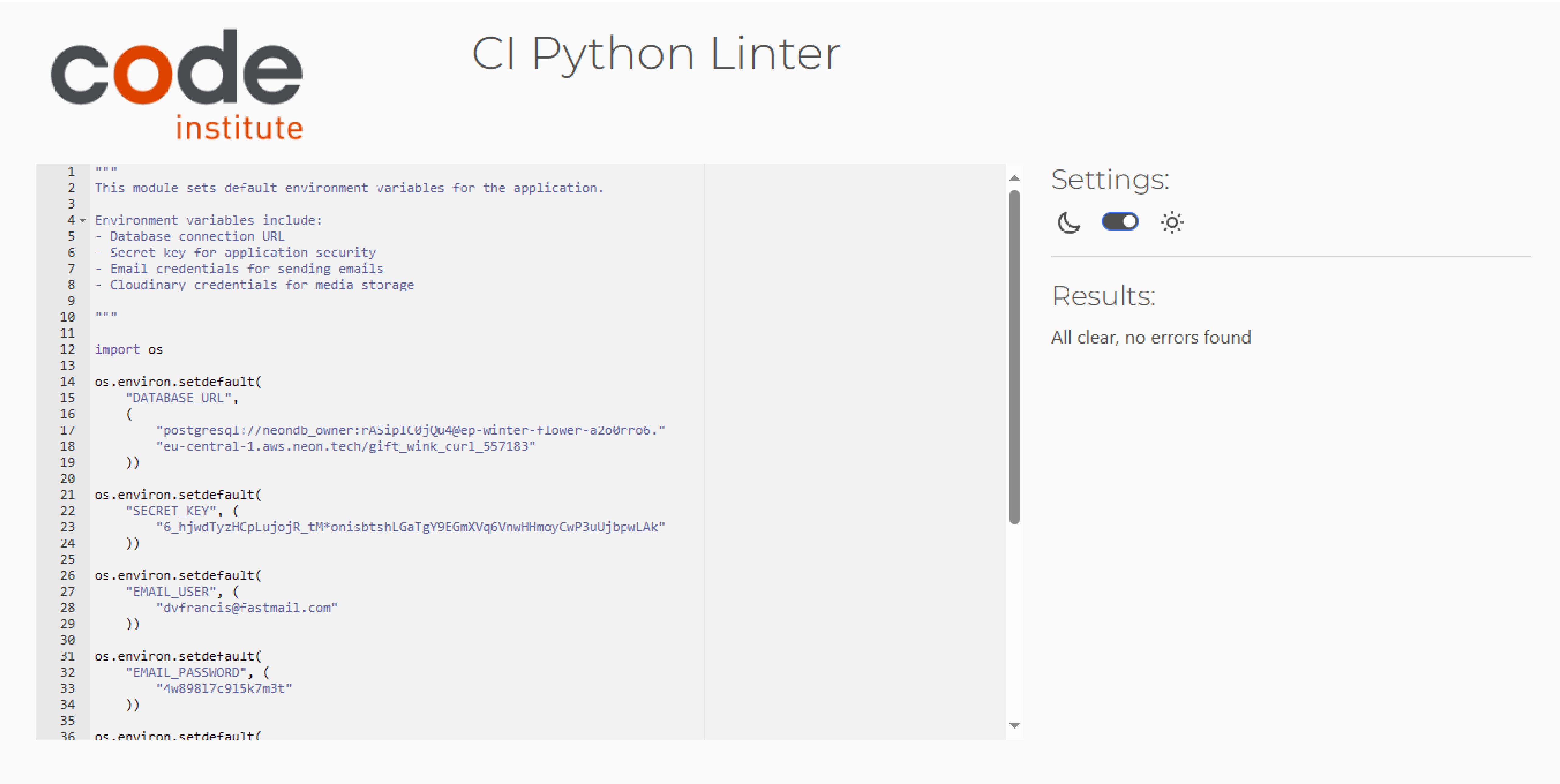
```
> __pycache__
> .github
> .venv
> .vscode
> account
> assets
> contact
v craftr
  > __pycache__
  > static
  > templates
    __init__.py
    asgi.py
    settings.py
    urls.py
    views.py
    wsgi.py
```



```
> details
> diary
> documentation
> faq
> home
> login
> register
> static
> staticfiles
  .gitignore
  db.sqlite3
  env.py
  manage.py
  Procfile
  README.md
  requirements.txt
  TESTING.md
```

Python files created / amended were *env.py*, *views.py*, *urls.py*, and *settings.py*

env.py



code institute

CI Python Linter

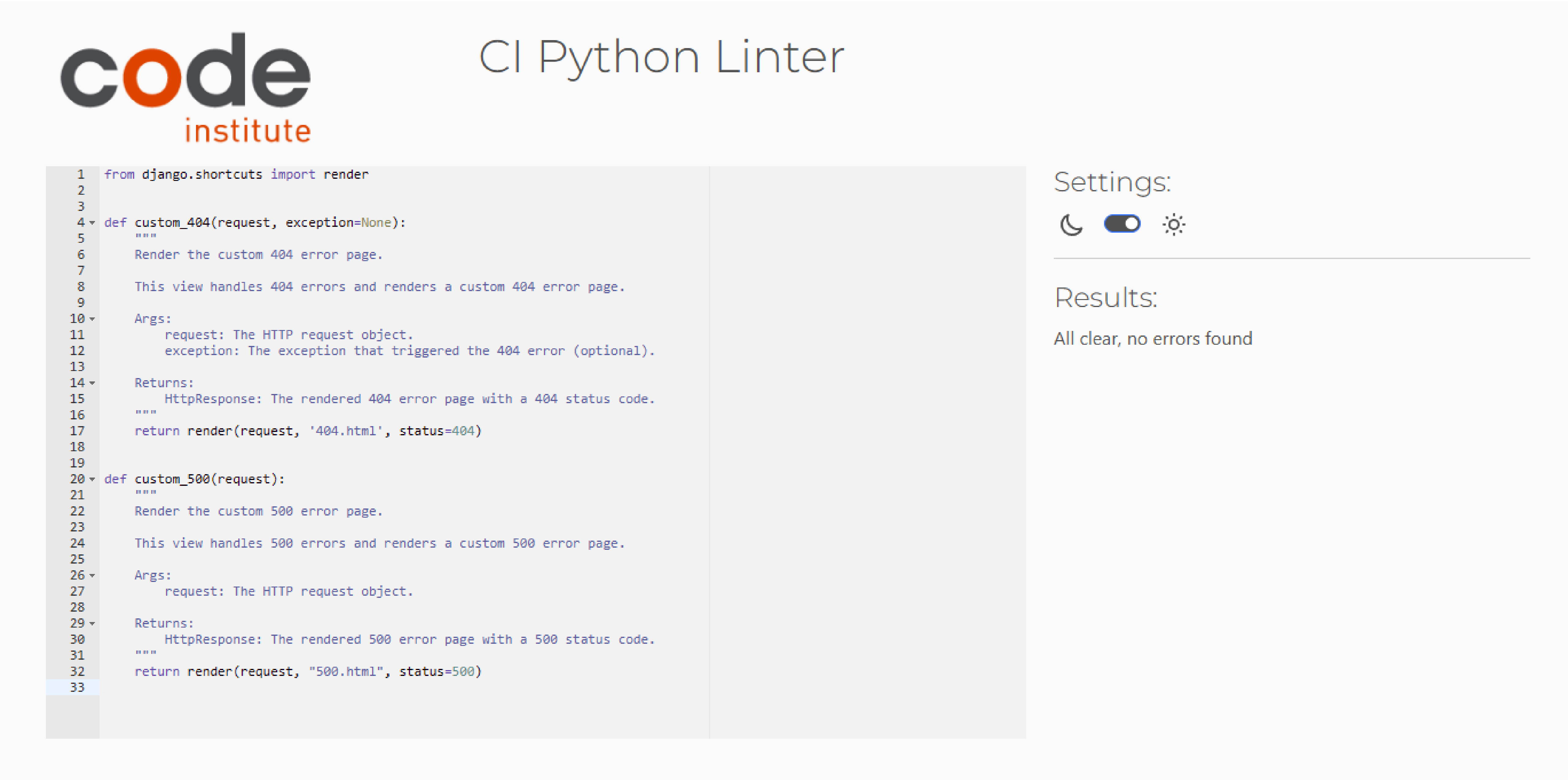
```
"""
This module sets default environment variables for the application.

Environment variables include:
- Database connection URL
- Secret key for application security
- Email credentials for sending emails
- Cloudinary credentials for media storage

"""

import os

os.environ.setdefault(
    "DATABASE_URL",
    (
        "postgresql://neondb_owner:rASipIC0jQu4@ep-winter-flower-a2o0rro6."
        "eu-central-1.aws.neon.tech/gift_wink_curl_557183"
    ))

os.environ.setdefault(
    "SECRET_KEY", (
        "6_hjwdTyzHCpLujojR_tM*onisbtshLGaTgY9EGmXVq6VnwHHmoyCwP3uUjbpwLAk"
    ))

os.environ.setdefault(
    "EMAIL_USER", (
        "dvfrancis@fastmail.com"
    ))

os.environ.setdefault(
    "EMAIL_PASSWORD", (
        "4w89817c915k7m3t"
    ))

os.environ.setdefault(
```

Settings:

Results:

All clear, no errors found

views.py



CI Python Linter

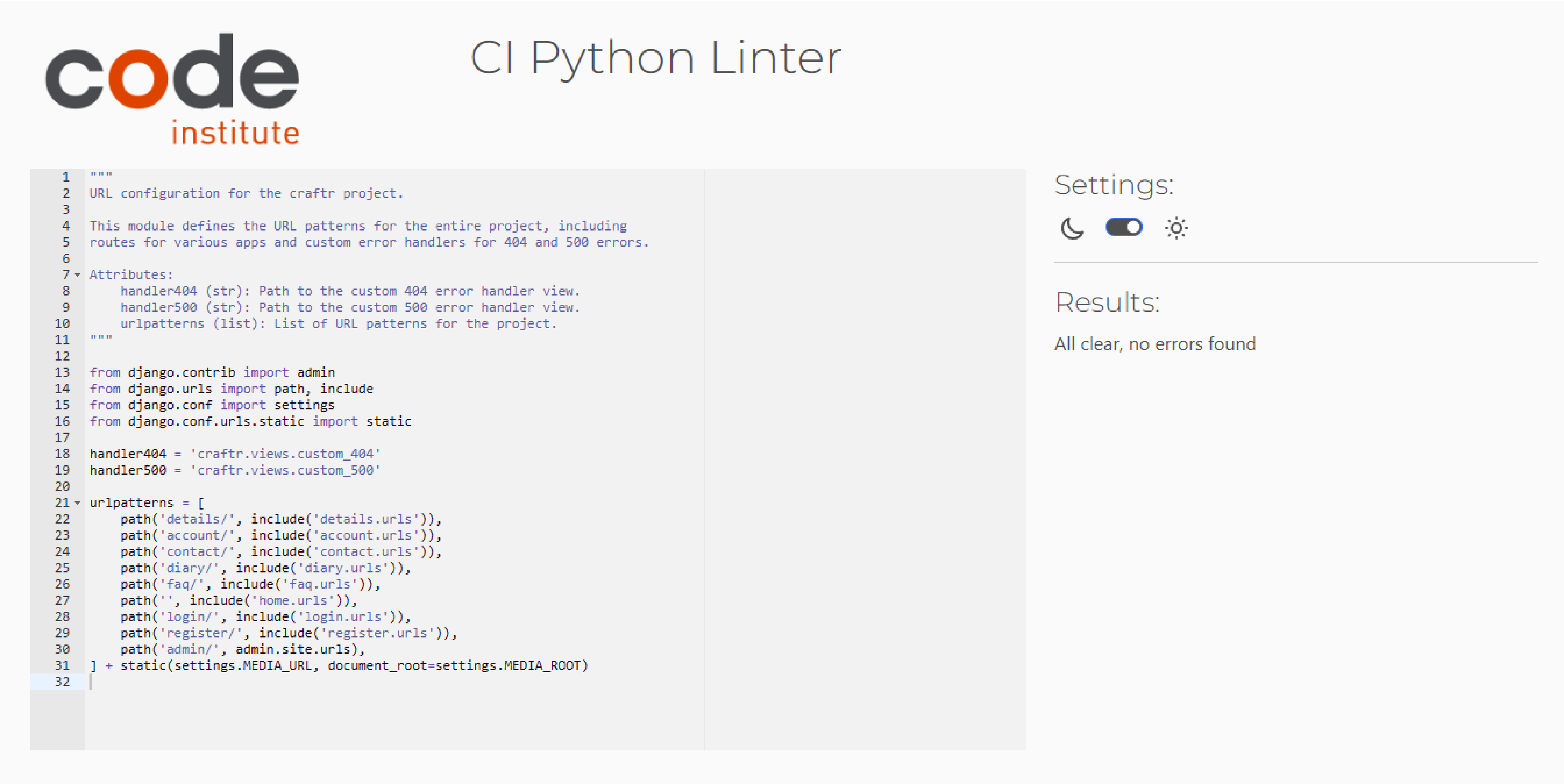
```
from django.shortcuts import render


def custom_404(request, exception=None):
    """
    Render the custom 404 error page.

    This view handles 404 errors and renders a custom 404 error page.

    Args:
        request: The HTTP request object.
        exception: The exception that triggered the 404 error (optional).

    Returns:
        HttpResponse: The rendered 404 error page with a 404 status code.
    """
    return render(request, '404.html', status=404)


def custom_500(request):
    """
    Render the custom 500 error page.

    This view handles 500 errors and renders a custom 500 error page.

    Args:
        request: The HTTP request object.

    Returns:
        HttpResponse: The rendered 500 error page with a 500 status code.
    """
    return render(request, "500.html", status=500)
```

Settings:

Results:

All clear, no errors found

urls.py



CI Python Linter

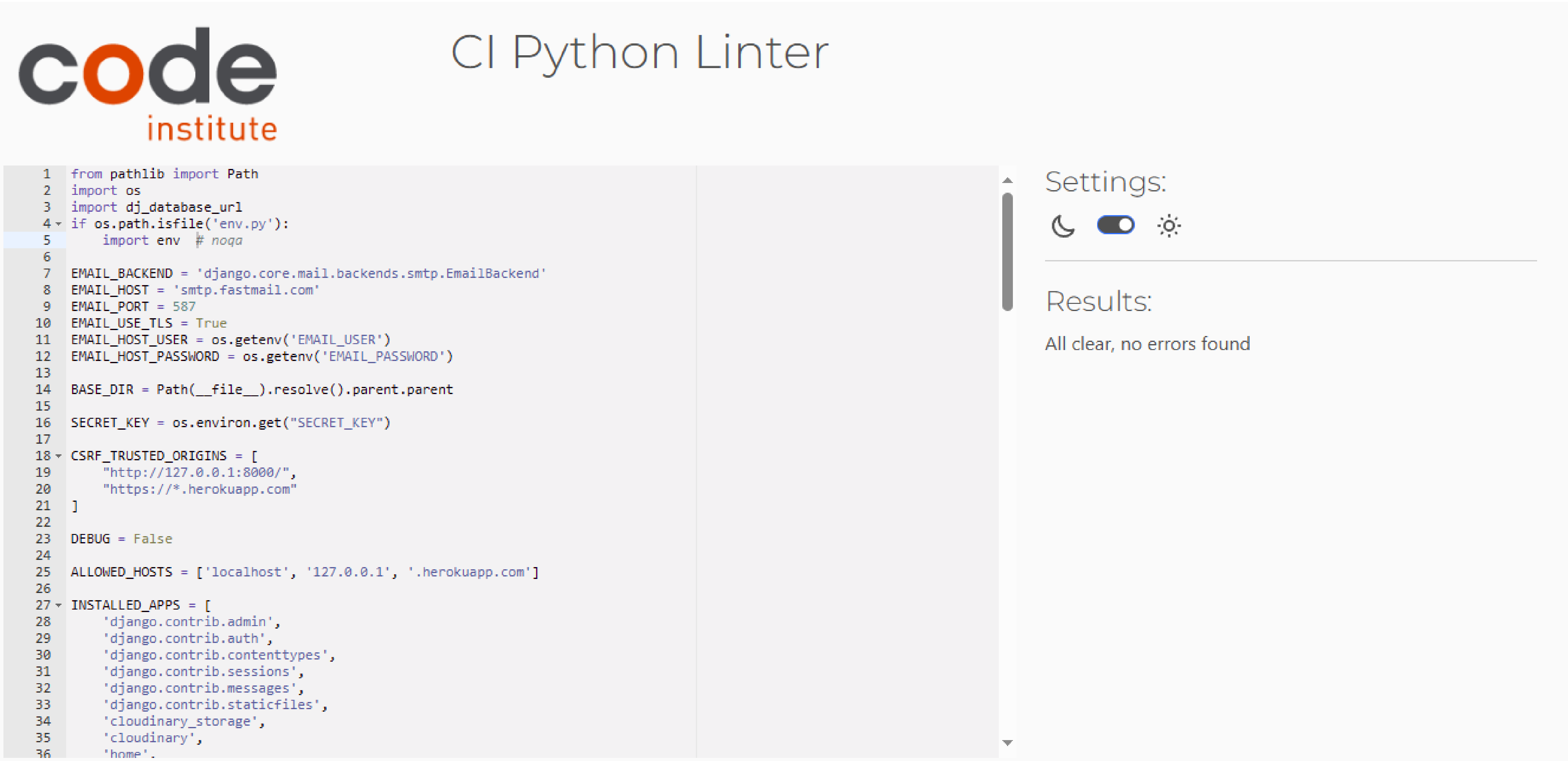
```
"""
URL configuration for the craftr project.

This module defines the URL patterns for the entire project, including
routes for various apps and custom error handlers for 404 and 500 errors.

Attributes:
    handler404 (str): Path to the custom 404 error handler view.
    handler500 (str): Path to the custom 500 error handler view.
    urlpatterns (list): List of URL patterns for the project.
"""

from django.contrib import admin
from django.urls import path, include
from django.conf import settings
from django.conf.urls.static import static

handler404 = 'craftr.views.custom_404'
handler500 = 'craftr.views.custom_500'

urlpatterns = [
    path('details/', include('details.urls')),
    path('account/', include('account.urls')),
    path('contact/', include('contact.urls')),
    path('diary/', include('diary.urls')),
    path('faq/', include('faq.urls')),
    path('', include('home.urls')),
    path('login/', include('login.urls')),
    path('register/', include('register.urls')),
    path('admin/', admin.site.urls),
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Settings:

Results:

All clear, no errors found

settings.py



CI Python Linter

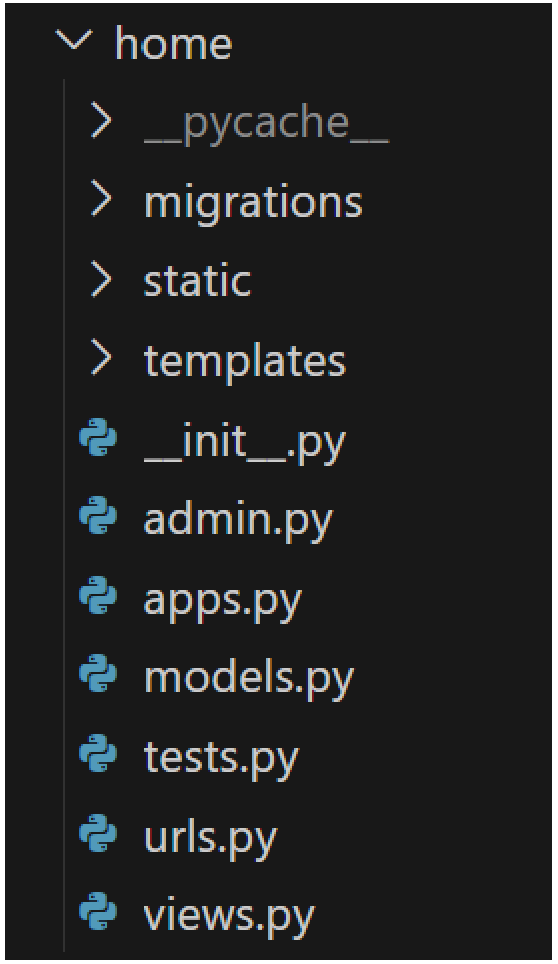
```
from pathlib import Path
import os
import dj_database_url
if os.path.isfile('env.py'):
    import env  # noqa

EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = 'smtp.fastmail.com'
EMAIL_PORT = 587
EMAIL_USE_TLS = True
EMAIL_HOST_USER = os.getenv('EMAIL_USER')
EMAIL_HOST_PASSWORD = os.getenv('EMAIL_PASSWORD')

BASE_DIR = Path(__file__).resolve().parent.parent

SECRET_KEY = os.environ.get("SECRET_KEY")

CSRF_TRUSTED_ORIGINS = [
    "http://127.0.0.1:8000/",
    "https://*.herokuapp.com"
]

DEBUG = False

ALLOWED_HOSTS = ['localhost', '127.0.0.1', '.herokuapp.com']

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'cloudinary_storage',
    'cloudinary',
    'home',
```

Settings:

Results:

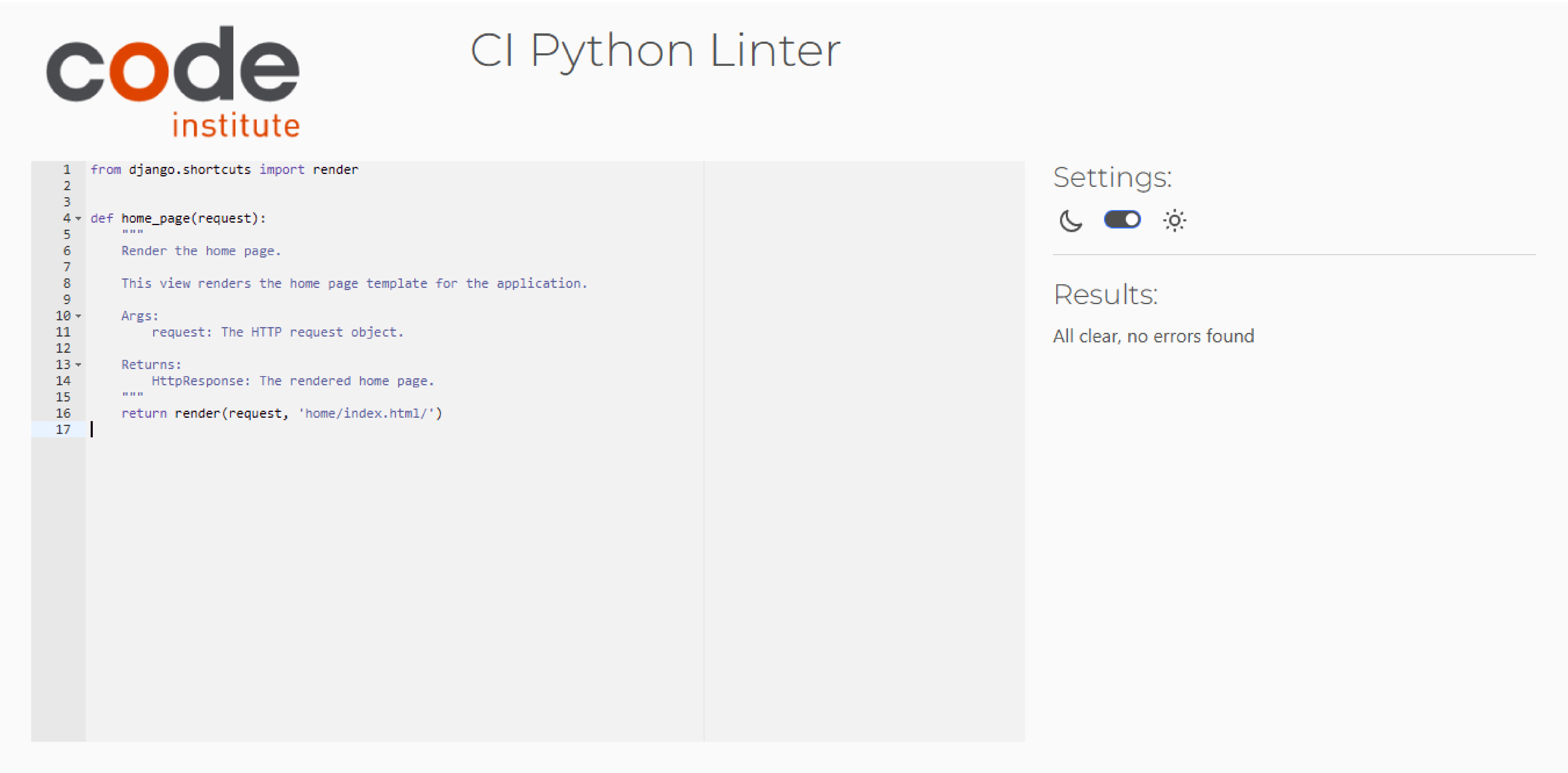
All clear, no errors found

App - Home


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

Python files created / amended were *views.py, urls.py, and apps.py*

views.py



CI Python Linter

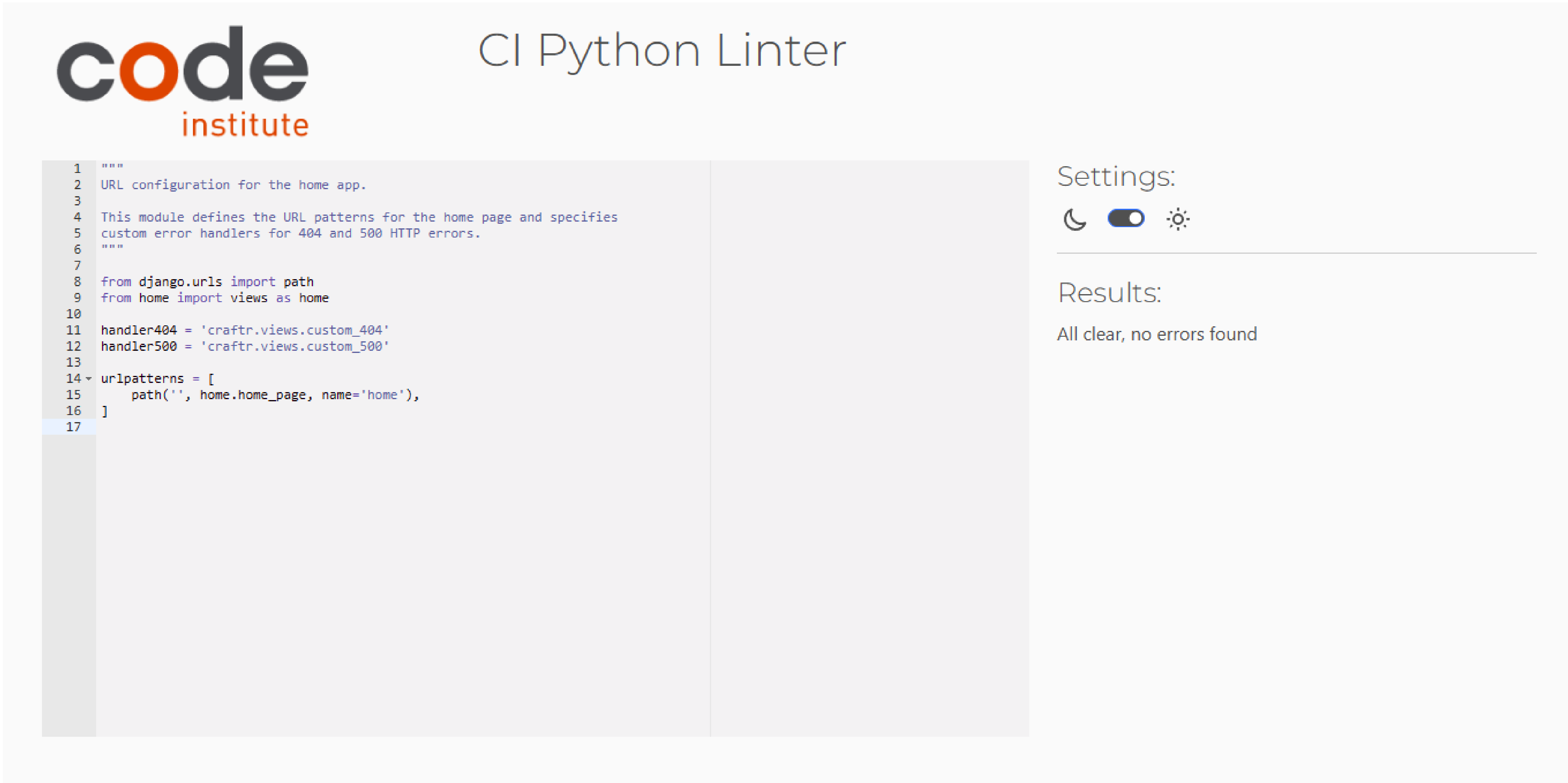
```python
from django.shortcuts import render


def home_page(request):
    """
    Render the home page.

    This view renders the home page template for the application.

    Args:
        request: The HTTP request object.

    Returns:
        HttpResponse: The rendered home page.
    """
    return render(request, 'home/index.html/')
```

Settings:

Results:

All clear, no errors found

urls.py

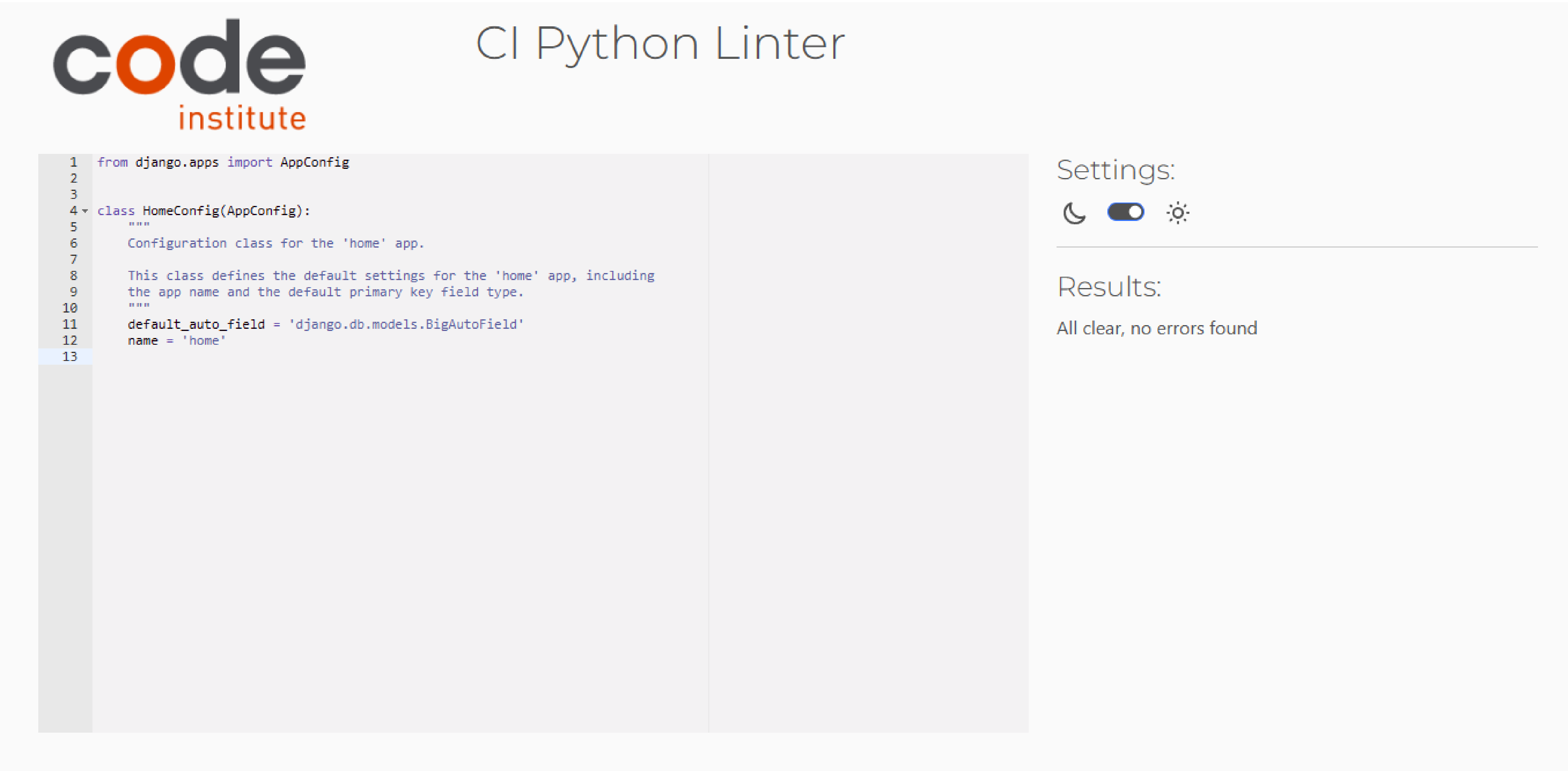


```
"""
URL configuration for the home app.

This module defines the URL patterns for the home page and specifies
custom error handlers for 404 and 500 HTTP errors.
"""

from django.urls import path
from home import views as home

handler404 = 'craftr.views.custom_404'
handler500 = 'craftr.views.custom_500'

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

Results:

All clear, no errors found

apps.py



CI Python Linter

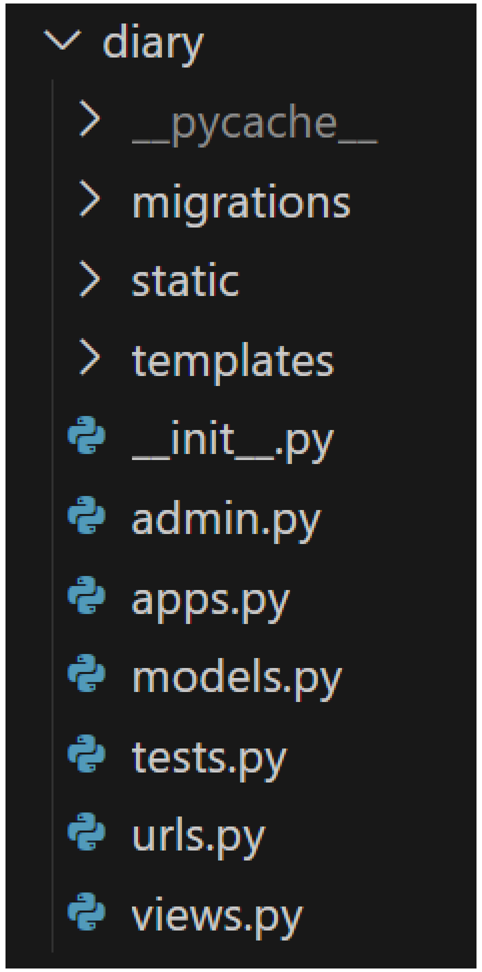
```
from django.apps import AppConfig


class HomeConfig(AppConfig):
    """
    Configuration class for the 'home' app.

    This class defines the default settings for the 'home' app, including
    the app name and the default primary key field type.
    """
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'home'
```

Settings:

Results:

All clear, no errors found

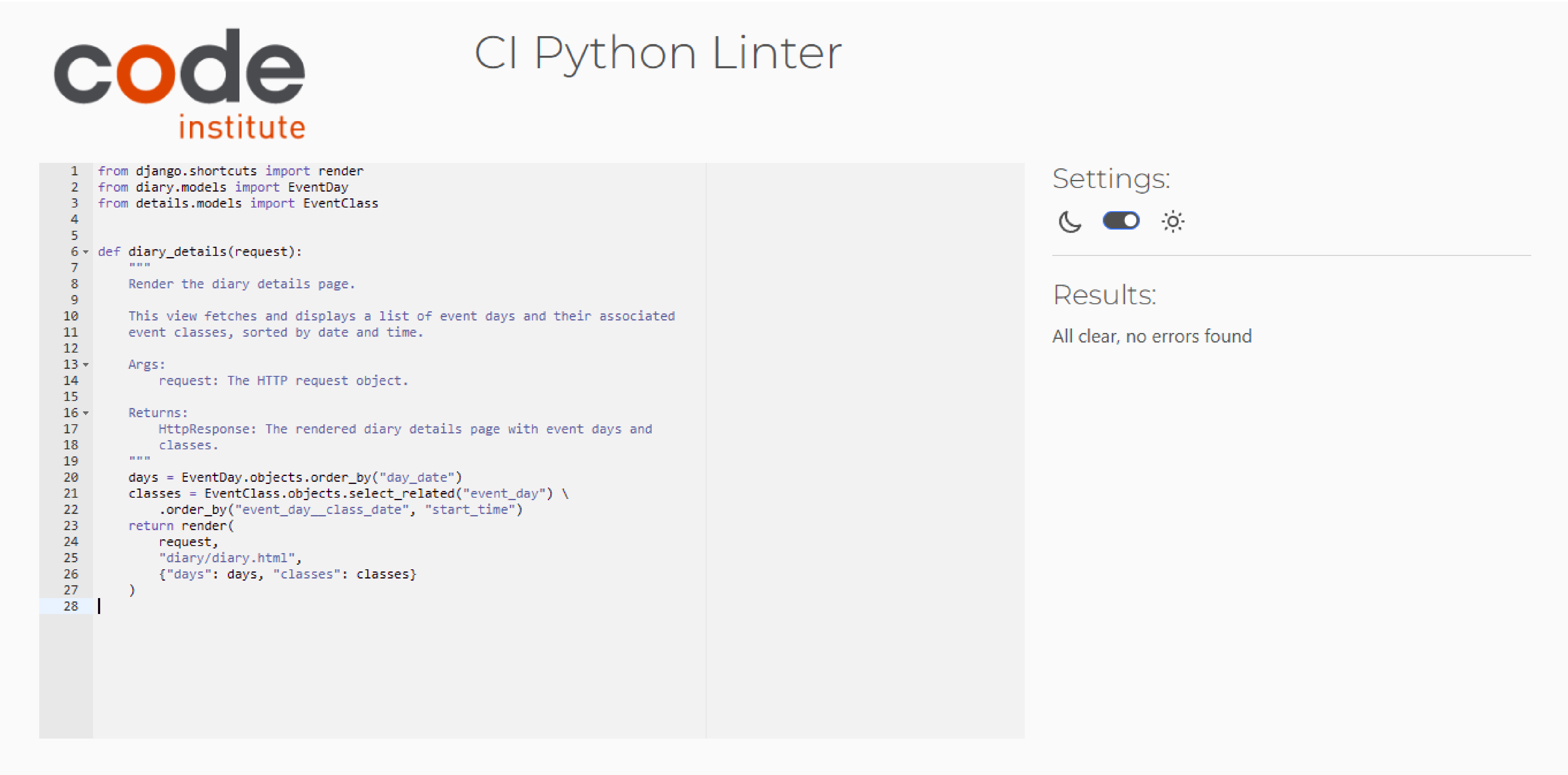
App - Diary



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

Python files created / amended were *views.py, urls.py, models.py, apps.py, and admin.py*

views.py



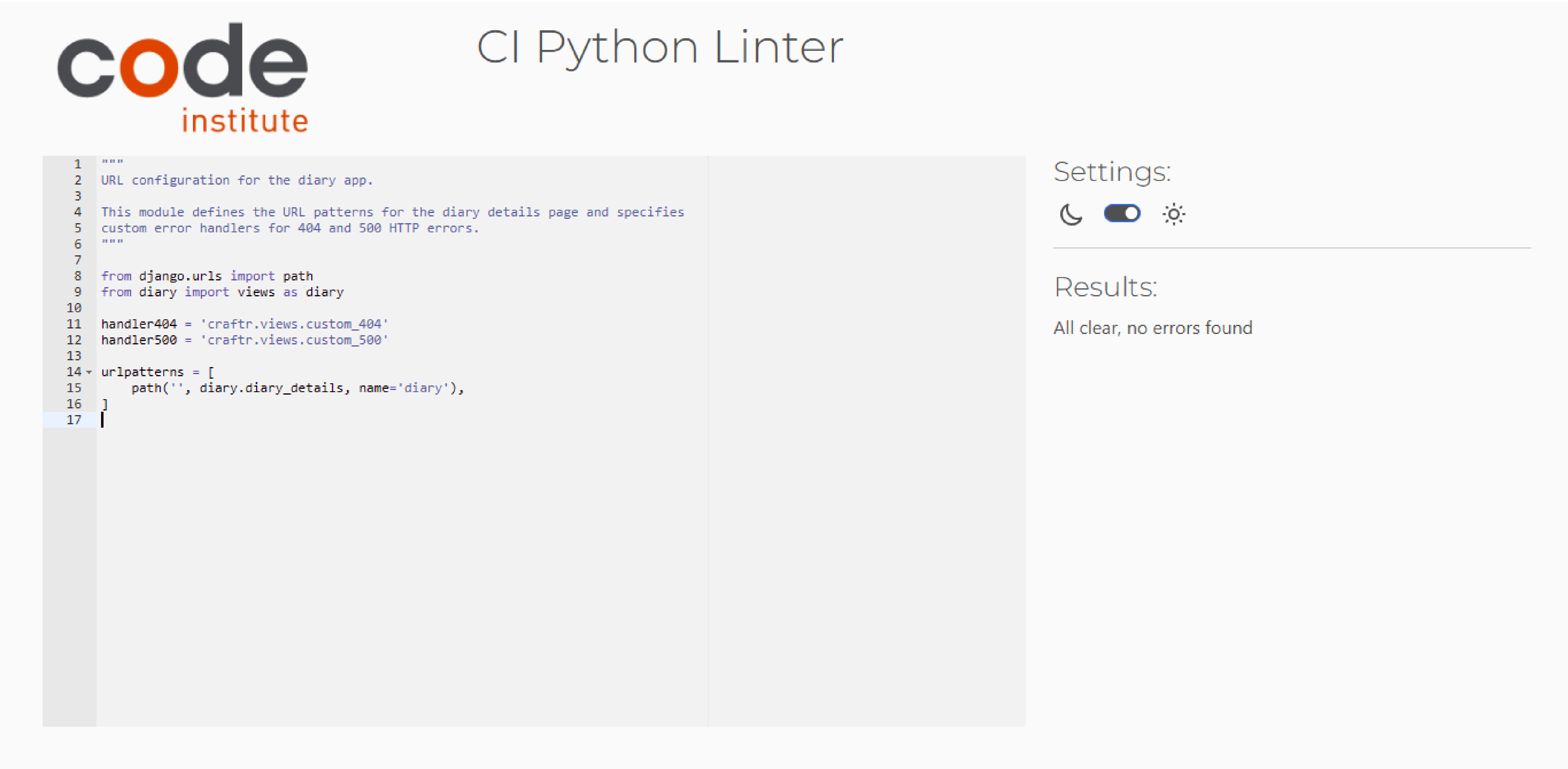
```python
from django.shortcuts import render
from diary.models import EventDay
from details.models import EventClass


def diary_details(request):
    """
    Render the diary details page.

    This view fetches and displays a list of event days and their associated
    event classes, sorted by date and time.

    Args:
        request: The HTTP request object.

    Returns:
        HttpResponse: The rendered diary details page with event days and
        classes.
    """
    days = EventDay.objects.order_by("day_date")
    classes = EventClass.objects.select_related("event_day") \
        .order_by("event_day__class_date", "start_time")
    return render(
        request,
        "diary/diary.html",
        {"days": days, "classes": classes}
    )
```

Settings:

Results:

All clear, no errors found

urls.py



CI Python Linter

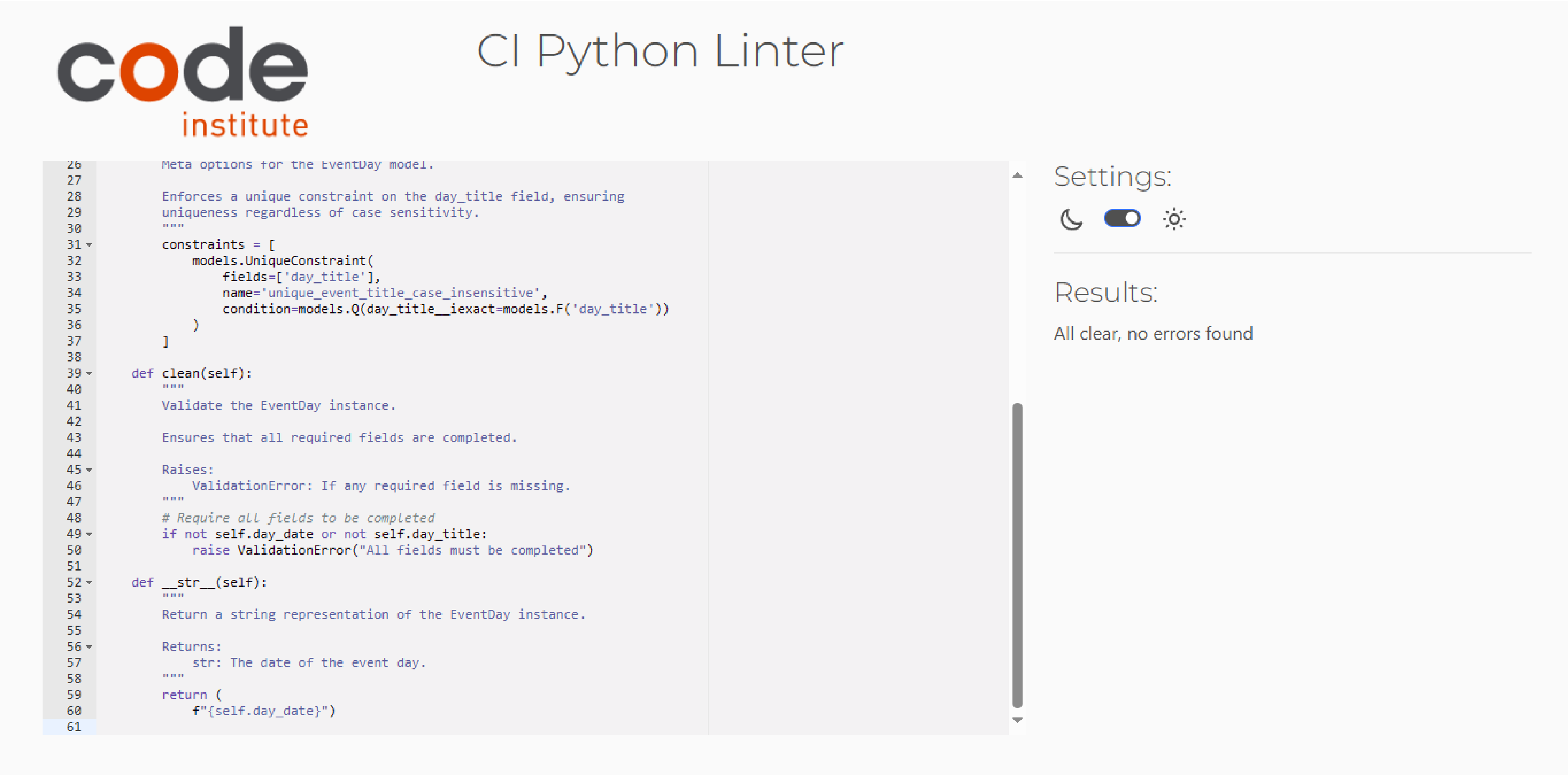
```python
"""
URL configuration for the diary app.

This module defines the URL patterns for the diary details page and specifies
custom error handlers for 404 and 500 HTTP errors.
"""

from django.urls import path
from diary import views as diary

handler404 = 'craftr.views.custom_404'
handler500 = 'craftr.views.custom_500'

urlpatterns = [
    path('', diary.diary_details, name='diary'),
]
```

Settings:

Results:

All clear, no errors found

models.py


CI Python Linter

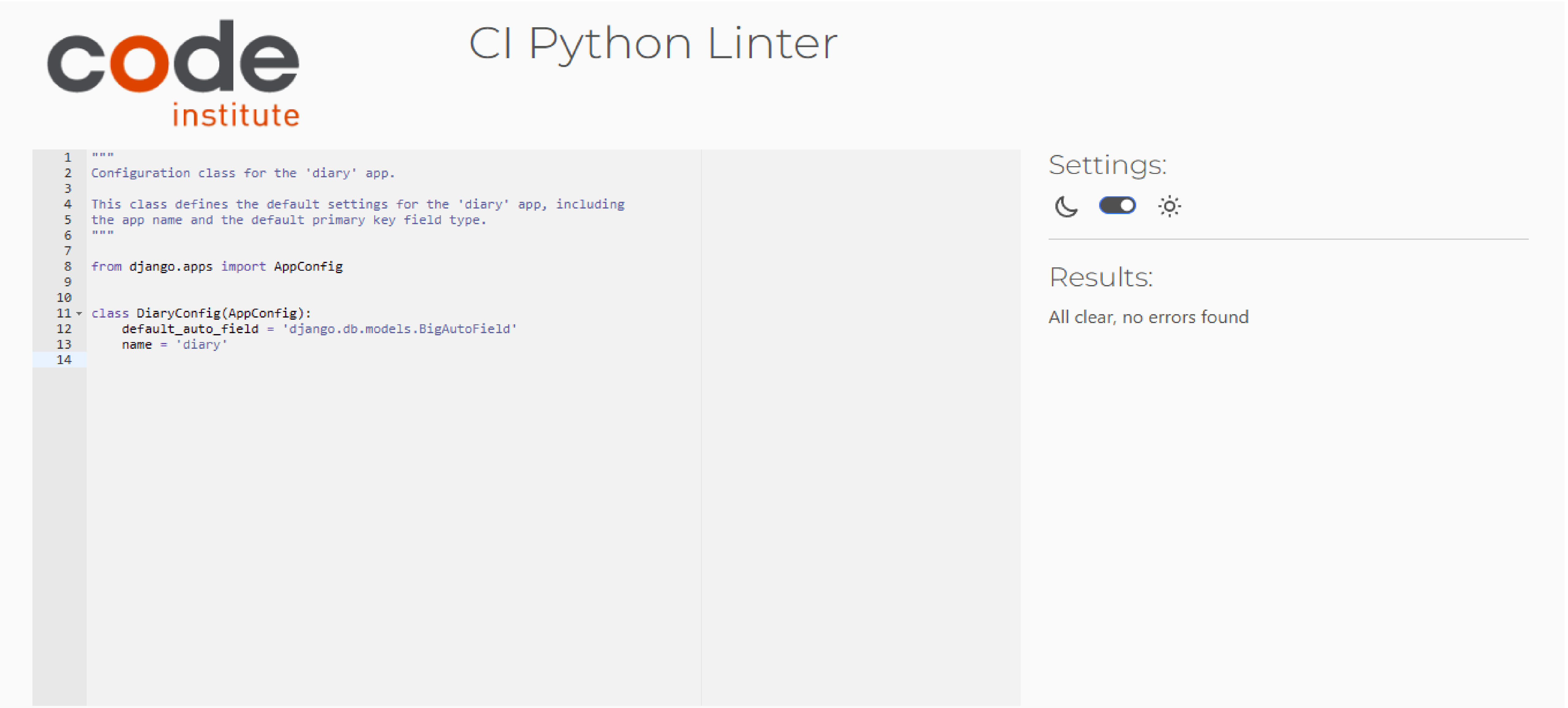
```
        Meta options for the EventDay model.

        Enforces a unique constraint on the day_title field, ensuring
        uniqueness regardless of case sensitivity.
        """
        constraints = [
            models.UniqueConstraint(
                fields=['day_title'],
                name='unique_event_title_case_insensitive',
                condition=models.Q(day_title__iexact=models.F('day_title'))
            )
        ]

    def clean(self):
        """
        Validate the EventDay instance.

        Ensures that all required fields are completed.

        Raises:
            ValidationError: If any required field is missing.
        """
        # Require all fields to be completed
        if not self.day_date or not self.day_title:
            raise ValidationError("All fields must be completed")

    def __str__(self):
        """
        Return a string representation of the EventDay instance.

        Returns:
            str: The date of the event day.
        """
        return (
            f"{self.day_date}")
```

Settings:

Results:

All clear, no errors found

apps.py

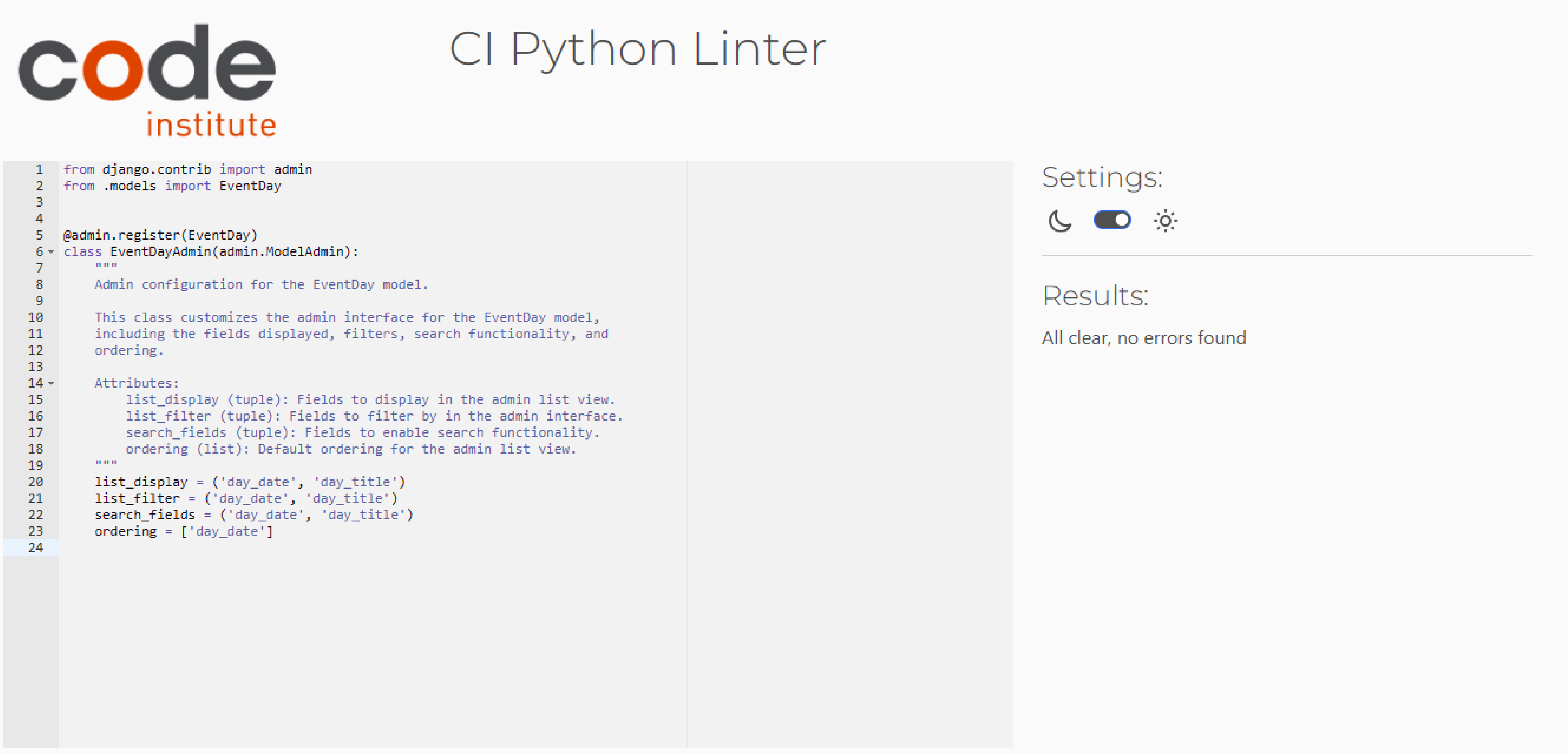


```
"""
Configuration class for the 'diary' app.

This class defines the default settings for the 'diary' app, including
the app name and the default primary key field type.
"""

from django.apps import AppConfig


class DiaryConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'diary'

```

Results:

All clear, no errors found

admin.py


CI Python Linter

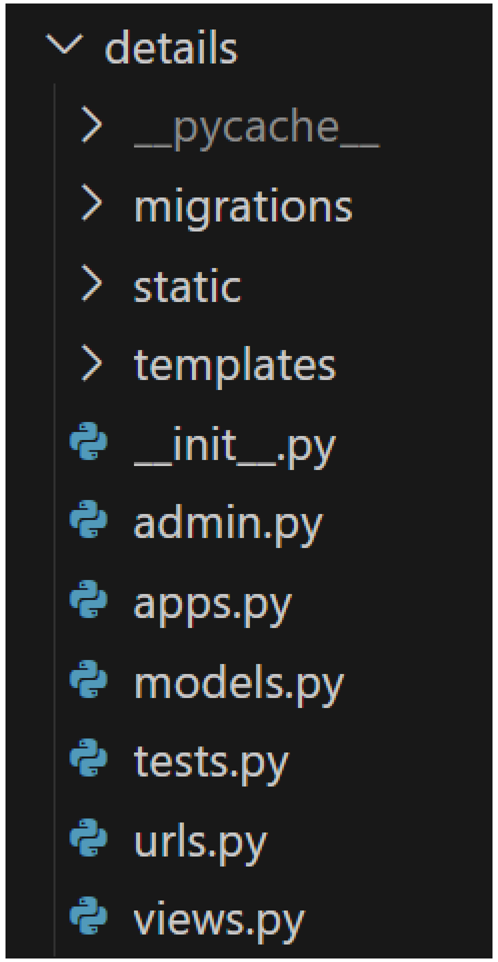
```
from django.contrib import admin
from .models import EventDay


@admin.register(EventDay)
class EventDayAdmin(admin.ModelAdmin):
    """
    Admin configuration for the EventDay model.

    This class customizes the admin interface for the EventDay model,
    including the fields displayed, filters, search functionality, and
    ordering.

    Attributes:
        list_display (tuple): Fields to display in the admin list view.
        list_filter (tuple): Fields to filter by in the admin interface.
        search_fields (tuple): Fields to enable search functionality.
        ordering (list): Default ordering for the admin list view.
    """
    list_display = ('day_date', 'day_title')
    list_filter = ('day_date', 'day_title')
    search_fields = ('day_date', 'day_title')
    ordering = ['day_date']

```

Settings:

Results:

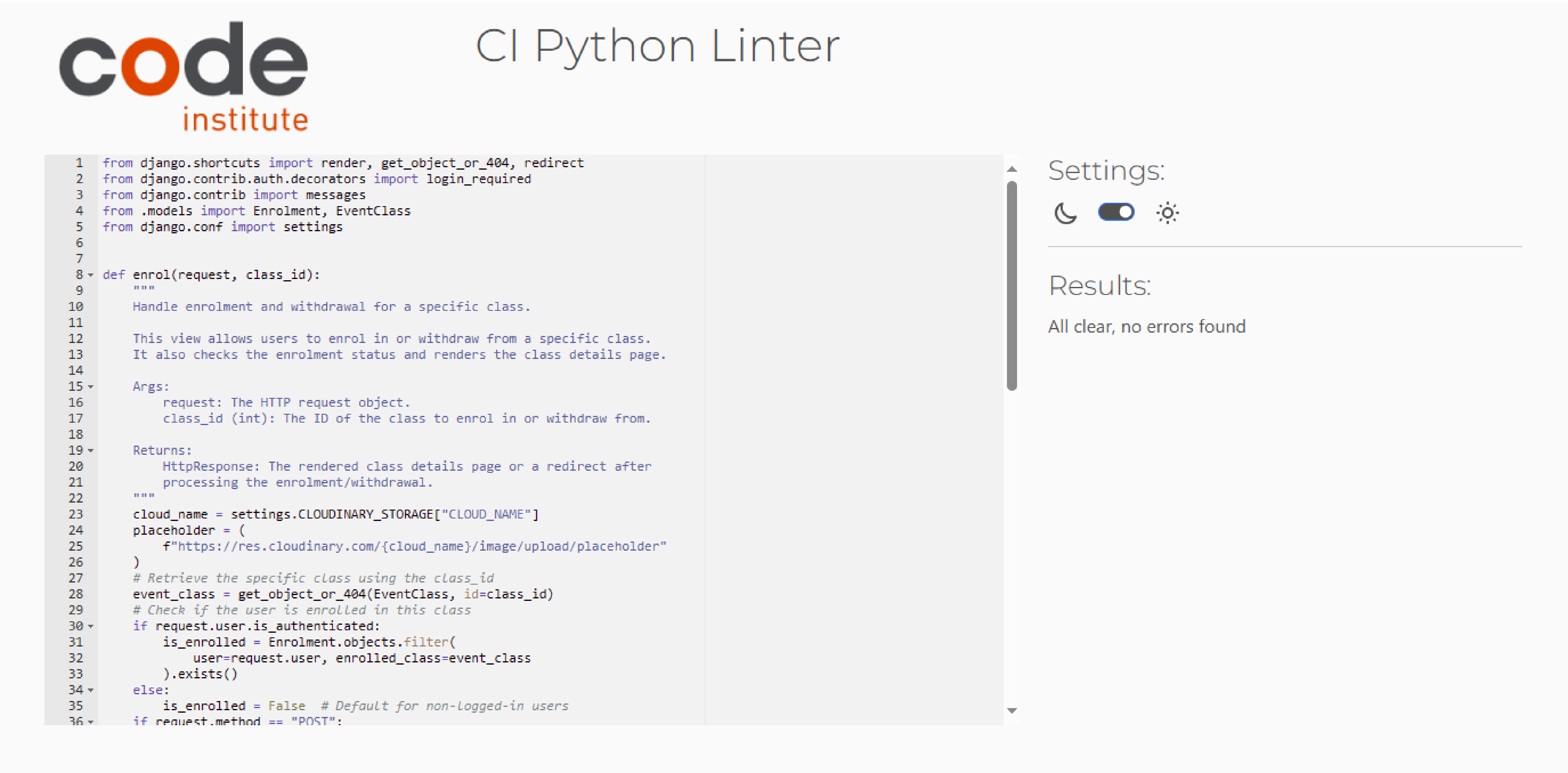
All clear, no errors found

App - Details



Python files created / amended were *views.py, urls.py, models.py, apps.py, and admin.py*

views.py



CI Python Linter

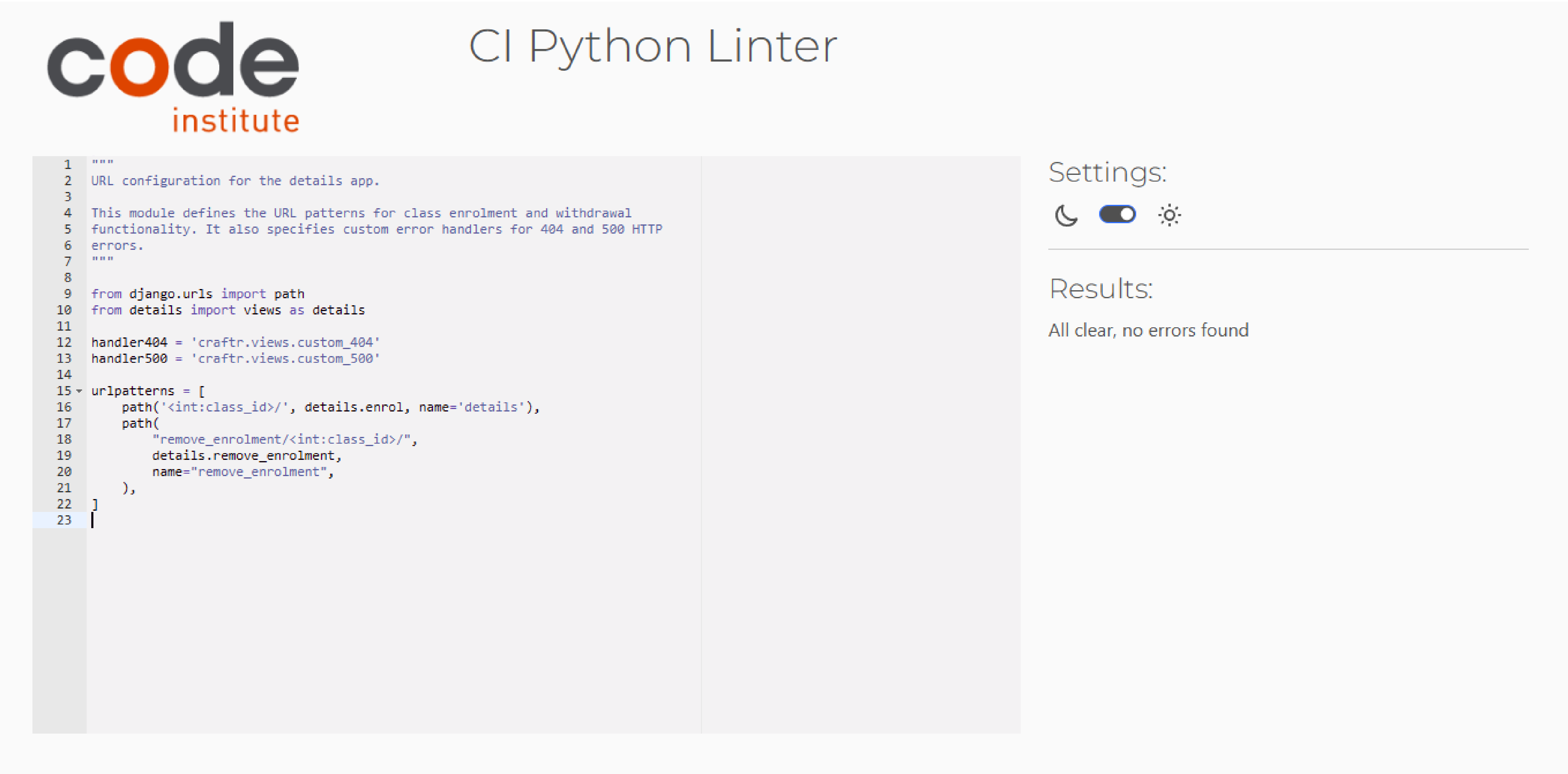
```
from django.shortcuts import render, get_object_or_404, redirect
from django.contrib.auth.decorators import login_required
from django.contrib import messages
from .models import Enrolment, EventClass
from django.conf import settings


def enrol(request, class_id):
    """
    Handle enrolment and withdrawal for a specific class.

    This view allows users to enrol in or withdraw from a specific class.
    It also checks the enrolment status and renders the class details page.

    Args:
        request: The HTTP request object.
        class_id (int): The ID of the class to enrol in or withdraw from.

    Returns:
        HttpResponse: The rendered class details page or a redirect after
        processing the enrolment/withdrawal.
    """
    cloud_name = settings.CLOUDINARY_STORAGE["CLOUD_NAME"]
    placeholder = (
        f"https://res.cloudinary.com/{cloud_name}/image/upload/placeholder"
    )
    # Retrieve the specific class using the class_id
    event_class = get_object_or_404(EventClass, id=class_id)
    # Check if the user is enrolled in this class
    if request.user.is_authenticated:
        is_enrolled = Enrolment.objects.filter(
            user=request.user, enrolled_class=event_class
        ).exists()
    else:
        is_enrolled = False  # Default for non-logged-in users
    if request.method == "POST":
```

Settings:

Results:

All clear, no errors found

urls.py



CI Python Linter

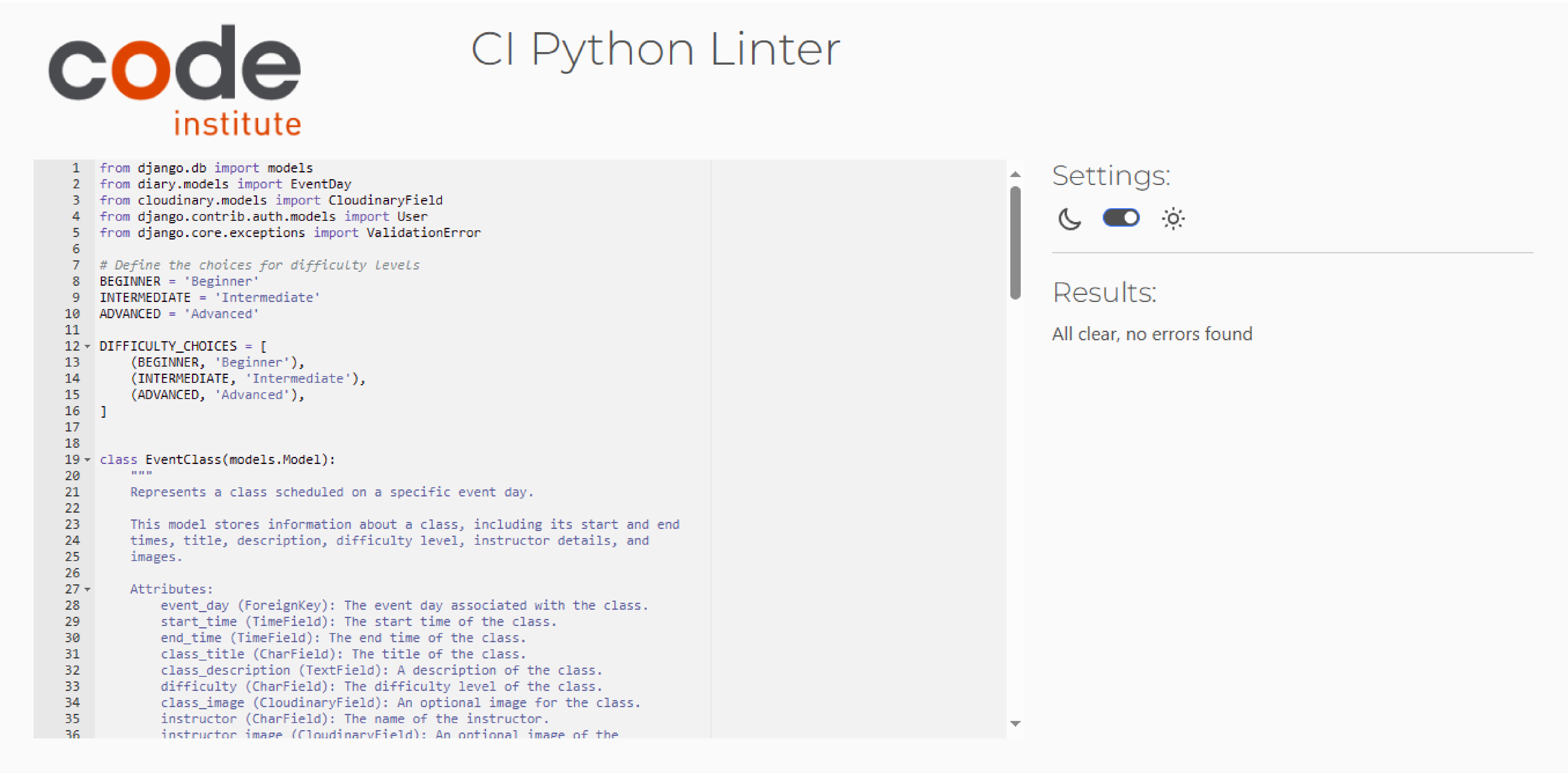
```
"""
URL configuration for the details app.

This module defines the URL patterns for class enrolment and withdrawal
functionality. It also specifies custom error handlers for 404 and 500 HTTP
errors.
"""

from django.urls import path
from details import views as details

handler404 = 'craftr.views.custom_404'
handler500 = 'craftr.views.custom_500'

urlpatterns = [
    path('<int:class_id>/', details.enrol, name='details'),
    path(
        "remove_enrolment/<int:class_id>/",
        details.remove_enrolment,
        name="remove_enrolment",
    ),
]

```

Settings:

Results:

All clear, no errors found

models.py


CI Python Linter

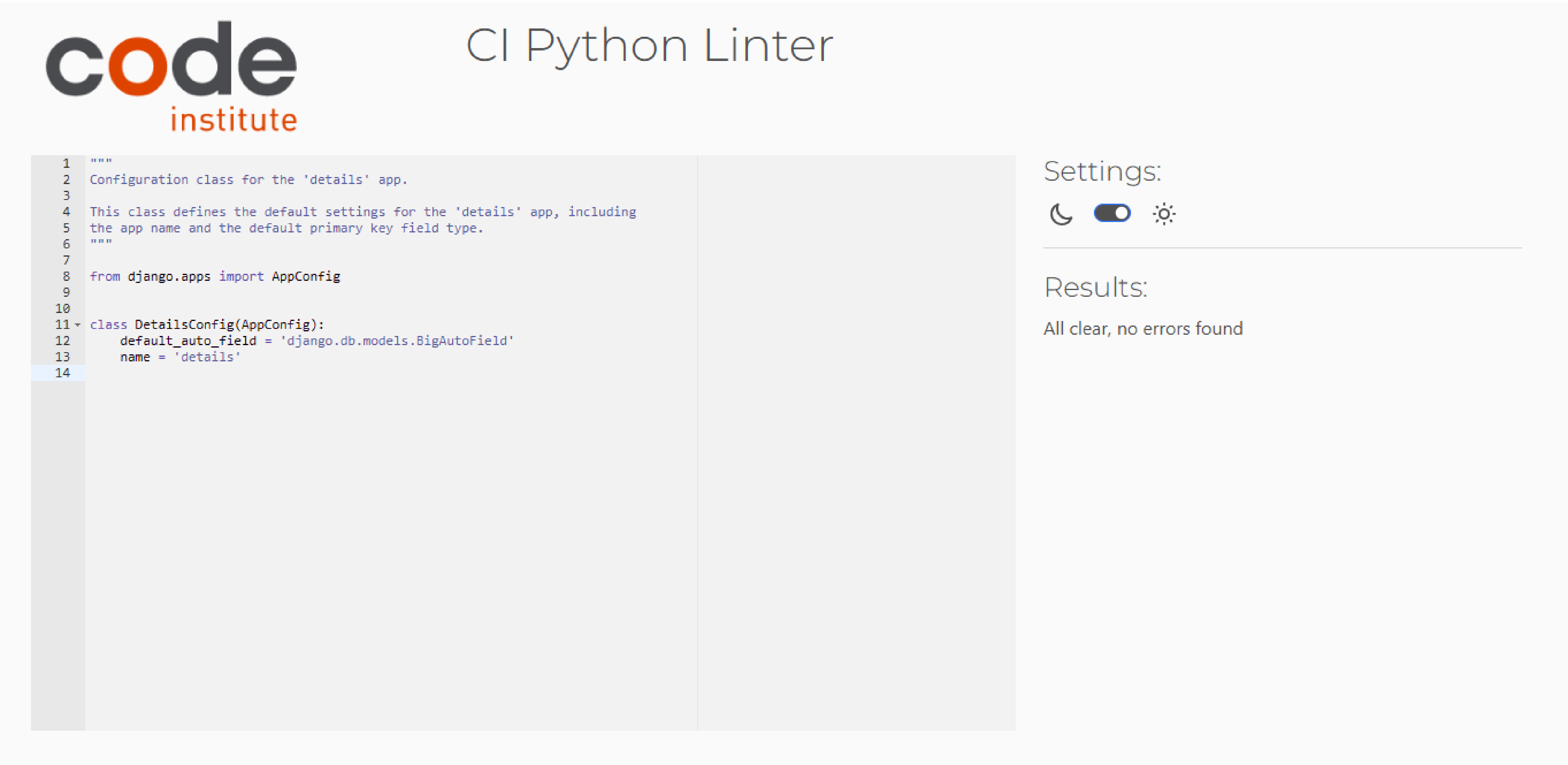
```
from django.db import models
from diary.models import EventDay
from cloudinary.models import CloudinaryField
from django.contrib.auth.models import User
from django.core.exceptions import ValidationError

# Define the choices for difficulty levels
BEGINNER = 'Beginner'
INTERMEDIATE = 'Intermediate'
ADVANCED = 'Advanced'

DIFFICULTY_CHOICES = [
    (BEGINNER, 'Beginner'),
    (INTERMEDIATE, 'Intermediate'),
    (ADVANCED, 'Advanced'),
]


class EventClass(models.Model):
    """
    Represents a class scheduled on a specific event day.

    This model stores information about a class, including its start and end
    times, title, description, difficulty level, instructor details, and
    images.

    Attributes:
        event_day (ForeignKey): The event day associated with the class.
        start_time (TimeField): The start time of the class.
        end_time (TimeField): The end time of the class.
        class_title (CharField): The title of the class.
        class_description (TextField): A description of the class.
        difficulty (CharField): The difficulty level of the class.
        class_image (CloudinaryField): An optional image for the class.
        instructor (CharField): The name of the instructor.
        instructor_image (CloudinaryField): An optional image of the
```

Settings:

Results:

All clear, no errors found

apps.py



CI Python Linter

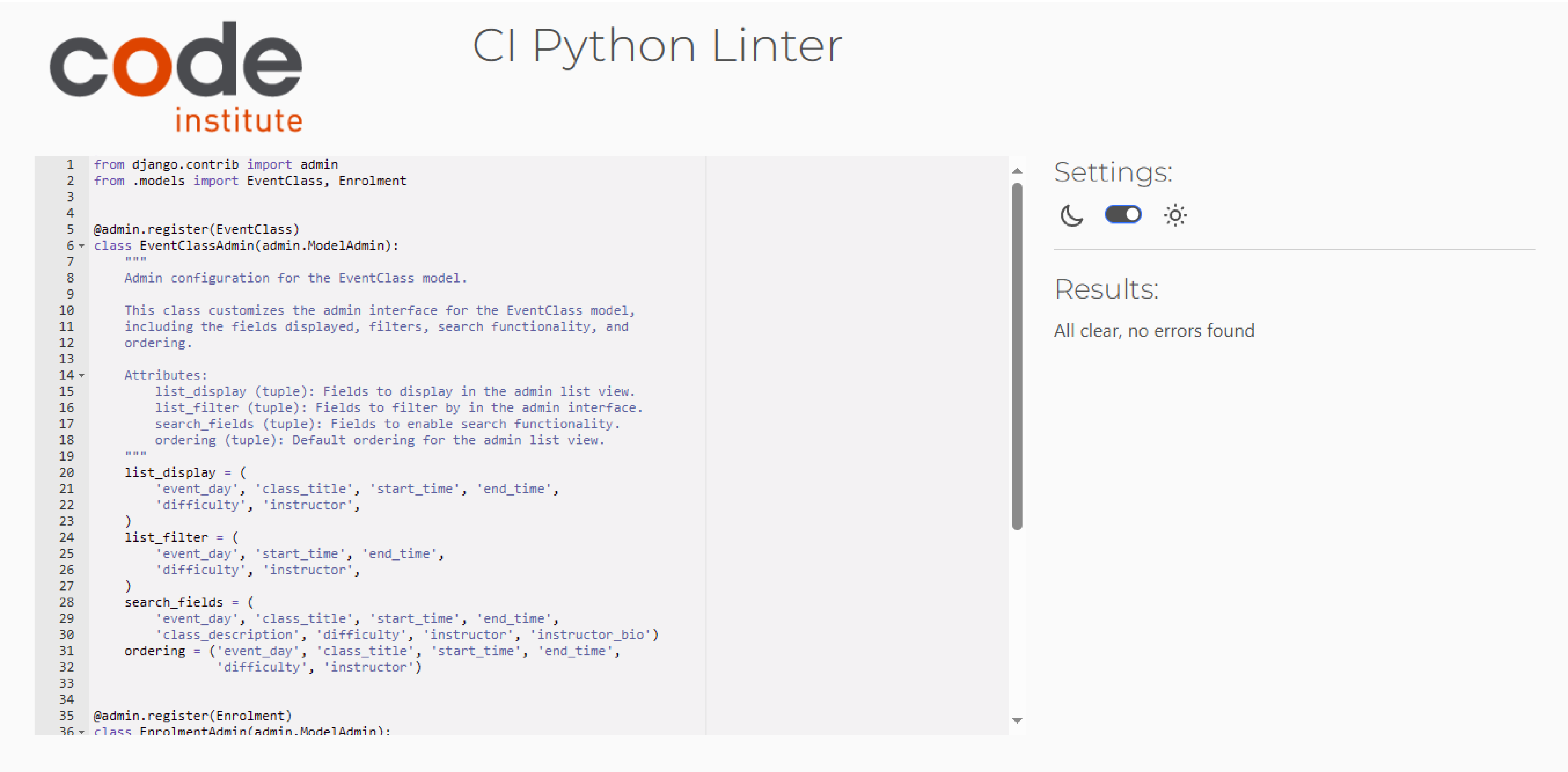
```
"""
Configuration class for the 'details' app.

This class defines the default settings for the 'details' app, including
the app name and the default primary key field type.
"""

from django.apps import AppConfig


class DetailsConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'details'

```

Settings:

Results:

All clear, no errors found

admin.py



CI Python Linter

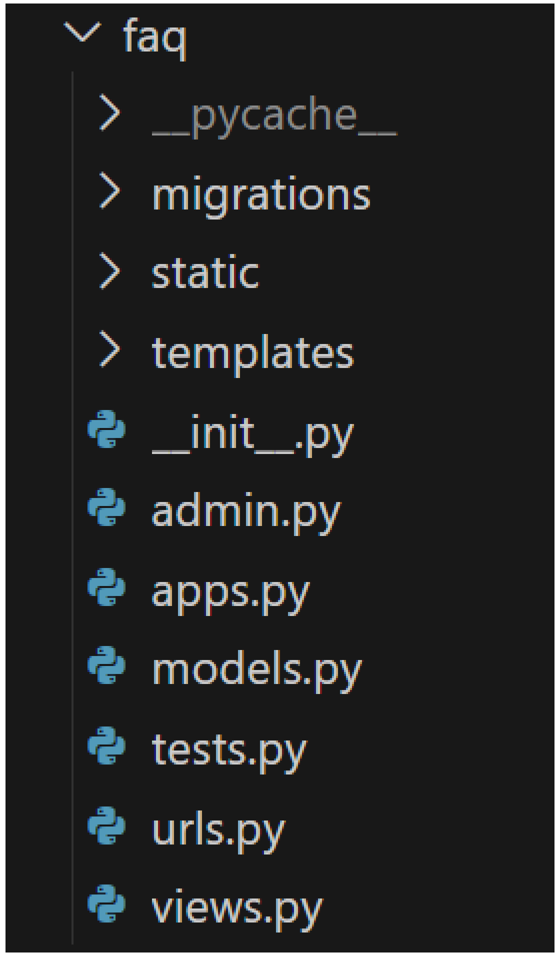
```
from django.contrib import admin
from .models import EventClass, Enrolment


@admin.register(EventClass)
class EventClassAdmin(admin.ModelAdmin):
    """
    Admin configuration for the EventClass model.

    This class customizes the admin interface for the EventClass model,
    including the fields displayed, filters, search functionality, and
    ordering.

    Attributes:
        list_display (tuple): Fields to display in the admin list view.
        list_filter (tuple): Fields to filter by in the admin interface.
        search_fields (tuple): Fields to enable search functionality.
        ordering (tuple): Default ordering for the admin list view.
    """
    list_display = (
        'event_day', 'class_title', 'start_time', 'end_time',
        'difficulty', 'instructor',
    )
    list_filter = (
        'event_day', 'start_time', 'end_time',
        'difficulty', 'instructor',
    )
    search_fields = (
        'event_day', 'class_title', 'start_time', 'end_time',
        'class_description', 'difficulty', 'instructor', 'instructor_bio')
    ordering = ('event_day', 'class_title', 'start_time', 'end_time',
                'difficulty', 'instructor')


@admin.register(Enrolment)
class EnrolmentAdmin(admin.ModelAdmin):
```

Settings:

Results:

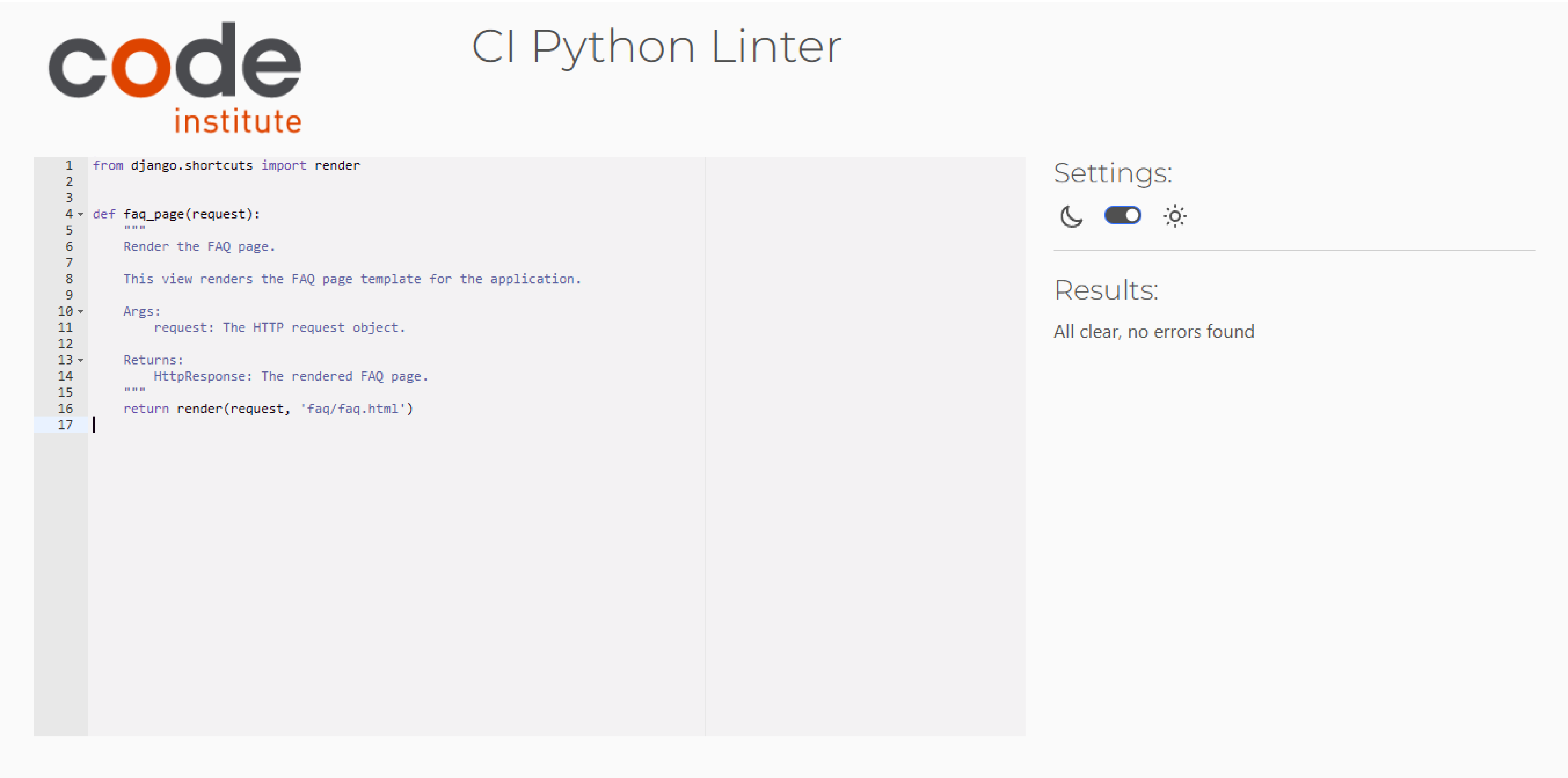
All clear, no errors found

App - FAQ


faq
- __pycache__
- migrations
- static
- templates
- __init__.py
- admin.py
- apps.py
- models.py
- tests.py
- urls.py
- views.py

Python files created / amended were *views.py, urls.py, and apps.py*

views.py



CI Python Linter

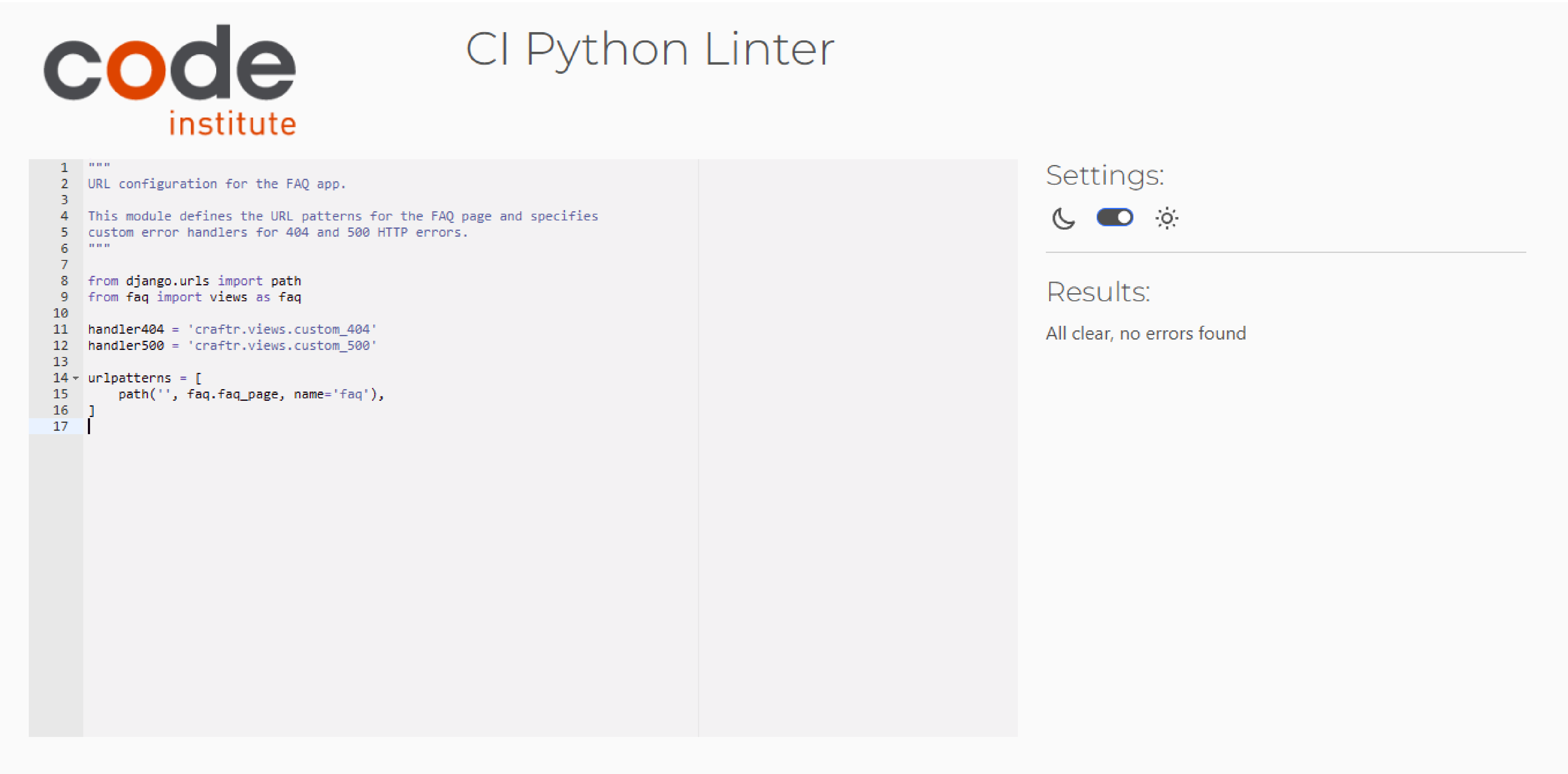
```python
from django.shortcuts import render


def faq_page(request):
    """
    Render the FAQ page.

    This view renders the FAQ page template for the application.

    Args:
        request: The HTTP request object.

    Returns:
        HttpResponse: The rendered FAQ page.
    """
    return render(request, 'faq/faq.html')

```

Settings:

Results:

All clear, no errors found

urls.py



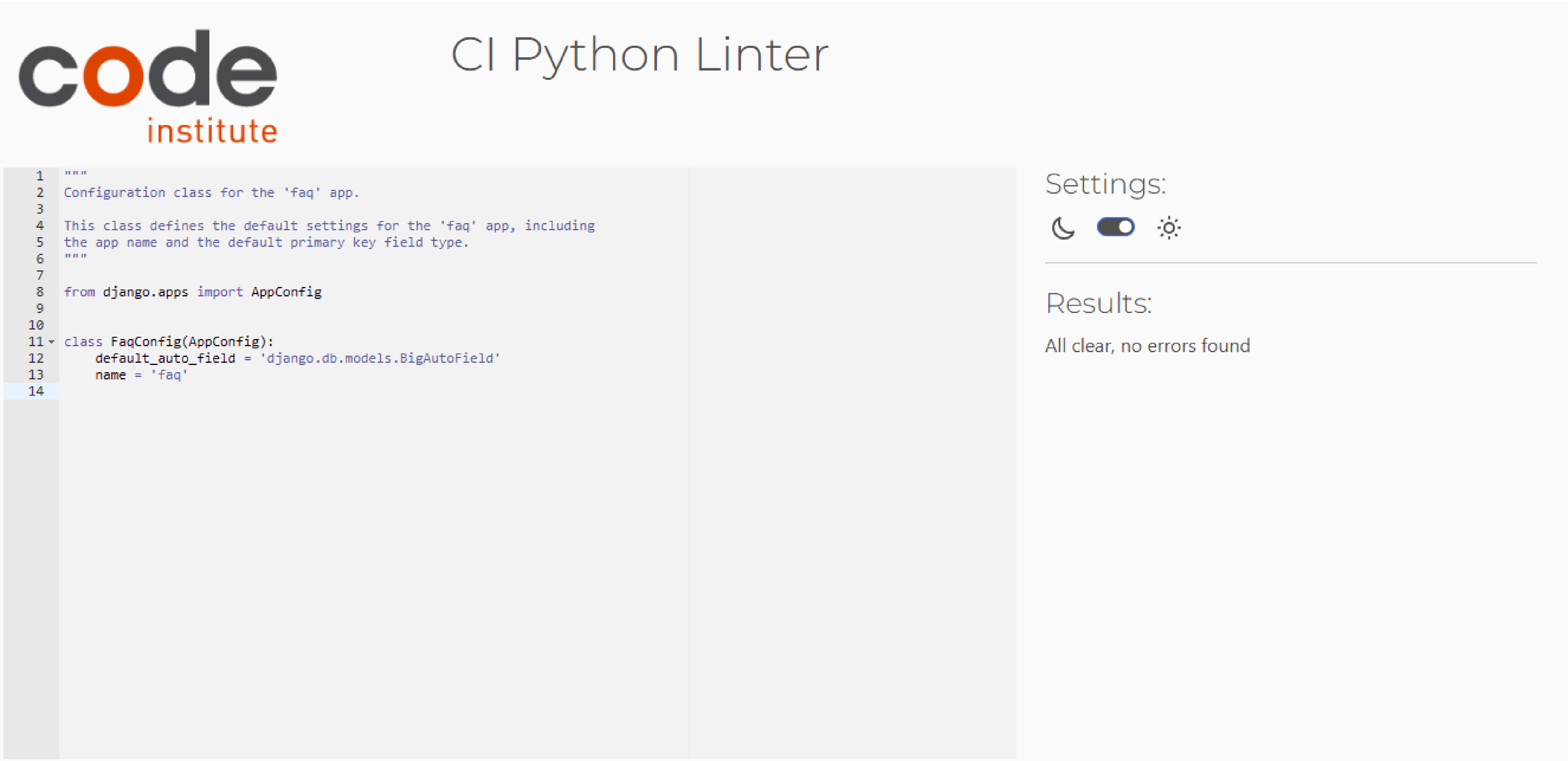
```
"""
URL configuration for the FAQ app.

This module defines the URL patterns for the FAQ page and specifies
custom error handlers for 404 and 500 HTTP errors.
"""

from django.urls import path
from faq import views as faq

handler404 = 'craftr.views.custom_404'
handler500 = 'craftr.views.custom_500'

urlpatterns = [
    path('', faq.faq_page, name='faq'),
]
```

Settings:

Results:

All clear, no errors found

apps.py


CI Python Linter

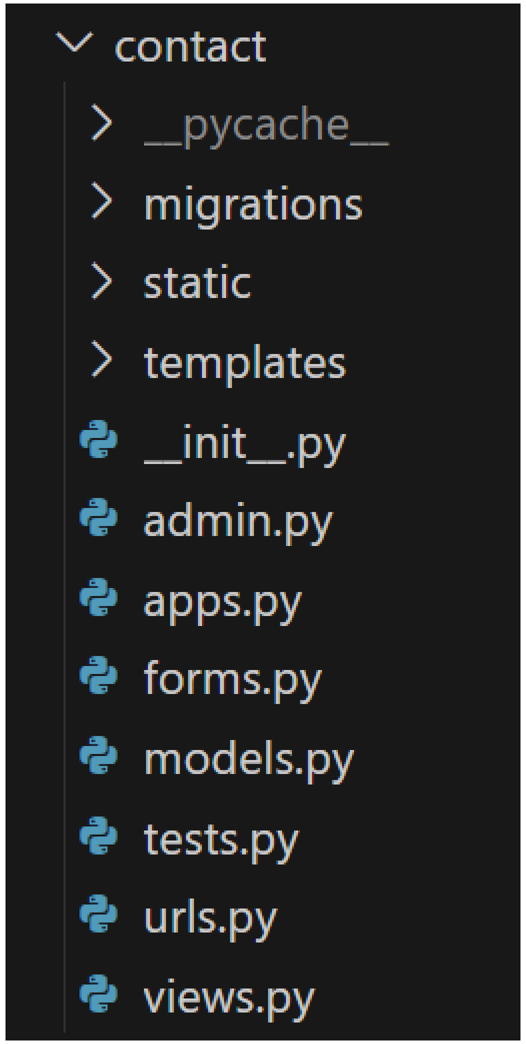
```
"""
Configuration class for the 'faq' app.

This class defines the default settings for the 'faq' app, including
the app name and the default primary key field type.
"""

from django.apps import AppConfig


class FaqConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'faq'
```

Settings:

Results:

All clear, no errors found

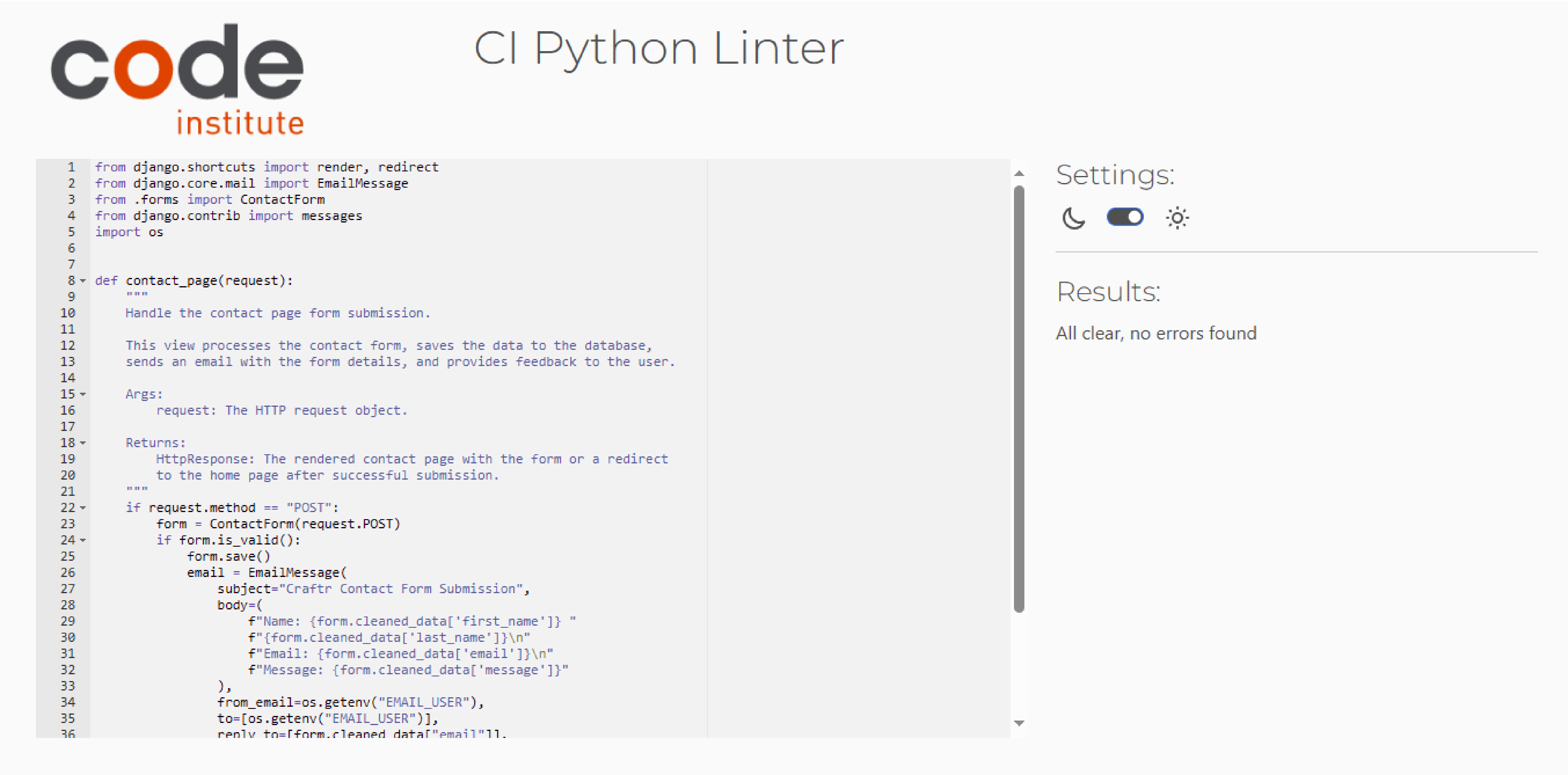
App - Contact



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

Python files created / amended were *views.py, urls.py, models.py, forms.py, apps.py, and admin.py*

views.py


CI Python Linter

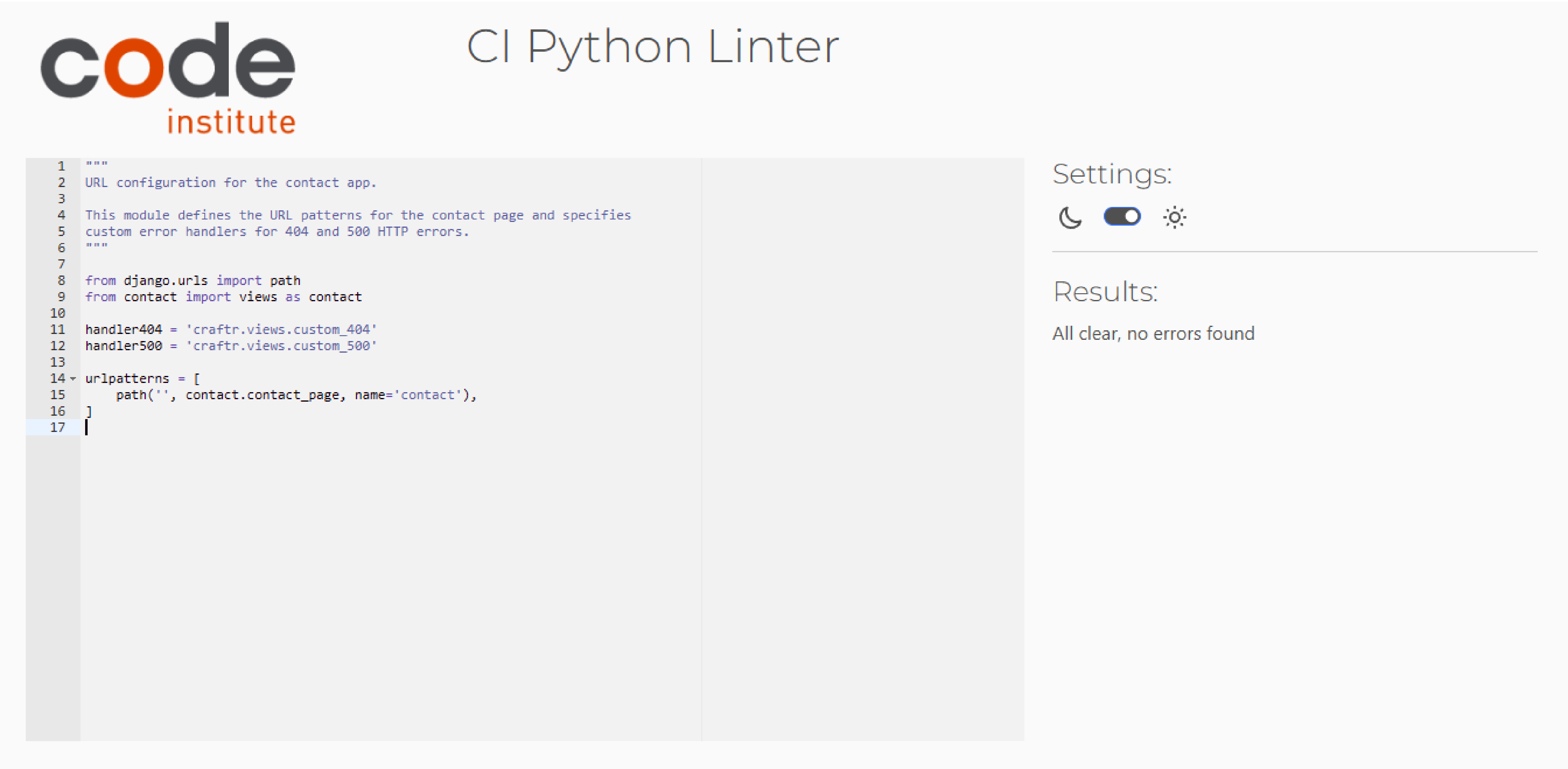
```python
from django.shortcuts import render, redirect
from django.core.mail import EmailMessage
from .forms import ContactForm
from django.contrib import messages
import os


def contact_page(request):
    """
    Handle the contact page form submission.

    This view processes the contact form, saves the data to the database,
    sends an email with the form details, and provides feedback to the user.

    Args:
        request: The HTTP request object.

    Returns:
        HttpResponse: The rendered contact page with the form or a redirect
        to the home page after successful submission.
    """
    if request.method == "POST":
        form = ContactForm(request.POST)
        if form.is_valid():
            form.save()
            email = EmailMessage(
                subject="Craftr Contact Form Submission",
                body=(
                    f"Name: {form.cleaned_data['first_name']} "
                    f"{form.cleaned_data['last_name']}\n"
                    f"Email: {form.cleaned_data['email']}\n"
                    f"Message: {form.cleaned_data['message']}"
                ),
                from_email=os.getenv("EMAIL_USER"),
                to=[os.getenv("EMAIL_USER")],
                reply_to=[form.cleaned_data["email"]],
```

Settings:

Results:

All clear, no errors found

urls.py



CI Python Linter

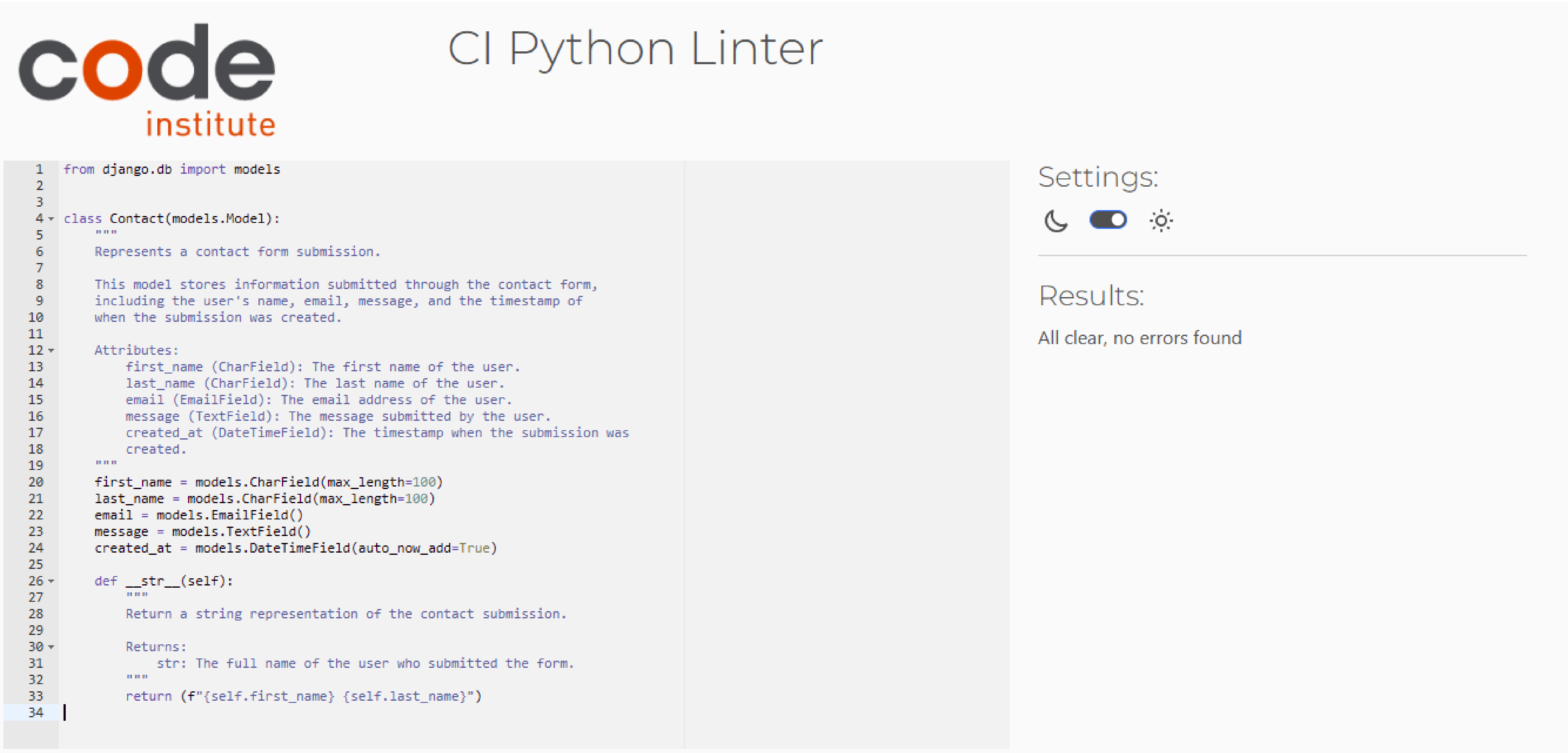
```
"""
URL configuration for the contact app.

This module defines the URL patterns for the contact page and specifies
custom error handlers for 404 and 500 HTTP errors.
"""

from django.urls import path
from contact import views as contact

handler404 = 'craftr.views.custom_404'
handler500 = 'craftr.views.custom_500'

urlpatterns = [
    path('', contact.contact_page, name='contact'),
]

```

Settings:

Results:

All clear, no errors found

models.py


CI Python Linter

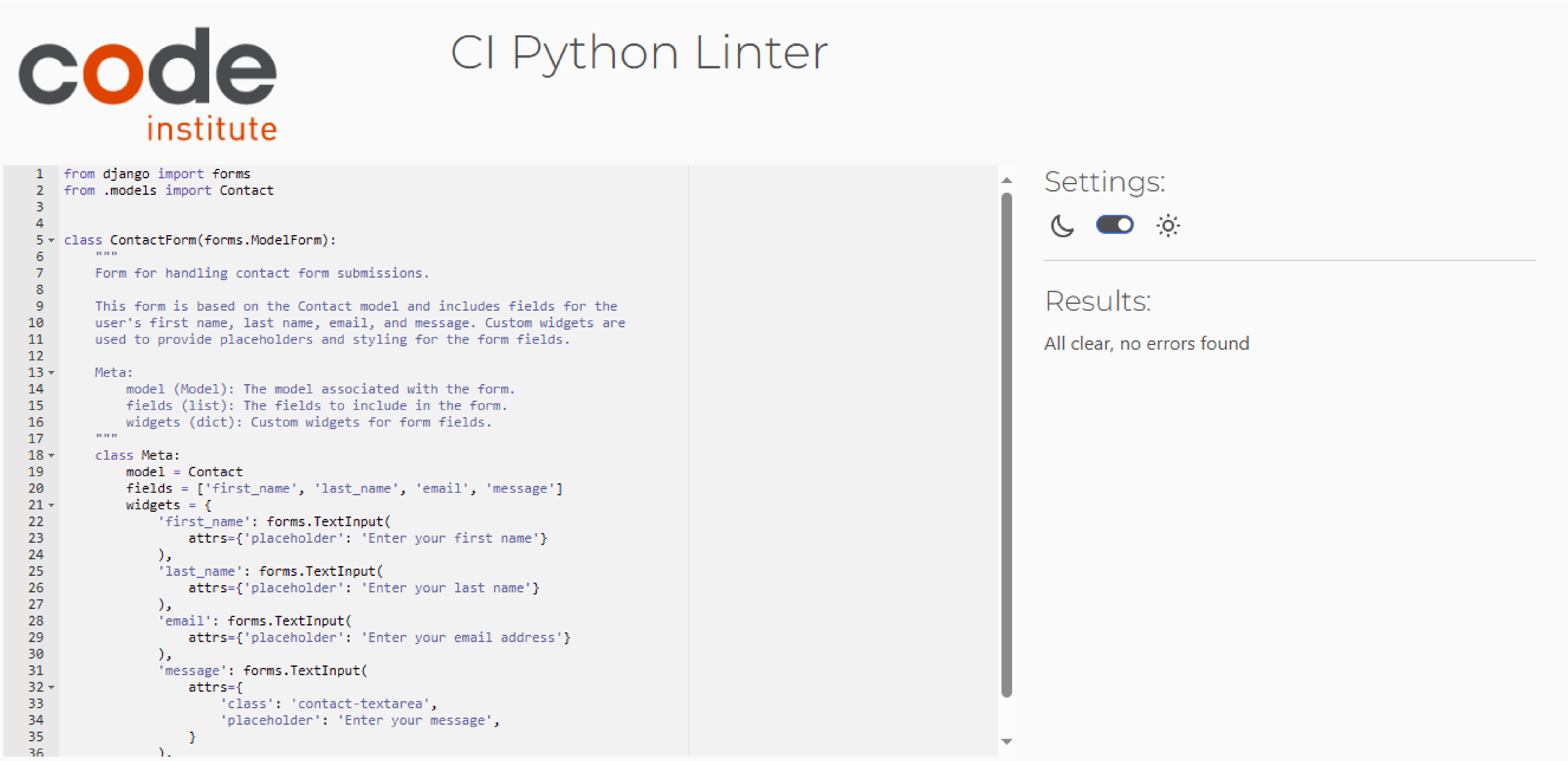
```
from django.db import models


class Contact(models.Model):
    """
    Represents a contact form submission.

    This model stores information submitted through the contact form,
    including the user's name, email, message, and the timestamp of
    when the submission was created.

    Attributes:
        first_name (CharField): The first name of the user.
        last_name (CharField): The last name of the user.
        email (EmailField): The email address of the user.
        message (TextField): The message submitted by the user.
        created_at (DateTimeField): The timestamp when the submission was
        created.
    """
    first_name = models.CharField(max_length=100)
    last_name = models.CharField(max_length=100)
    email = models.EmailField()
    message = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        """
        Return a string representation of the contact submission.

        Returns:
            str: The full name of the user who submitted the form.
        """
        return (f"{self.first_name} {self.last_name}")

```

Settings:

Results:

All clear, no errors found

forms.py


CI Python Linter

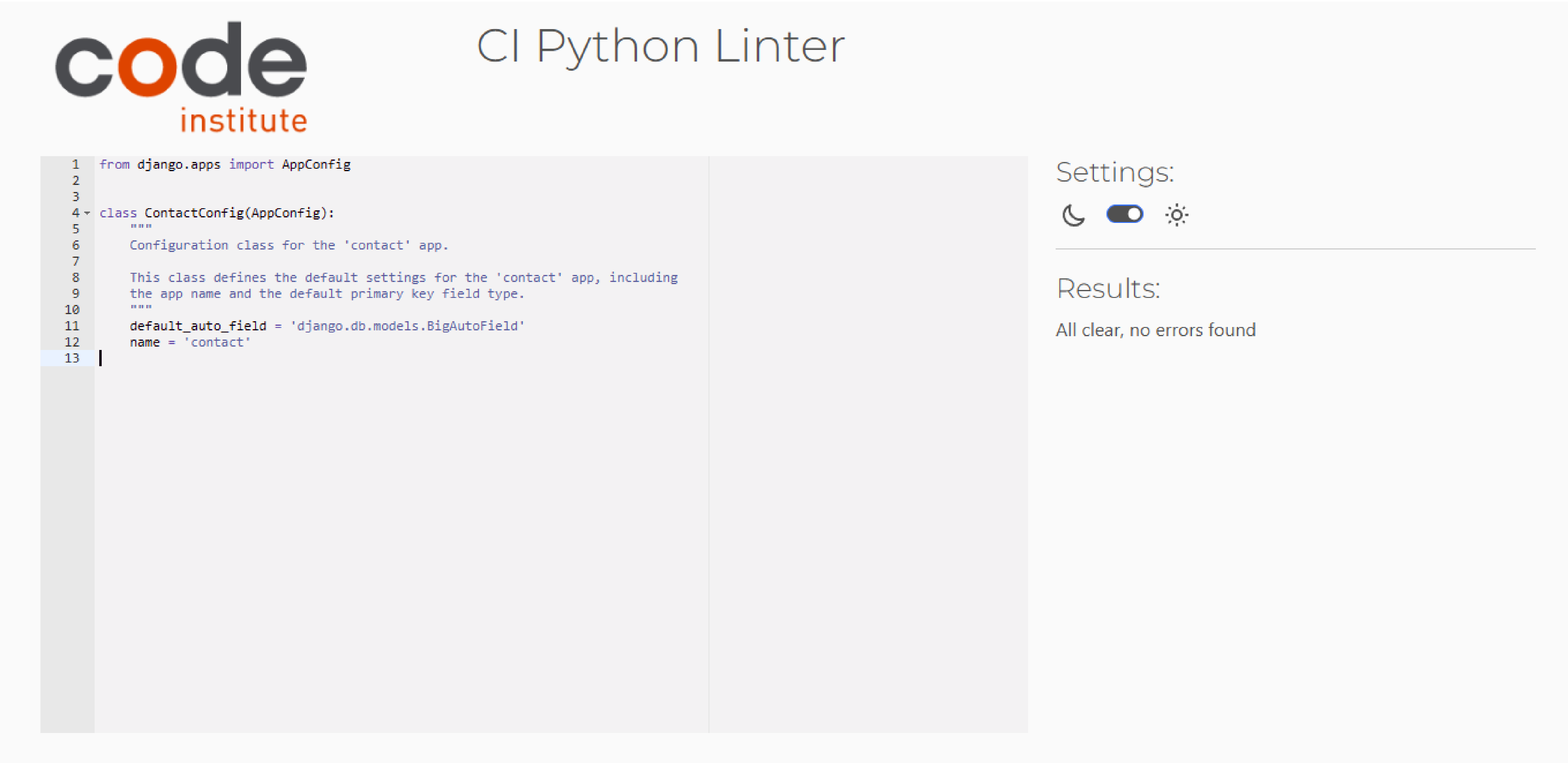
```
from django import forms
from .models import Contact


class ContactForm(forms.ModelForm):
    """
    Form for handling contact form submissions.

    This form is based on the Contact model and includes fields for the
    user's first name, last name, email, and message. Custom widgets are
    used to provide placeholders and styling for the form fields.

    Meta:
        model (Model): The model associated with the form.
        fields (list): The fields to include in the form.
        widgets (dict): Custom widgets for form fields.
    """
    class Meta:
        model = Contact
        fields = ['first_name', 'last_name', 'email', 'message']
        widgets = {
            'first_name': forms.TextInput(
                attrs={'placeholder': 'Enter your first name'}
            ),
            'last_name': forms.TextInput(
                attrs={'placeholder': 'Enter your last name'}
            ),
            'email': forms.TextInput(
                attrs={'placeholder': 'Enter your email address'}
            ),
            'message': forms.TextInput(
                attrs={
                    'class': 'contact-textarea',
                    'placeholder': 'Enter your message',
                }
            ),
```

Settings:

Results:

All clear, no errors found

apps.py


CI Python Linter

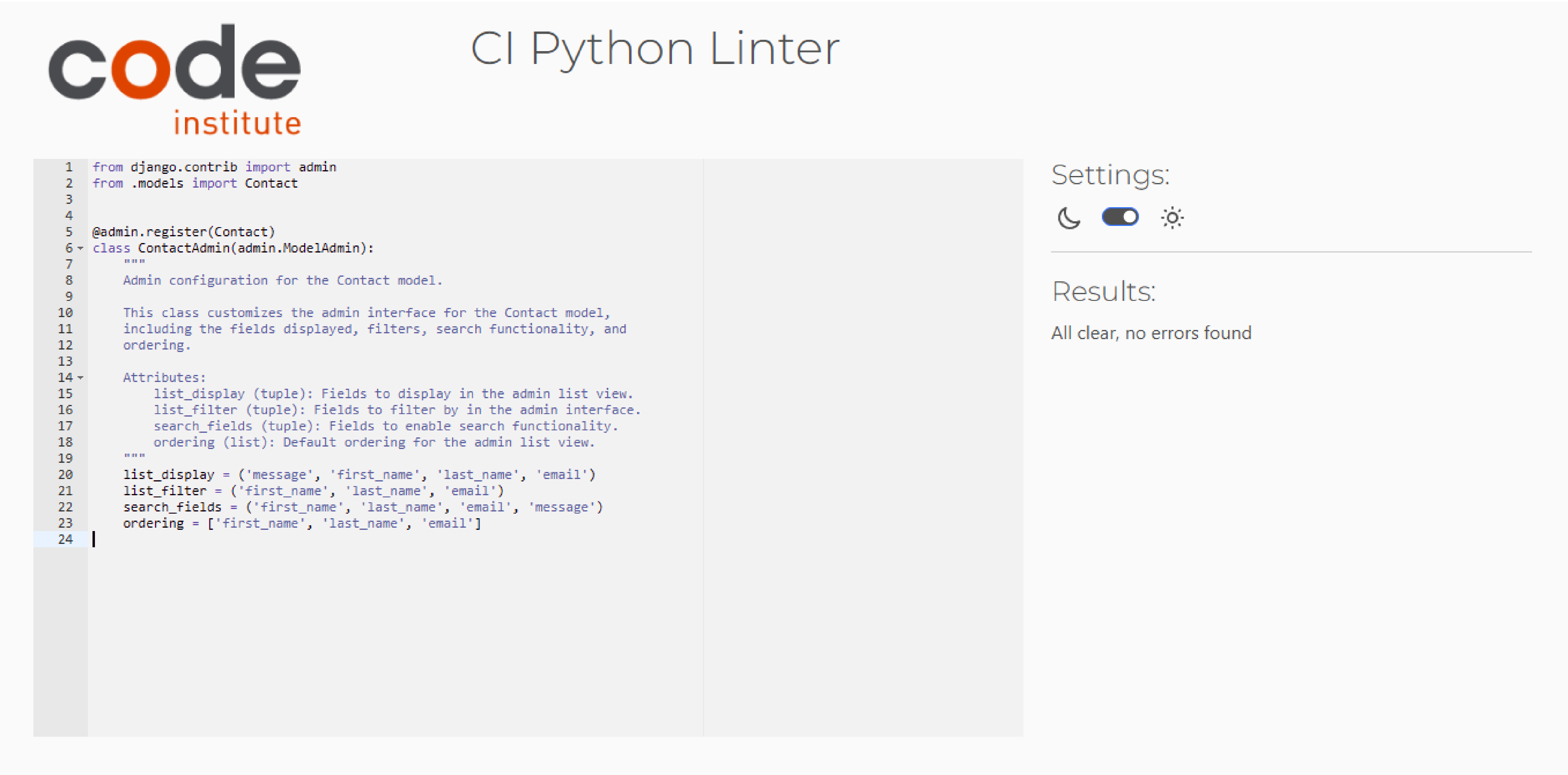
```
from django.apps import AppConfig


class ContactConfig(AppConfig):
    """
    Configuration class for the 'contact' app.

    This class defines the default settings for the 'contact' app, including
    the app name and the default primary key field type.
    """
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'contact'

```

Settings:

Results:

All clear, no errors found

admin.py



CI Python Linter

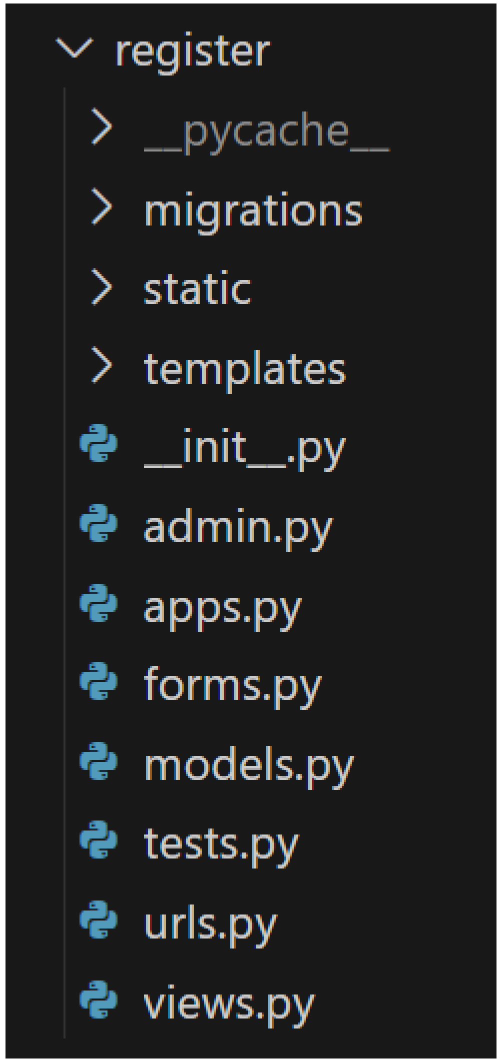
```python
from django.contrib import admin
from .models import Contact


@admin.register(Contact)
class ContactAdmin(admin.ModelAdmin):
    """
    Admin configuration for the Contact model.

    This class customizes the admin interface for the Contact model,
    including the fields displayed, filters, search functionality, and
    ordering.

    Attributes:
        list_display (tuple): Fields to display in the admin list view.
        list_filter (tuple): Fields to filter by in the admin interface.
        search_fields (tuple): Fields to enable search functionality.
        ordering (list): Default ordering for the admin list view.
    """
    list_display = ('message', 'first_name', 'last_name', 'email')
    list_filter = ('first_name', 'last_name', 'email')
    search_fields = ('first_name', 'last_name', 'email', 'message')
    ordering = ['first_name', 'last_name', 'email']

```

Settings:

Results:

All clear, no errors found

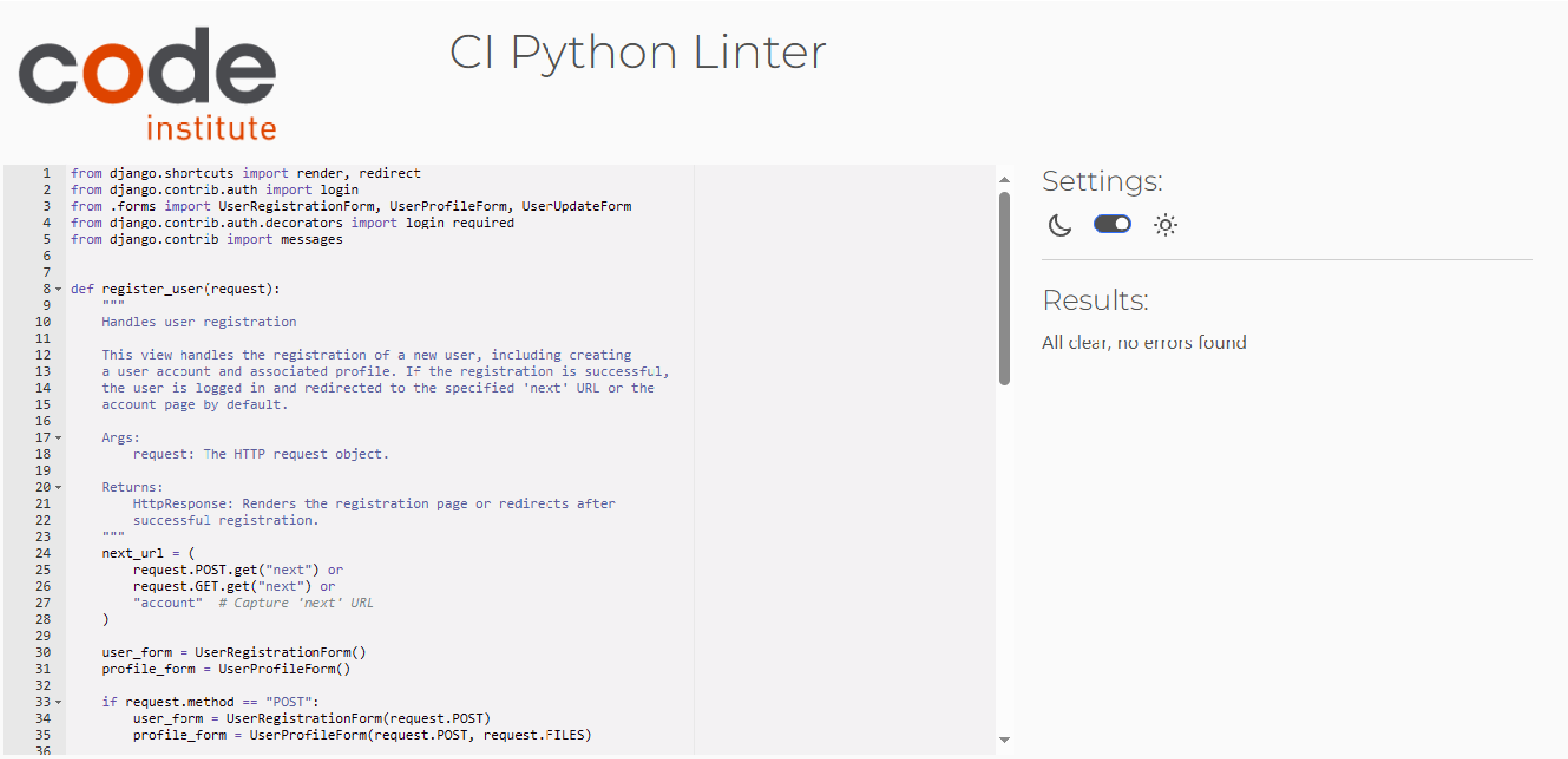
App - Register



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

Python files created / amended were *views.py, urls.py, models.py, forms.py, apps.py, and admin.py*

views.py



CI Python Linter

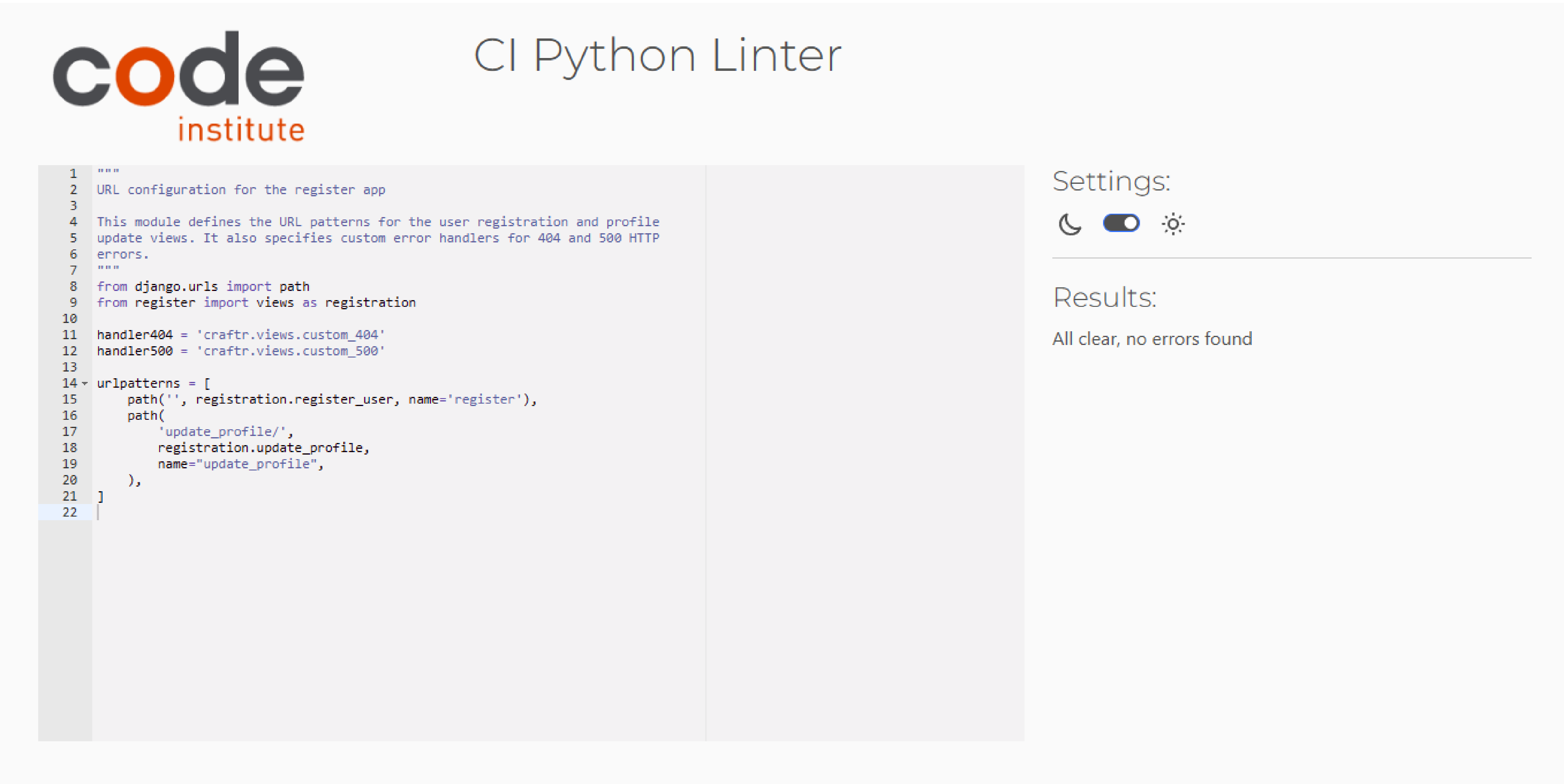
```python
from django.shortcuts import render, redirect
from django.contrib.auth import login
from .forms import UserRegistrationForm, UserProfileForm, UserUpdateForm
from django.contrib.auth.decorators import login_required
from django.contrib import messages


def register_user(request):
    """
    Handles user registration

    This view handles the registration of a new user, including creating
    a user account and associated profile. If the registration is successful,
    the user is logged in and redirected to the specified 'next' URL or the
    account page by default.

    Args:
        request: The HTTP request object.

    Returns:
        HttpResponse: Renders the registration page or redirects after
        successful registration.
    """
    next_url = (
        request.POST.get("next") or
        request.GET.get("next") or
        "account"  # Capture 'next' URL
    )

    user_form = UserRegistrationForm()
    profile_form = UserProfileForm()

    if request.method == "POST":
        user_form = UserRegistrationForm(request.POST)
        profile_form = UserProfileForm(request.POST, request.FILES)
```

Settings:

Results:

All clear, no errors found

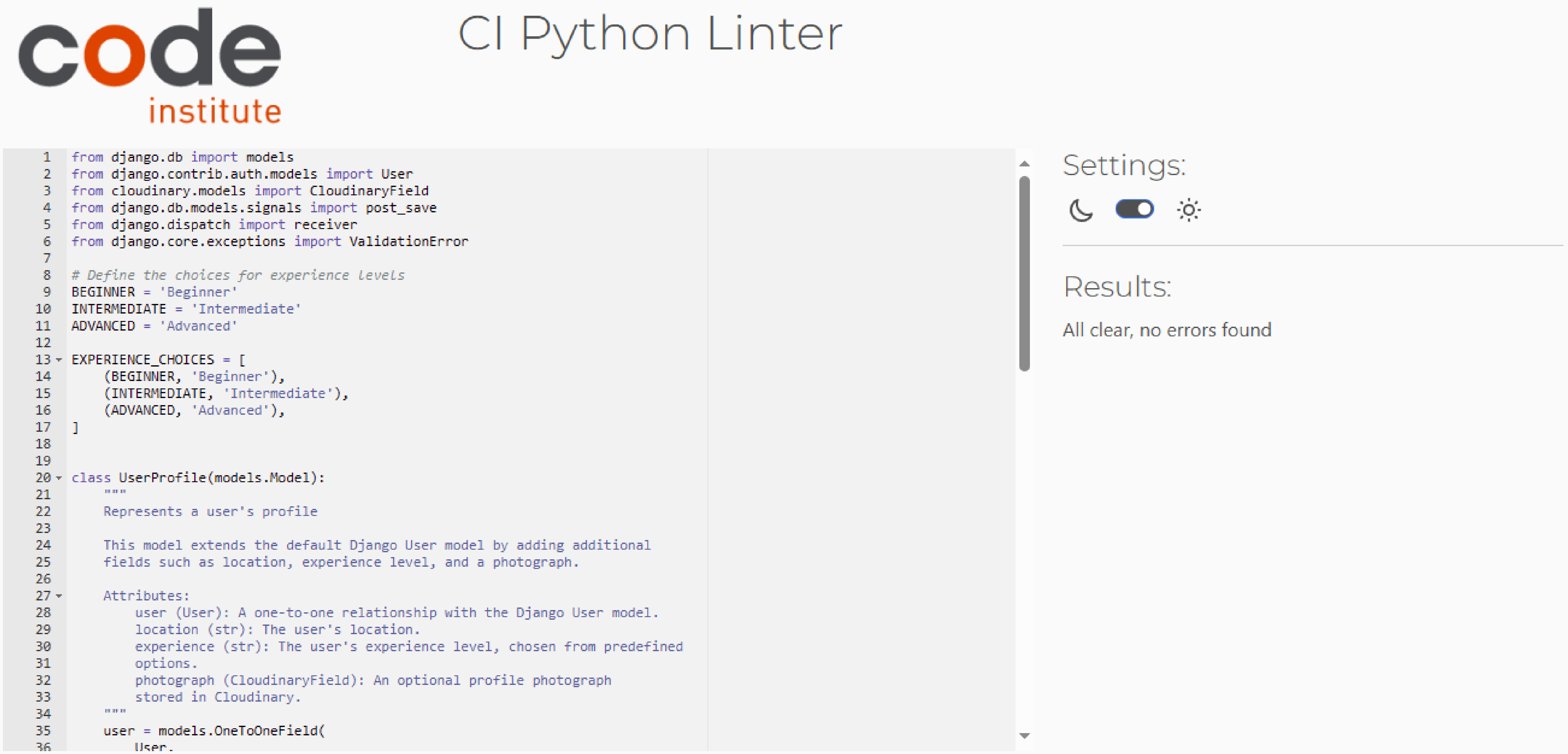
urls.py



```
"""
URL configuration for the register app

This module defines the URL patterns for the user registration and profile
update views. It also specifies custom error handlers for 404 and 500 HTTP
errors.
"""
from django.urls import path
from register import views as registration

handler404 = 'craftr.views.custom_404'
handler500 = 'craftr.views.custom_500'

urlpatterns = [
    path('', registration.register_user, name='register'),
    path(
        'update_profile/',
        registration.update_profile,
        name="update_profile",
    ),
]
```

All clear, no errors found

models.py



CI Python Linter

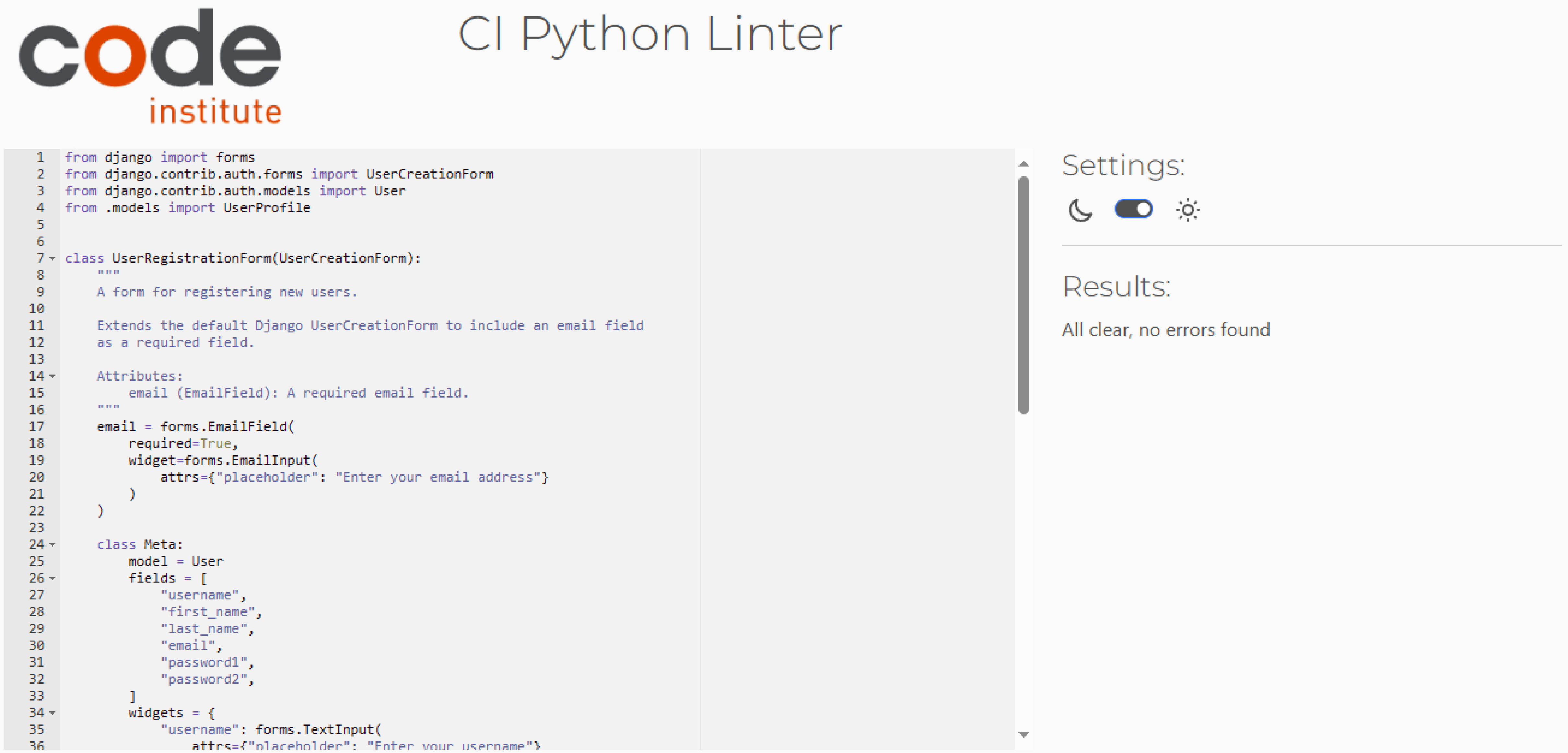
```
from django.db import models
from django.contrib.auth.models import User
from cloudinary.models import CloudinaryField
from django.db.models.signals import post_save
from django.dispatch import receiver
from django.core.exceptions import ValidationError

# Define the choices for experience levels
BEGINNER = 'Beginner'
INTERMEDIATE = 'Intermediate'
ADVANCED = 'Advanced'

EXPERIENCE_CHOICES = [
    (BEGINNER, 'Beginner'),
    (INTERMEDIATE, 'Intermediate'),
    (ADVANCED, 'Advanced'),
]


class UserProfile(models.Model):
    """
    Represents a user's profile

    This model extends the default Django User model by adding additional
    fields such as location, experience level, and a photograph.

    Attributes:
        user (User): A one-to-one relationship with the Django User model.
        location (str): The user's location.
        experience (str): The user's experience level, chosen from predefined
        options.
        photograph (CloudinaryField): An optional profile photograph
        stored in Cloudinary.
    """
    user = models.OneToOneField(
        User,
```

Settings:

Results:

All clear, no errors found

forms.py



CI Python Linter

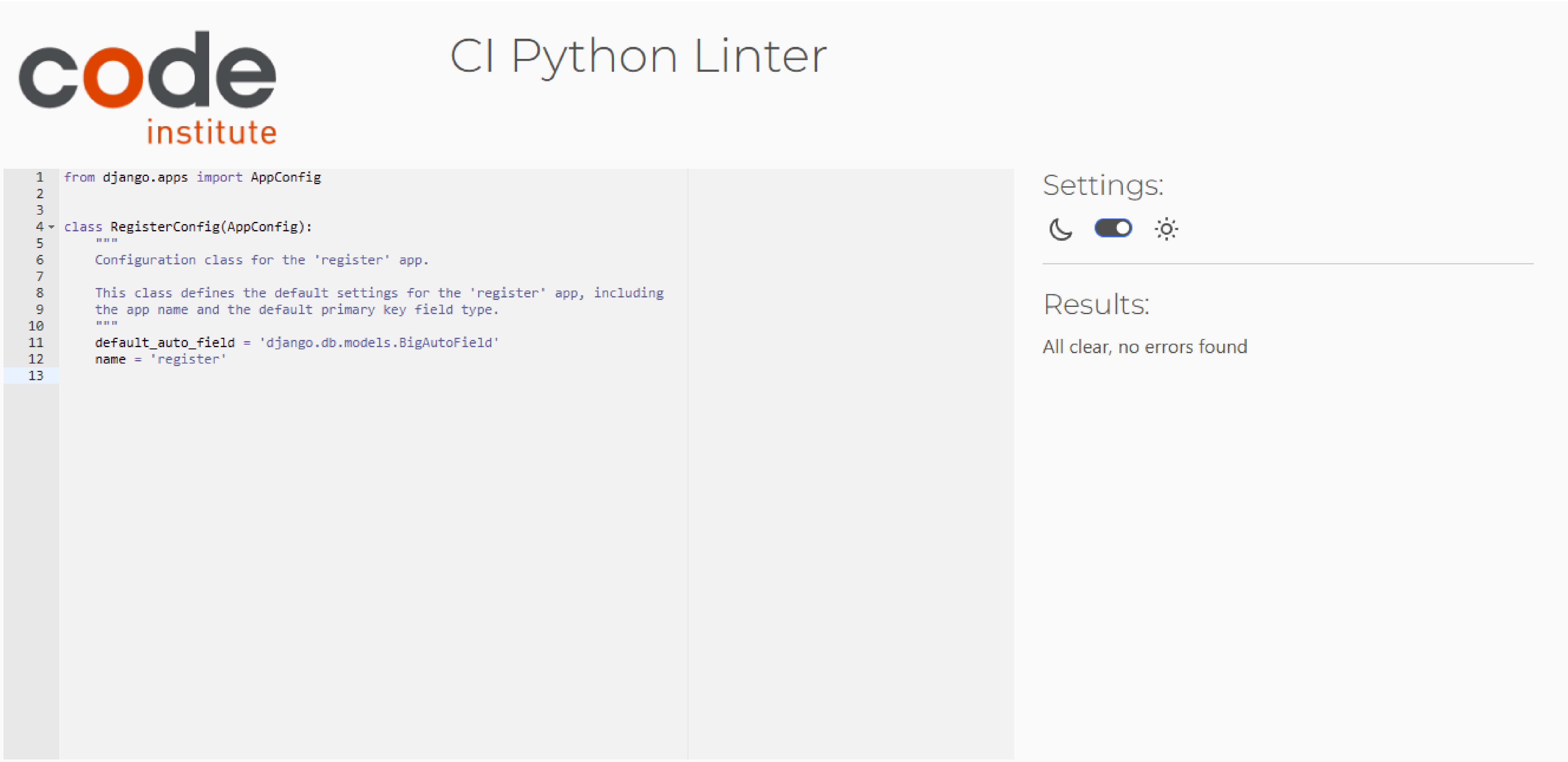
```
from django import forms
from django.contrib.auth.forms import UserCreationForm
from django.contrib.auth.models import User
from .models import UserProfile


class UserRegistrationForm(UserCreationForm):
    """
    A form for registering new users.

    Extends the default Django UserCreationForm to include an email field
    as a required field.

    Attributes:
        email (EmailField): A required email field.
    """
    email = forms.EmailField(
        required=True,
        widget=forms.EmailInput(
            attrs={"placeholder": "Enter your email address"}
        )
    )

    class Meta:
        model = User
        fields = [
            "username",
            "first_name",
            "last_name",
            "email",
            "password1",
            "password2",
        ]
        widgets = {
            "username": forms.TextInput(
                attrs={"placeholder": "Enter your username"}
```

Settings:

Results:

All clear, no errors found

apps.py

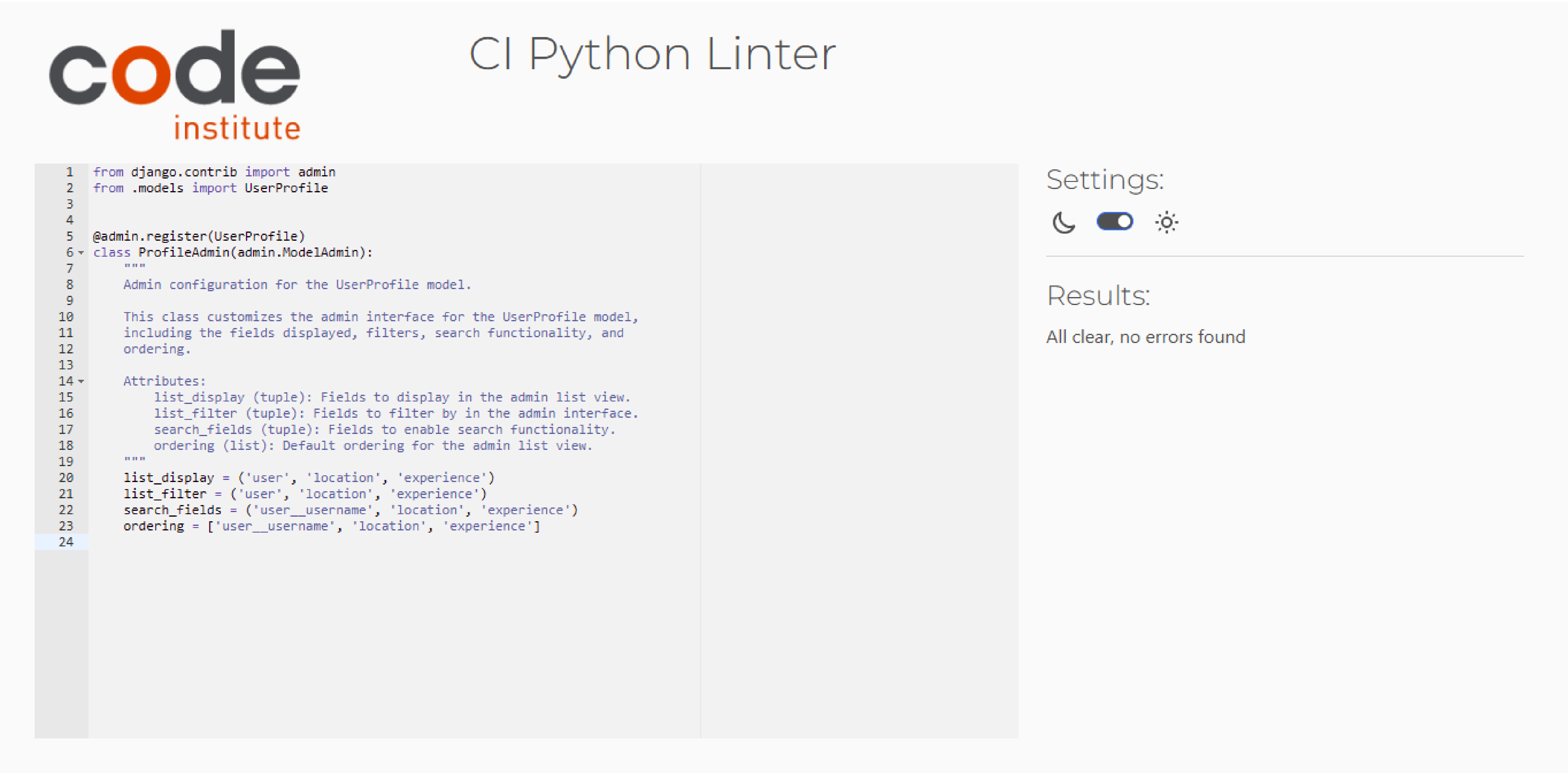


```python
from django.apps import AppConfig


class RegisterConfig(AppConfig):
    """
    Configuration class for the 'register' app.

    This class defines the default settings for the 'register' app, including
    the app name and the default primary key field type.
    """
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'register'

```

Results:

All clear, no errors found

admin.py



CI Python Linter

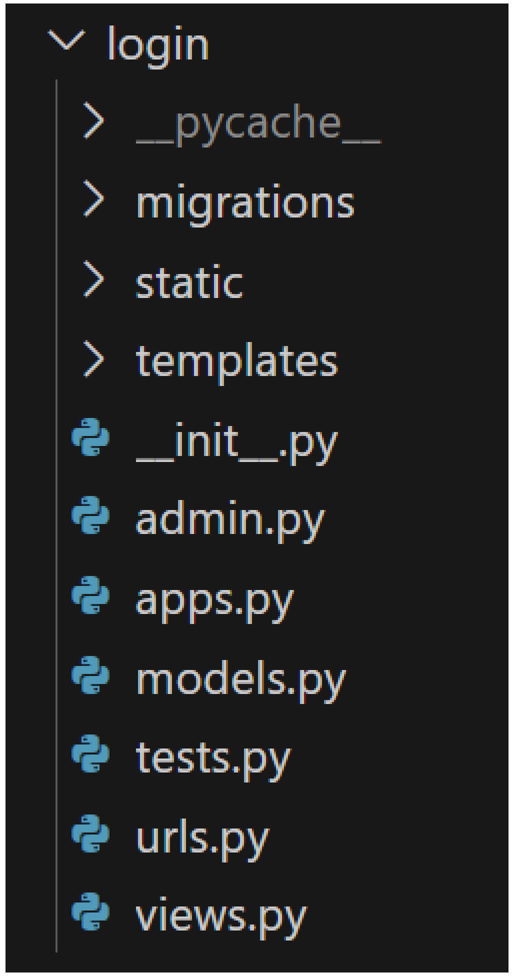
```
from django.contrib import admin
from .models import UserProfile


@admin.register(UserProfile)
class ProfileAdmin(admin.ModelAdmin):
    """
    Admin configuration for the UserProfile model.

    This class customizes the admin interface for the UserProfile model,
    including the fields displayed, filters, search functionality, and
    ordering.

    Attributes:
        list_display (tuple): Fields to display in the admin list view.
        list_filter (tuple): Fields to filter by in the admin interface.
        search_fields (tuple): Fields to enable search functionality.
        ordering (list): Default ordering for the admin list view.
    """
    list_display = ('user', 'location', 'experience')
    list_filter = ('user', 'location', 'experience')
    search_fields = ('user__username', 'location', 'experience')
    ordering = ['user__username', 'location', 'experience']

```

Settings:

Results:

All clear, no errors found

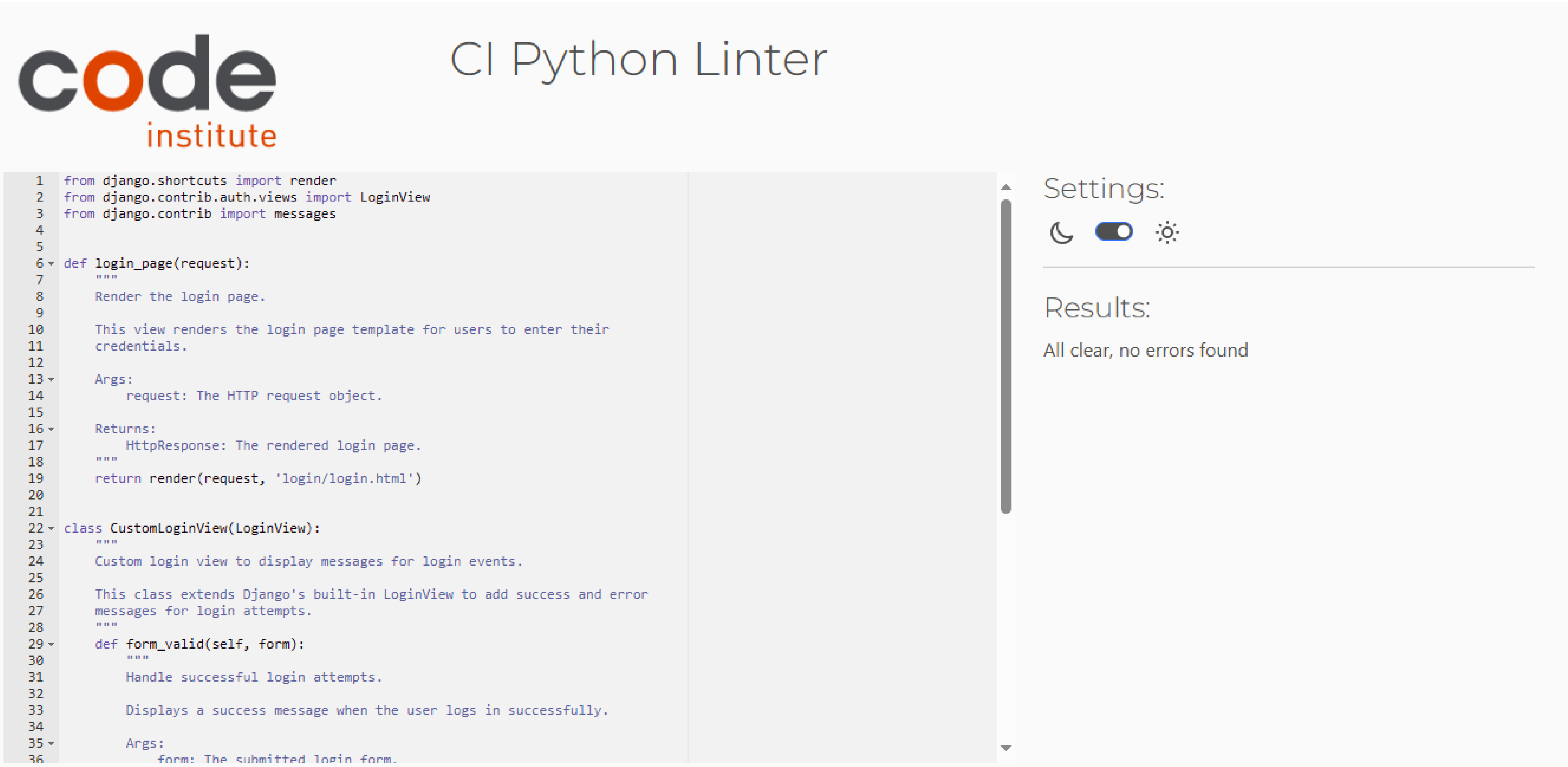
App - Login



login
- __pycache__
- migrations
- static
- templates
- __init__.py
- admin.py
- apps.py
- models.py
- tests.py
- urls.py
- views.py

Python files created / amended were *views.py, urls.py, and apps.py*

views.py



CI Python Linter

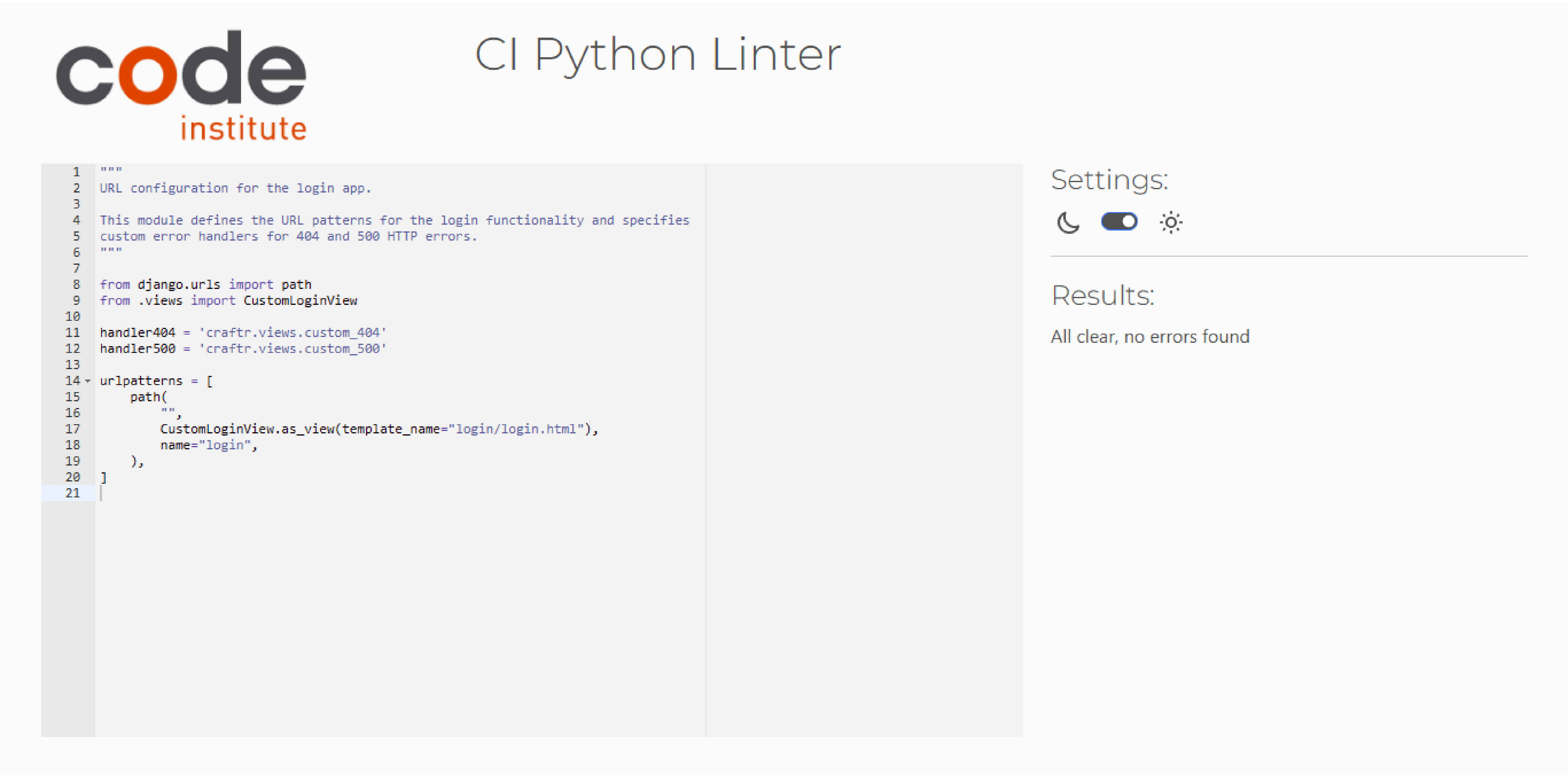
```
from django.shortcuts import render
from django.contrib.auth.views import LoginView
from django.contrib import messages


def login_page(request):
    """
    Render the login page.

    This view renders the login page template for users to enter their
    credentials.

    Args:
        request: The HTTP request object.

    Returns:
        HttpResponse: The rendered login page.
    """
    return render(request, 'login/login.html')


class CustomLoginView(LoginView):
    """
    Custom login view to display messages for login events.

    This class extends Django's built-in LoginView to add success and error
    messages for login attempts.
    """
    def form_valid(self, form):
        """
        Handle successful login attempts.

        Displays a success message when the user logs in successfully.

        Args:
            form: The submitted login form.
```

Settings:

Results:

All clear, no errors found

urls.py


CI Python Linter

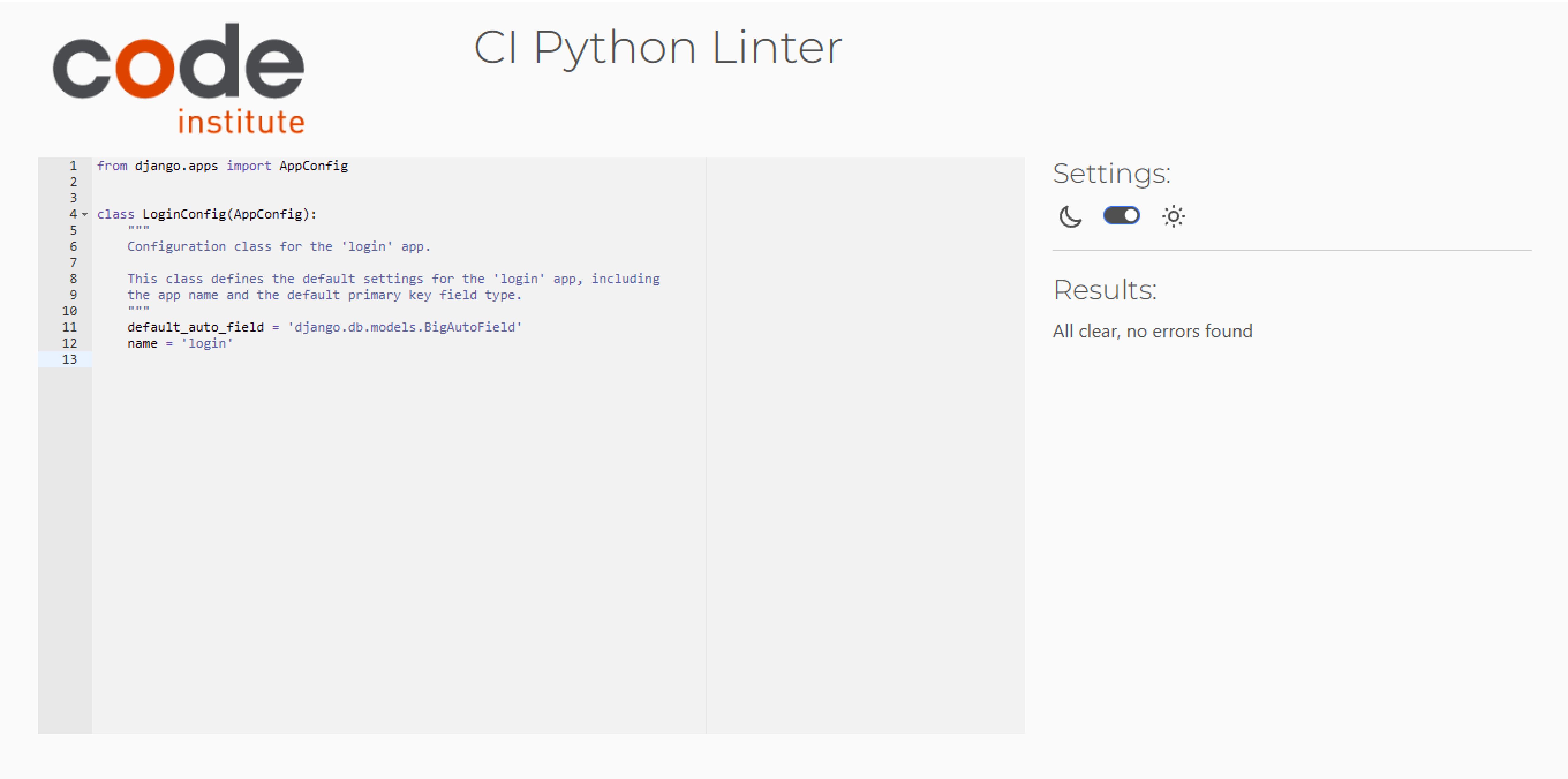
```
"""
URL configuration for the login app.

This module defines the URL patterns for the login functionality and specifies
custom error handlers for 404 and 500 HTTP errors.
"""

from django.urls import path
from .views import CustomLoginView

handler404 = 'craftr.views.custom_404'
handler500 = 'craftr.views.custom_500'

urlpatterns = [
    path(
        "",
        CustomLoginView.as_view(template_name="login/login.html"),
        name="login",
    ),
]
```

Settings:

Results:

All clear, no errors found

apps.py



CI Python Linter

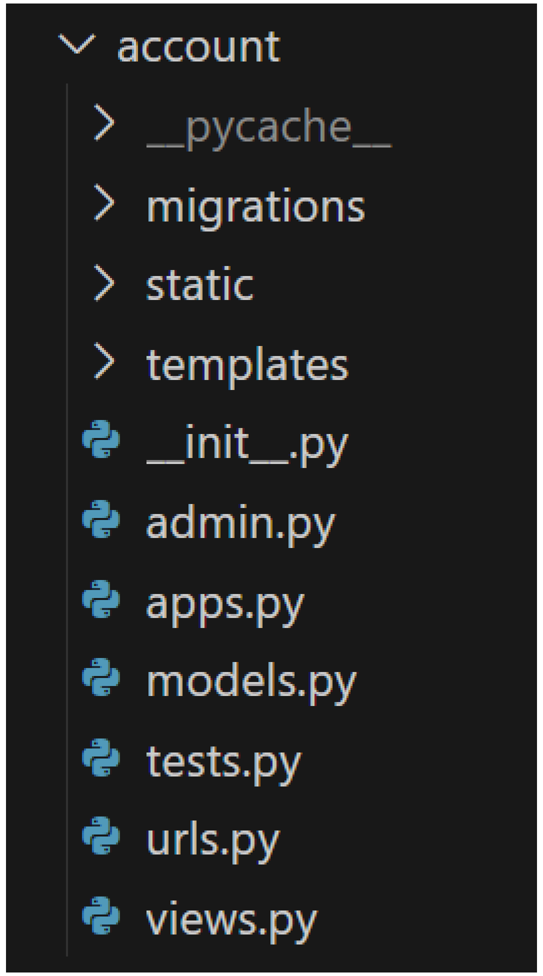
```
from django.apps import AppConfig


class LoginConfig(AppConfig):
    """
    Configuration class for the 'login' app.

    This class defines the default settings for the 'login' app, including
    the app name and the default primary key field type.
    """
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'login'
```

Settings:

Results:

All clear, no errors found

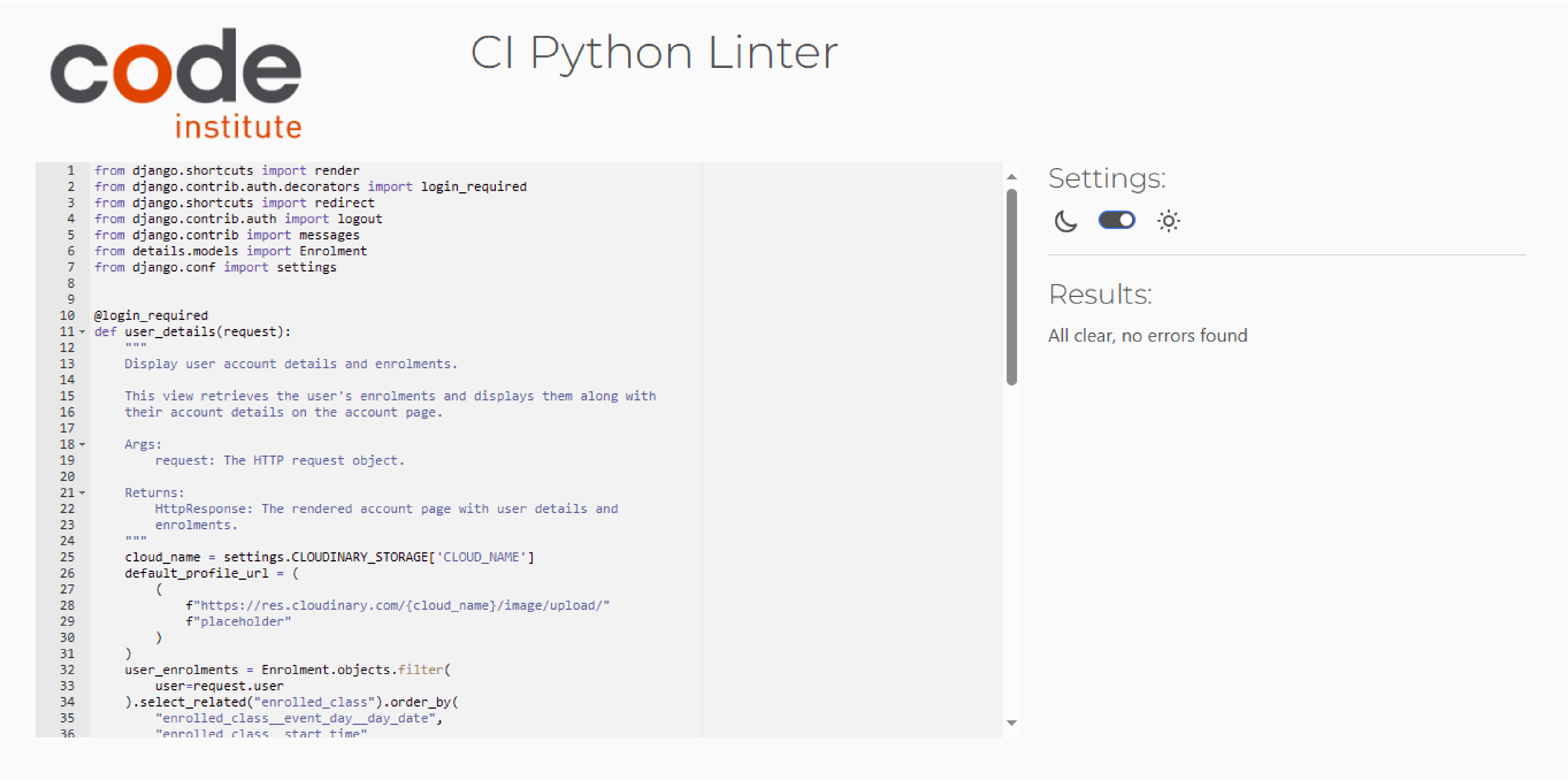
App - Account



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

Python files created / amended were *views.py, urls.py, and apps.py*

views.py


CI Python Linter

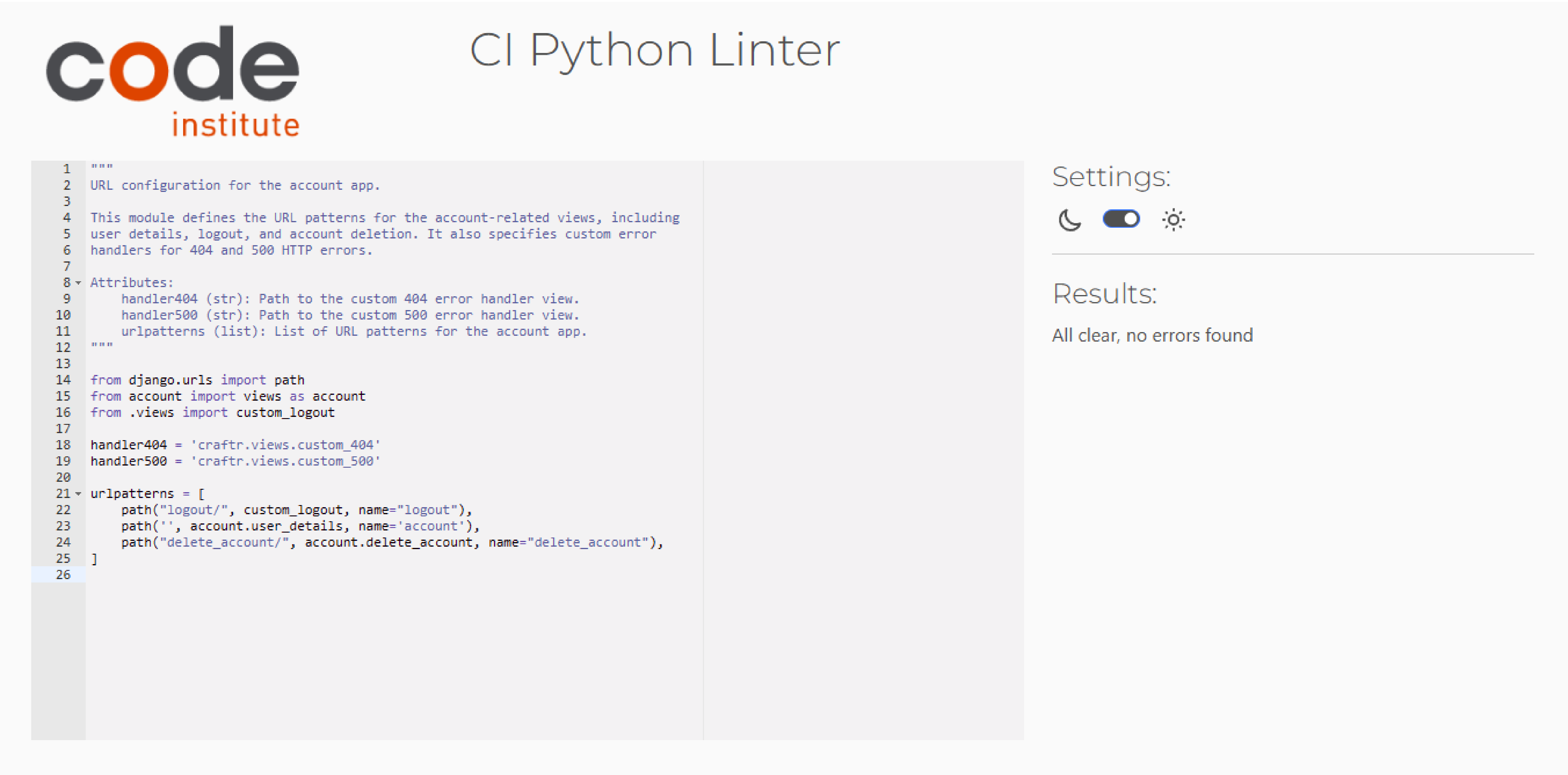
```
from django.shortcuts import render
from django.contrib.auth.decorators import login_required
from django.shortcuts import redirect
from django.contrib.auth import logout
from django.contrib import messages
from details.models import Enrolment
from django.conf import settings


@login_required
def user_details(request):
    """
    Display user account details and enrolments.

    This view retrieves the user's enrolments and displays them along with
    their account details on the account page.

    Args:
        request: The HTTP request object.

    Returns:
        HttpResponse: The rendered account page with user details and
        enrolments.
    """
    cloud_name = settings.CLOUDINARY_STORAGE['CLOUD_NAME']
    default_profile_url = (
        (
            f"https://res.cloudinary.com/{cloud_name}/image/upload/"
            f"placeholder"
        )
    )
    user_enrolments = Enrolment.objects.filter(
        user=request.user
    ).select_related("enrolled_class").order_by(
        "enrolled_class__event_day__day_date",
        "enrolled_class__start_time"
```

Settings:

Results:

All clear, no errors found

urls.py



CI Python Linter

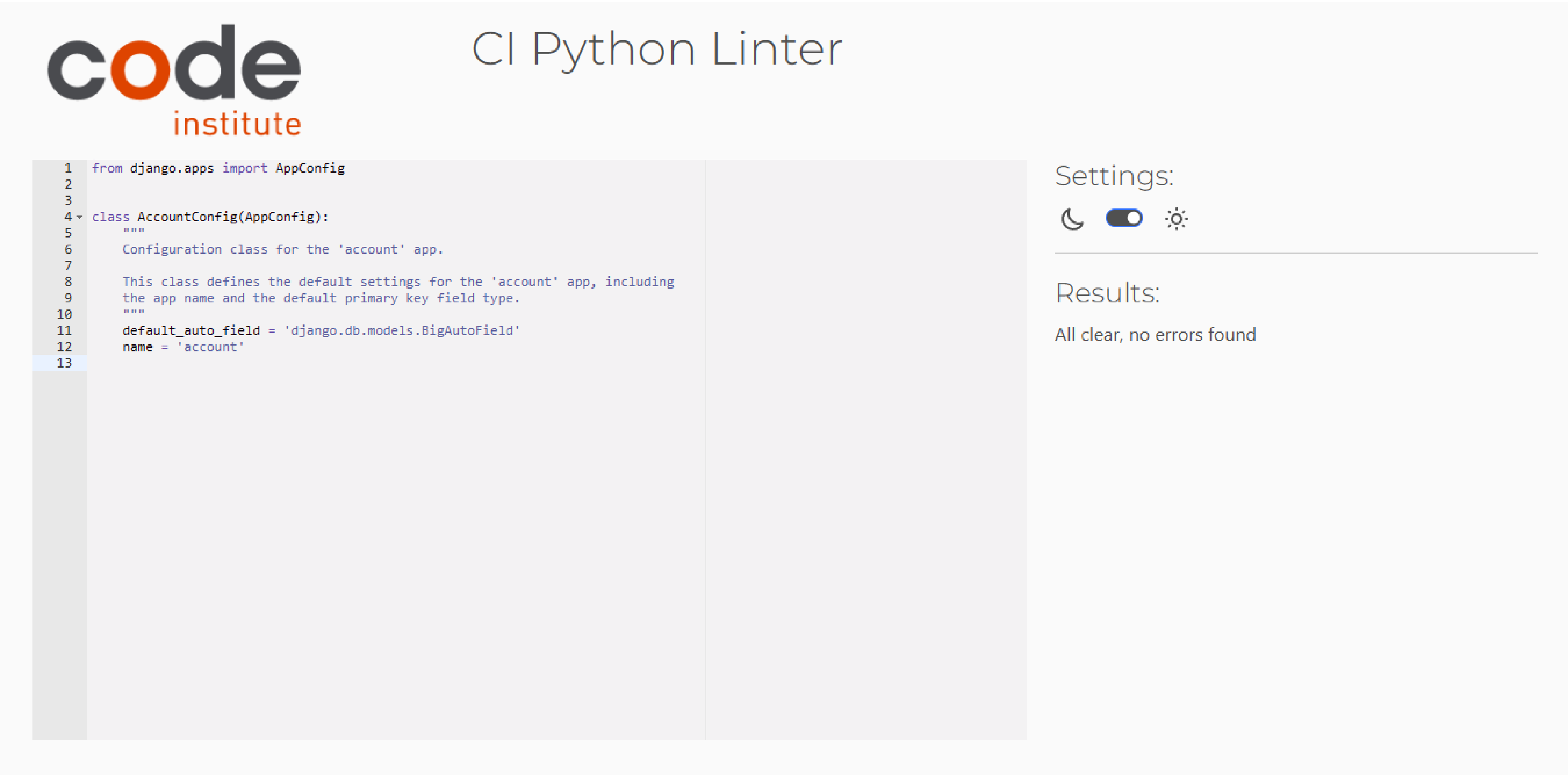
```
"""
URL configuration for the account app.

This module defines the URL patterns for the account-related views, including
user details, logout, and account deletion. It also specifies custom error
handlers for 404 and 500 HTTP errors.

Attributes:
    handler404 (str): Path to the custom 404 error handler view.
    handler500 (str): Path to the custom 500 error handler view.
    urlpatterns (list): List of URL patterns for the account app.
"""

from django.urls import path
from account import views as account
from .views import custom_logout

handler404 = 'craftr.views.custom_404'
handler500 = 'craftr.views.custom_500'

urlpatterns = [
    path("logout/", custom_logout, name="logout"),
    path('', account.user_details, name='account'),
    path("delete_account/", account.delete_account, name="delete_account"),
]
```

Settings:

Results:

All clear, no errors found

apps.py



```python
from django.apps import AppConfig


class AccountConfig(AppConfig):
    """
    Configuration class for the 'account' app.

    This class defines the default settings for the 'account' app, including
    the app name and the default primary key field type.
    """
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'account'

```

Results:

All clear, no errors found