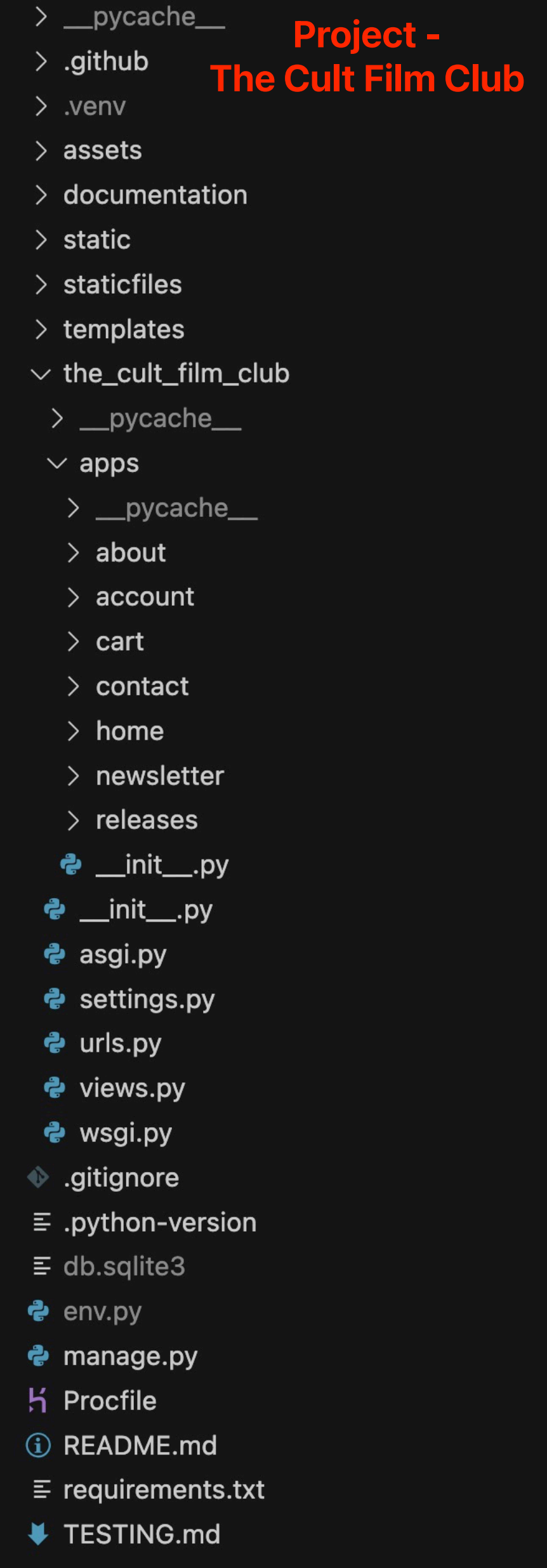

Project -
The Cult Film Club
__pycache__
.github
.venv
assets
documentation
static
staticfiles
templates
the_cult_film_club
__pycache__
apps
__pycache__
about
account
cart
contact
home
newsletter
releases
__init__.py
__init__.py
asgi.py
settings.py
urls.py
views.py
wsgi.py
.gitignore
.python-version
db.sqlite3
env.py
manage.py
Procfile
README.md
requirements.txt
TESTING.md

CI Python Linter

asgi.py

```python
import os

from django.core.asgi import get_asgi_application

os.environ.setdefault("DJANGO_SETTINGS_MODULE", "the_cult_film_club.settings")

application = get_asgi_application()
```

Settings:

Results:

All clear, no errors found

CI Python Linter

settings.py

```python
from pathlib import Path
import os
import dj_database_url
from decimal import Decimal

# Load environment variables from env.py if present (for local development)
if os.path.isfile("env.py"):
    import env  # noqa

# Debug mode (should be False in production)
DEBUG = False

# Email backend configuration for sending emails via Fastmail SMTP
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")
DEFAULT_FROM_EMAIL = "tcfc@dominicfrancis.co.uk"

# Custom user signup form for django-allauth
ACCOUNT_FORMS = {
    'signup': 'the_cult_film_club.apps.account.forms.AccountSignupForm',
}

# Allow login by email or username
ACCOUNT_LOGIN_METHODS = {"email", "username"}

# Fields required for signup (with * indicating required)
ACCOUNT_SIGNUP_FIELDS = [
```

Settings:

Results:

All clear, no errors found

CI Python Linter

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 django.shortcuts import render
from django.core.exceptions import PermissionDenied, SuspiciousOperation


# Custom error handlers
def custom_bad_request(request, exception):
    return render(request, "error_pages/bad_request.html", status=400)


def custom_permission_denied(request, exception):
    return render(request, "error_pages/permission_denied.html", status=403)


def custom_page_not_found(request, exception):
    return render(request, "error_pages/page_not_found.html", status=404)


def custom_server_error(request):
    return render(request, "error_pages/server_error.html", status=500)


# Test error views (for development/debug only)

def raise_server_error(request):
    raise Exception("Test 500 error")
```

Settings:

Results:

All clear, no errors found

CI Python Linter

views.py

```python
def custom_permission_denied(request, exception=None):
    """
    Render the custom 403 error page.

    This view handles permission denied errors, and renders a custom 403 error
    page.

    Args:
        request: The HTTP request object.
        exception: The exception that triggered the 403 error (optional).

    Returns:
        HttpResponse: The rendered 403 error page with a 403 status code.
    """
    return render(request, "error_pages/permission_denied.html", status=403)


def bad_request(request, exception=None):
    """
    Render the custom 400 error page.

    This view handles bad request errors, and renders a custom 400 error page.

    Args:
        request: The HTTP request object.
        exception: The exception that triggered the 400 error (optional).
    Returns:
        HttpResponse: The rendered 400 error page with a 400 status code.
    """
    return render(request, "error_pages/bad_request.html", status=400)

```

Settings:

Results:

All clear, no errors found

CI Python Linter

wsgi.py

```
"""
WSGI config for the_cult_film_club 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/5.2/howto/deployment/wsgi/
"""

import os

from django.core.wsgi import get_wsgi_application

os.environ.setdefault("DJANGO_SETTINGS_MODULE", "the_cult_film_club.settings")

application = get_wsgi_application()
```

Settings:

Results:

All clear, no errors found

CI Python Linter

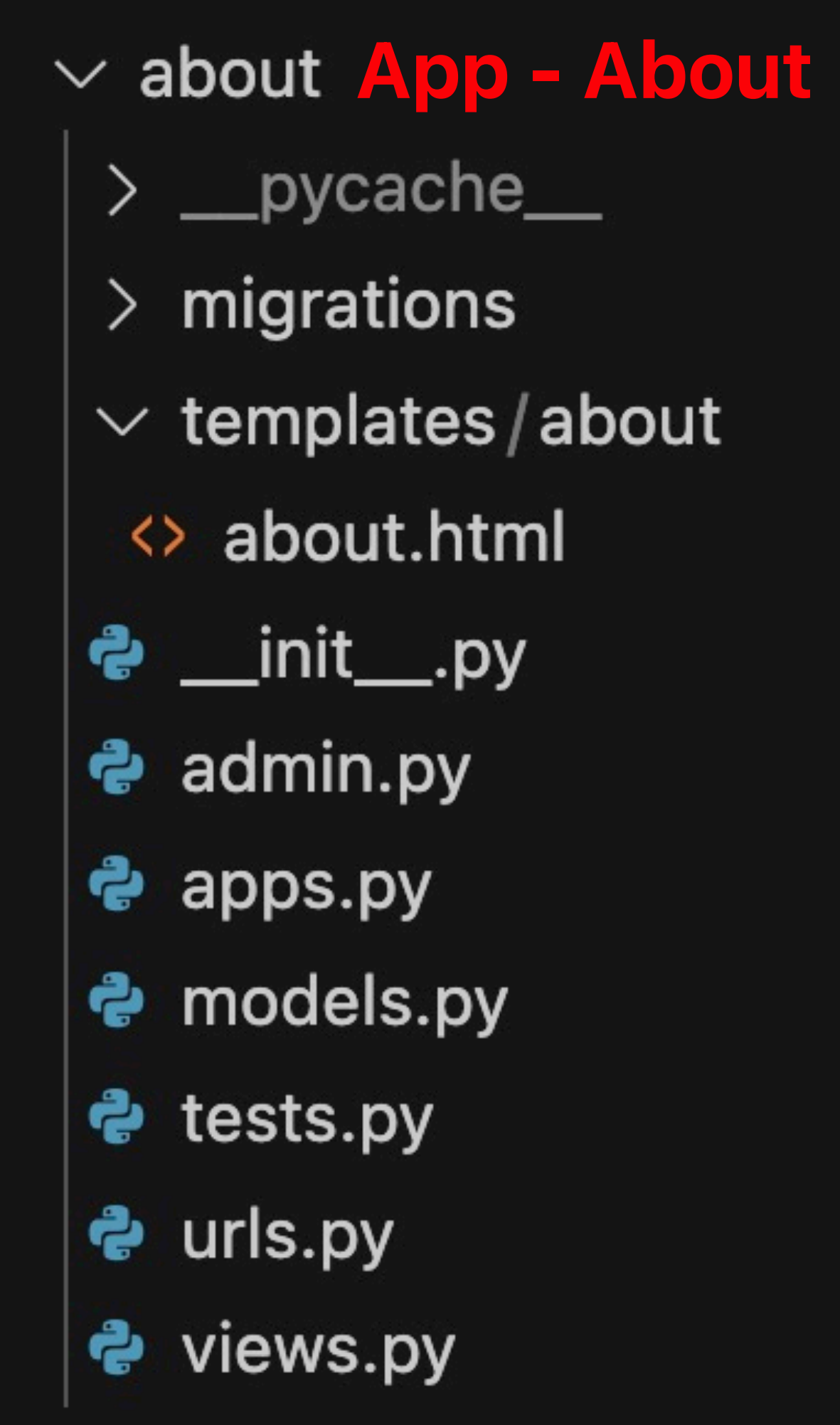
manage.py

```python
#!/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", "the_cult_film_club.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


about
App - About
__pycache__
migrations
templates/about
about.html
__init__.py
admin.py
apps.py
models.py
tests.py
urls.py
views.py

CI Python Linter

apps.py

```python
from django.apps import AppConfig


class AboutConfig(AppConfig):
    """
    Configuration for the About app
    """
    default_auto_field = "django.db.models.BigAutoField"
    name = "the_cult_film_club.apps.about"
    verbose_name = "About"

```

Settings:

Results:

All clear, no errors found

CI Python Linter

urls.py

```python
"""URL configuration for the About app"""

from django.urls import path
from . import views

urlpatterns = [
    path('', views.about, name='about'),
]

```

Settings:

Results:

All clear, no errors found

CI Python Linter

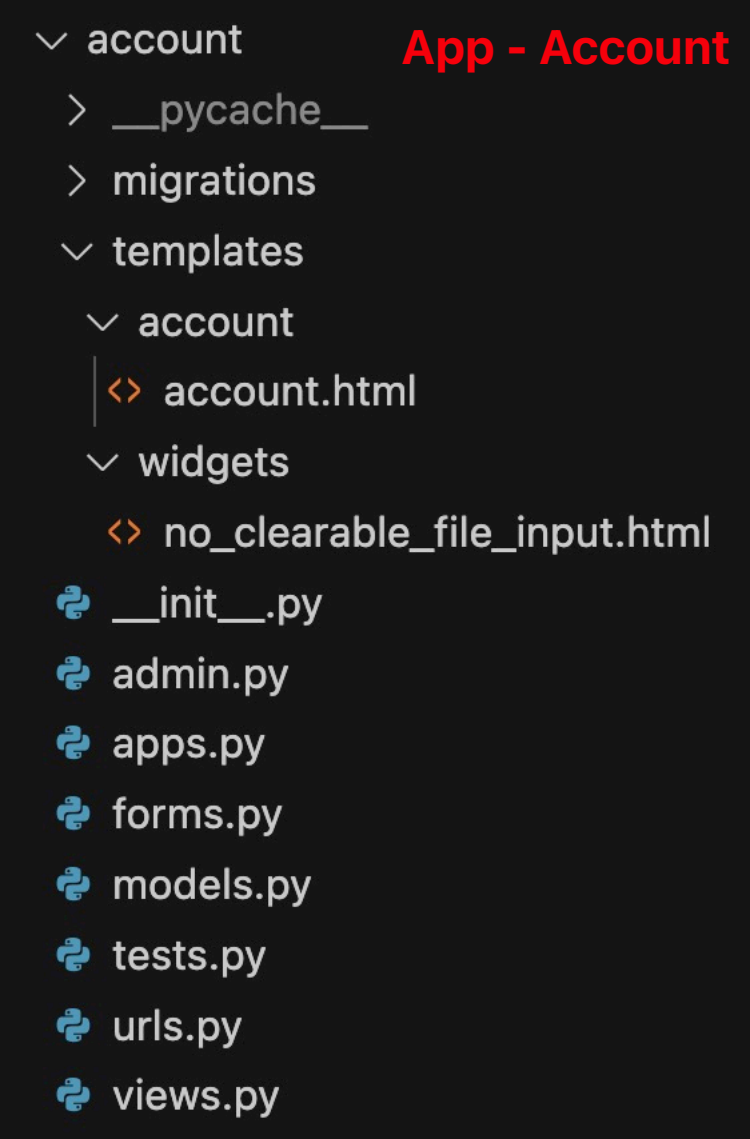
views.py

```python
from django.shortcuts import render


def about(request):
    """
    Render the About page
    """
    return render(request, "about/about.html")

```

Settings:

Results:

All clear, no errors found


account
App - Account
__pycache__
migrations
templates
account
account.html
widgets
no_clearable_file_input.html
__init__.py
admin.py
apps.py
forms.py
models.py
tests.py
urls.py
views.py

CI Python Linter

admin.py

```python
from django.contrib import admin
from django import forms
from django.utils.html import format_html
from django_ckeditor_5.widgets import CKEditor5Widget

from .models import Profile, Address, Wishlist, WishlistItem


class WishlistItemAdminForm(forms.ModelForm):
    """
    Custom form for WishlistItem with CKEditor widget for rich-text notes
    """
    class Meta:
        model = WishlistItem
        fields = "__all__"
        widgets = {
            "notes": CKEditor5Widget(config_name="extends"),
        }


@admin.register(Profile)
class ProfileAdmin(admin.ModelAdmin):
    """
    Admin interface for user profiles
    """
    list_display = ['user']
    list_filter = ['user']
    search_fields = ['user__username']
    ordering = ['user__username']
```

Settings:

Results:

All clear, no errors found

CI Python Linter

apps.py

```python
from django.apps import AppConfig


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

    This sets a custom label for internal Django app registry and a
    user-friendly verbose name for the admin interface
    """
    default_auto_field = "django.db.models.BigAutoField"
    name = "the_cult_film_club.apps.account"

    # Custom label used to avoid app name conflicts in complex projects
    label = "the_cult_film_club_account"

    # Display name for the app in the Django admin
    verbose_name = "User Profiles"
```

Settings:

Results:

All clear, no errors found

CI Python Linter

forms.py

```python
from allauth.account.forms import SignupForm
from django import forms
from django_countries.fields import CountryField
from django_countries.widgets import CountrySelectWidget

from .models import Profile, Address, WishlistItem, PriorityLevel


class AccessibleCountrySelectWidget(CountrySelectWidget):
    """
    Custom CountrySelectWidget that removes invalid placeholder attribute
    and adds proper alt text to flag images
    """
    def __init__(self, attrs=None):
        # Remove placeholder from attrs if present
        if attrs and 'placeholder' in attrs:
            attrs = attrs.copy()
            del attrs['placeholder']
        super().__init__(attrs)

    def render(self, name, value, attrs=None, renderer=None):
        html = super().render(name, value, attrs, renderer)
        # Add alt attribute to the flag image
        html = html.replace(
            'src="/static/flags/__.gif"',
            'src="/static/flags/__.gif" alt="Country flag"'
        )
        return html


class AccountSignupForm(SignupForm):
```

Settings:

Results:

All clear, no errors found

CI Python Linter

models.py

```python
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_ckeditor_5.fields import CKEditor5Field
from django_countries.fields import CountryField


class Profile(models.Model):
    """
    Stores user profile information, including photograph and addresses
    """
    user = models.OneToOneField(
        User,
        on_delete=models.CASCADE,
        related_name="profile"
    )
    photograph = CloudinaryField(
        'image',
        default='placeholder',
        blank=True,
        null=False
    )

    def __str__(self):
        return (
            f"This is {self.user.username}'s profile"
            if self.user
            else "No user assigned to this profile"
        )
```

Settings:

Results:

All clear, no errors found

CI Python Linter

urls.py

```python
"""URL configuration for the Account app"""

from django.urls import path
from . import views
from the_cult_film_club.apps.cart import views as cart_views


urlpatterns = [
    path("user_profile/", views.user_profile, name="user_profile"),
    path('add/<int:item_id>/', cart_views.add_to_cart, name='add_to_cart')
]
```

Settings:

Results:

All clear, no errors found

CI Python Linter

views.py

```python
from django.shortcuts import render, get_object_or_404, redirect
from django.contrib import messages
from .models import Profile, Address, Wishlist, WishlistItem
from the_cult_film_club.apps.cart.models import Order
from .forms import ProfilePhotoForm, AddressForm, WishlistItemForm
from the_cult_film_club.apps.releases.models import Releases
from django.contrib.auth.decorators import login_required
from django.http import HttpRequest, HttpResponse
from django.contrib.auth import logout
import cloudinary.uploader


@login_required
def user_profile(request: HttpRequest) -> HttpResponse:
    """
    Display and manage the user's profile, addresses, wishlist, and orders,
    including editing wishlist items.
    """
    user_profile = get_object_or_404(Profile, user=request.user)
    orders = (
        Order.objects
        .filter(user_profile__user=request.user)
        .order_by('-date')
    )
    addresses = Address.objects.filter(user=request.user)
    wishlists = Wishlist.objects.filter(user=request.user)
    selected_wishlist_id = (
        request.GET.get("wishlist") or request.POST.get("wishlist")
    )
    selected_wishlist = (
        wishlists.filter(id=selected_wishlist_id).first()
```

Settings:

Results:

All clear, no errors found

App - Cart

- cart
  - __pycache__
  - migrations
  - templates/cart
    - cart.html
    - checkout_success.html
    - checkout.html
    - conf_body.txt
    - conf_subject.txt
    - delete_discount_code.html
    - discount_codes.html
    - edit_discount_code.html
    - order_detail.html
  - templatetags
    - __pycache__
    - __init__.py
    - cart_tools.py
  - __init__.py
  - admin.py
  - apps.py
  - contexts.py
  - forms.py
  - models.py
  - signals.py
  - tests.py
  - urls.py
  - views.py
  - webhook_handler.py
  - webhooks.py

CI Python Linter

admin.py

```python
from django.contrib import admin
from .models import Order, OrderLineItem, DiscountCode


class OrderLineItemAdminInline(admin.TabularInline):
    """
    Inline admin interface for order line items,
    displayed within the Order admin page.
    """
    model = OrderLineItem
    readonly_fields = ('lineitem_total',)
    extra = 0  # Prevent extra empty forms


@admin.register(Order)
class OrderAdmin(admin.ModelAdmin):
    """
    Admin interface for Orders,
    including inline display of related order line items.
    """
    inlines = (OrderLineItemAdminInline,)

    readonly_fields = (
        'order_number', 'date',
        'delivery_cost', 'subtotal',
        'total', 'original_bag',
        'stripe_pid',
    )

    fields = (
        'order_number', 'date',
```

Settings:

Results:

All clear, no errors found

CI Python Linter

apps.py

```python
from django.apps import AppConfig


class CartConfig(AppConfig):
    """
    Configuration for the Cart app,
    including registration of signals on app ready.
    """
    default_auto_field = "django.db.models.BigAutoField"
    name = "the_cult_film_club.apps.cart"
    verbose_name = "Shop"

    def ready(self):
        """
        Import signals to ensure signal handlers are connected
        when the app is ready.
        """
        # Importing signals here to register signal handlers
        from the_cult_film_club.apps.cart import signals  # noqa: F401

```

Settings:

Results:

All clear, no errors found

CI Python Linter

contexts.py

```python
from decimal import Decimal
from django.conf import settings
from django.shortcuts import get_object_or_404
from the_cult_film_club.apps.releases.models import Releases


def purchases(request):
    """
    Build a detailed cart context dictionary including
    purchases, subtotal, delivery cost, discounts, and totals.

    Also supports sorting by copies_available via GET param
    'sort=copies_available'.
    """
    cart = request.session.get('cart', {})
    discount_code = request.session.get("discount_code", "")
    discount_percent = request.session.get("discount_percent", 0)

    purchases_list = []
    subtotal = Decimal('0.00')
    total_quantity = 0
    sorting_by_copies = request.GET.get('sort') == 'copies_available'

    # Early return for empty cart with default context values
    if not cart:
        return {
            'purchases': [],
            'subtotal': subtotal,
            'total_quantity': total_quantity,
            'delivery_rate': settings.DELIVERY_RATE,
            'delivery': Decimal('0.00'),
```

Settings:

Results:

All clear, no errors found

CI Python Linter

forms.py

```python
from django import forms
from django.utils import timezone
from .models import Order, DiscountCode
from django_countries.fields import CountryField
from django_countries.widgets import CountrySelectWidget


class AccessibleCountrySelectWidget(CountrySelectWidget):
    def __init__(self, attrs=None):
        super().__init__(attrs)
        self.attrs['class'] = 'form-control countryselectwidget form-select'

    def create_option(
        self, name, value, label, selected, index,
        subindex=None, attrs=None
    ):
        option = super().create_option(
            name, value, label, selected, index, subindex, attrs
        )
        return option

    def render(self, name, value, attrs=None, renderer=None):
        html = super().render(name, value, attrs, renderer)
        # Add alt text to flag image
        html = html.replace(
            '<img class="country-select-flag"',
            '<img class="country-select-flag" alt="Country flag"'
        )
        return html


```

Settings:

Results:

All clear, no errors found

CI Python Linter

models.py

```python
import uuid
from decimal import Decimal, ROUND_HALF_UP
from django.db import models
from django.db.models import Sum
from django.conf import settings
from the_cult_film_club.apps.releases.models import Releases
from django_countries.fields import CountryField
from django.utils import timezone


class Order(models.Model):
    """
    Represents a customer's order, including user profile (optional),
    shipping details, payment info, and totals.
    """
    order_number = models.CharField(
        max_length=32,
        null=False,
        editable=False,
        unique=True,
        help_text="Unique order identifier"
    )
    user_profile = models.ForeignKey(
        "the_cult_film_club_account.Profile",
        on_delete=models.CASCADE,
        null=True,
        blank=True,
        related_name='orders',
        help_text="Profile of user who placed the order (optional)"
    )
    full_name = models.CharField(max_length=50, null=False, blank=False)
```

Settings:

Results:

All clear, no errors found

CI Python Linter

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_order_total_on_save(sender, instance, created, **kwargs):
    """
    Signal handler to update the related Order's total
    whenever an OrderLineItem is created or updated.
    """
    instance.order.update_total()


@receiver(post_delete, sender=OrderLineItem)
def update_order_total_on_delete(sender, instance, **kwargs):
    """
    Signal handler to update the related Order's total
    whenever an OrderLineItem is deleted.
    """
    instance.order.update_total()

```

Settings:

Results:

All clear, no errors found

CI Python Linter

urls.py

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

urlpatterns = [
    # Shopping cart main page
    path("", views.shopping_cart, name="cart"),

    # Cart item management
    path("add/<int:item_id>/", views.add_to_cart, name="add_to_cart"),
    path("adjust/<int:item_id>/", views.amend_cart, name="amend_cart"),
    path(
        "remove/<int:item_id>/",
        views.remove_from_cart,
        name="remove_from_cart"
    ),

    # Discount code application
    path("apply_discount/", views.apply_discount, name="apply_discount"),

    # Checkout flow
    path("order/", views.checkout, name="checkout"),
    path(
        "checkout_success/<str:order_number>/",
        views.checkout_success,
        name="checkout_success"
    ),

    # Stripe webhook endpoint
    path("wh/", webhook, name="webhook"),
```

Settings:

Results:

All clear, no errors found

CI Python Linter

views.py

```python
from django.shortcuts import (
    render, redirect, reverse, HttpResponse, get_object_or_404
)
from django.contrib import messages
from django.views.decorators.http import require_POST, require_GET
from django.conf import settings
from django.contrib.auth.decorators import login_required
from django.http import JsonResponse
from decimal import Decimal
import json
import stripe
from the_cult_film_club.apps.releases.models import Releases
from the_cult_film_club.apps.cart.models import DiscountCode, Order
from the_cult_film_club.apps.cart.forms import DiscountCodeForm, OrderForm
from the_cult_film_club.apps.account.models import (
    Profile, Address, Wishlist, WishlistItem
)
from .contexts import purchases
from functools import wraps
from django.core.exceptions import PermissionDenied
from django.contrib.auth.views import redirect_to_login


def superuser_required(view_func):
    """
    Decorator for views that ensures the user is both authenticated and a
    superuser.

    - If the user is not authenticated, redirects them to the login page.
    - If the user is authenticated but not a superuser, raises a
      PermissionDenied (403) error.
```

Settings:

Results:

All clear, no errors found

CI Python Linter

webhook_handler.py

```python
from django.http import HttpResponse
from django.core.mail import send_mail
from django.conf import settings
from django.template.loader import render_to_string
from django.contrib.auth import get_user_model
from django.db import IntegrityError

from .models import Order, OrderLineItem
from the_cult_film_club.apps.releases.models import Releases

import stripe
import json
from decimal import Decimal


class StripeWH_Handler:
    """
    Handle Stripe webhooks for order creation, stock management,
    and email confirmation after successful payments
    """

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

    def _send_confirmation_email(self, order):
        """
        Sends order confirmation email to the user
        """
        cust_email = order.email
        subject = render_to_string(
            'cart/conf_subject.txt',
```

Settings:

Results:

All clear, no errors found

CI Python Linter

webhook.py

```python
from django.http import HttpResponse
from django.conf import settings
from django.views.decorators.csrf import csrf_exempt
from .webhook_handler import StripeWH_Handler
import stripe


@csrf_exempt
def webhook(request):
    """
    Endpoint for handling Stripe webhooks securely

    Verifies the event signature and delegates to the appropriate handler
    based on the event type
    """
    stripe.api_key = settings.STRIPE_SECRET_KEY

    # Retrieve the raw body and Stripe's signature header
    payload = request.body
    sig_header = request.META.get('HTTP_STRIPE_SIGNATURE')
    event = None

    try:
        # Verify and construct the event from payload and signature
        event = stripe.Webhook.construct_event(
            payload=payload,
            sig_header=sig_header,
            secret=settings.STRIPE_WH_SECRET
        )
    except ValueError as e:
        # Invalid payload
```

Settings:

Results:

All clear, no errors found

CI Python Linter

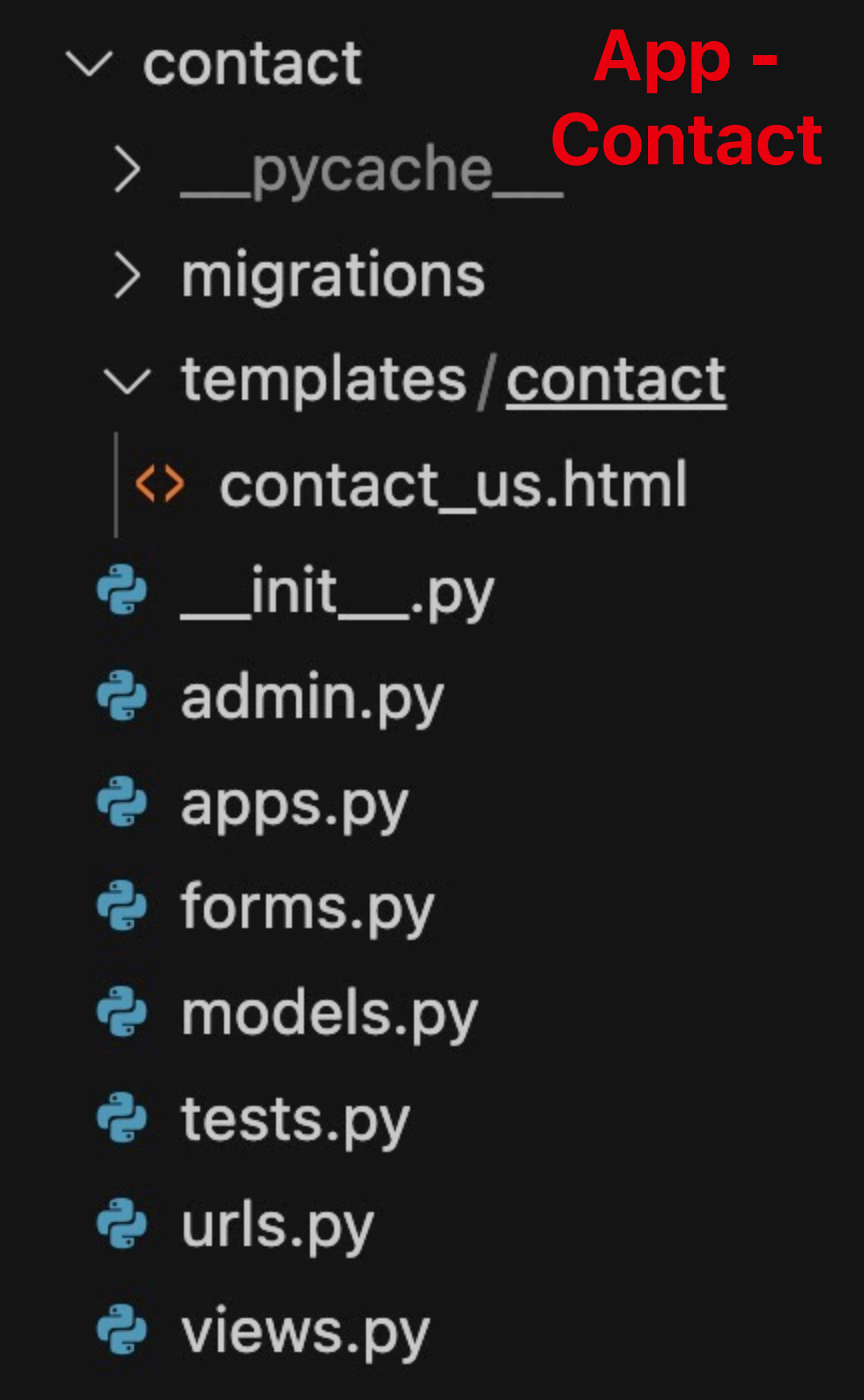
cart_tools.py

```python
from django import template


register = template.Library()


@register.filter(name="calc_subtotal")
def calc_subtotal(price, quantity):
    """Calculate the subtotal for a given price and quantity."""
    return price * quantity

```

Settings:

Results:

All clear, no errors found


contact
App -
Contact
__pycache__
migrations
templates / contact
contact_us.html
__init__.py
admin.py
apps.py
forms.py
models.py
tests.py
urls.py
views.py

CI Python Linter

admin.py

```python
from django.contrib import admin
from django import forms
from django_ckeditor_5.widgets import CKEditor5Widget
from .models import ContactUs


class ContactUsAdminForm(forms.ModelForm):
    class Meta:
        model = ContactUs
        fields = "__all__"
        widgets = {
            "message": CKEditor5Widget(config_name="extends"),
        }


@admin.register(ContactUs)
class ContactAdmin(admin.ModelAdmin):
    """
    Admin interface for ContactUs messages.
    """
    form = ContactUsAdminForm

    list_display = (
        'first_name',
        'last_name',
        'email',
        'short_message',
        'created',
    )
    list_filter = ('first_name', 'last_name', 'email', 'created')
    search_fields = ('first_name', 'last_name', 'email', 'message')
```

Settings:

Results:

All clear, no errors found

CI Python Linter

apps.py

```python
from django.apps import AppConfig


class ContactConfig(AppConfig):
    """
    Configuration for the Contact app.
    """
    default_auto_field = "django.db.models.BigAutoField"
    name = "the_cult_film_club.apps.contact"
    verbose_name = "Contact Form"

```

Settings:

Results:

All clear, no errors found

CI Python Linter

forms.py

```python
from django import forms
from .models import ContactUs


class ContactUsForm(forms.ModelForm):
    """
    Form for users to submit contact messages.
    Uses the ContactUs model fields: first name, last name, email, and message.
    """

    class Meta:
        model = ContactUs
        fields = ['first_name', 'last_name', 'email', 'message']
        widgets = {
            'first_name': forms.TextInput(attrs={
                'placeholder': 'Your first name',
                'class': 'form-control',
            }),
            'last_name': forms.TextInput(attrs={
                'placeholder': 'Your last name',
                'class': 'form-control',
            }),
            'email': forms.EmailInput(attrs={
                'placeholder': 'Your email address',
                'class': 'form-control',
            }),
            'message': forms.Textarea(attrs={
                'placeholder': 'Write your message here...',
                'class': 'form-control',
                'rows': 5,
            }),
```

Settings:

Results:

All clear, no errors found

CI Python Linter

models.py

```python
from django.db import models
from django.core.validators import validate_email
from django_ckeditor_5.fields import CKEditor5Field


class ContactUs(models.Model):
    """
    Model to store messages submitted via the Contact Us form.
    """

    class Meta:
        verbose_name = "Message"
        verbose_name_plural = "Messages"
        ordering = ['-created']  # Show newest messages first by default

    created = models.DateTimeField(
        auto_now_add=True,
        help_text="Timestamp when the message was created"
    )
    first_name = models.CharField(
        max_length=50,
        blank=False,
        null=False,
        help_text="First name of the sender"
    )
    last_name = models.CharField(
        max_length=50,
        blank=False,
        null=False,
        help_text="Last name of the sender"
    )
```

Settings:

Results:

All clear, no errors found

CI Python Linter

urls.py

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

urlpatterns = [
    # Route for the Contact Us page, mapped to the contact_us view
    path('', contact_us, name='contact_us'),
]
```

Settings:

Results:

All clear, no errors found

CI Python Linter

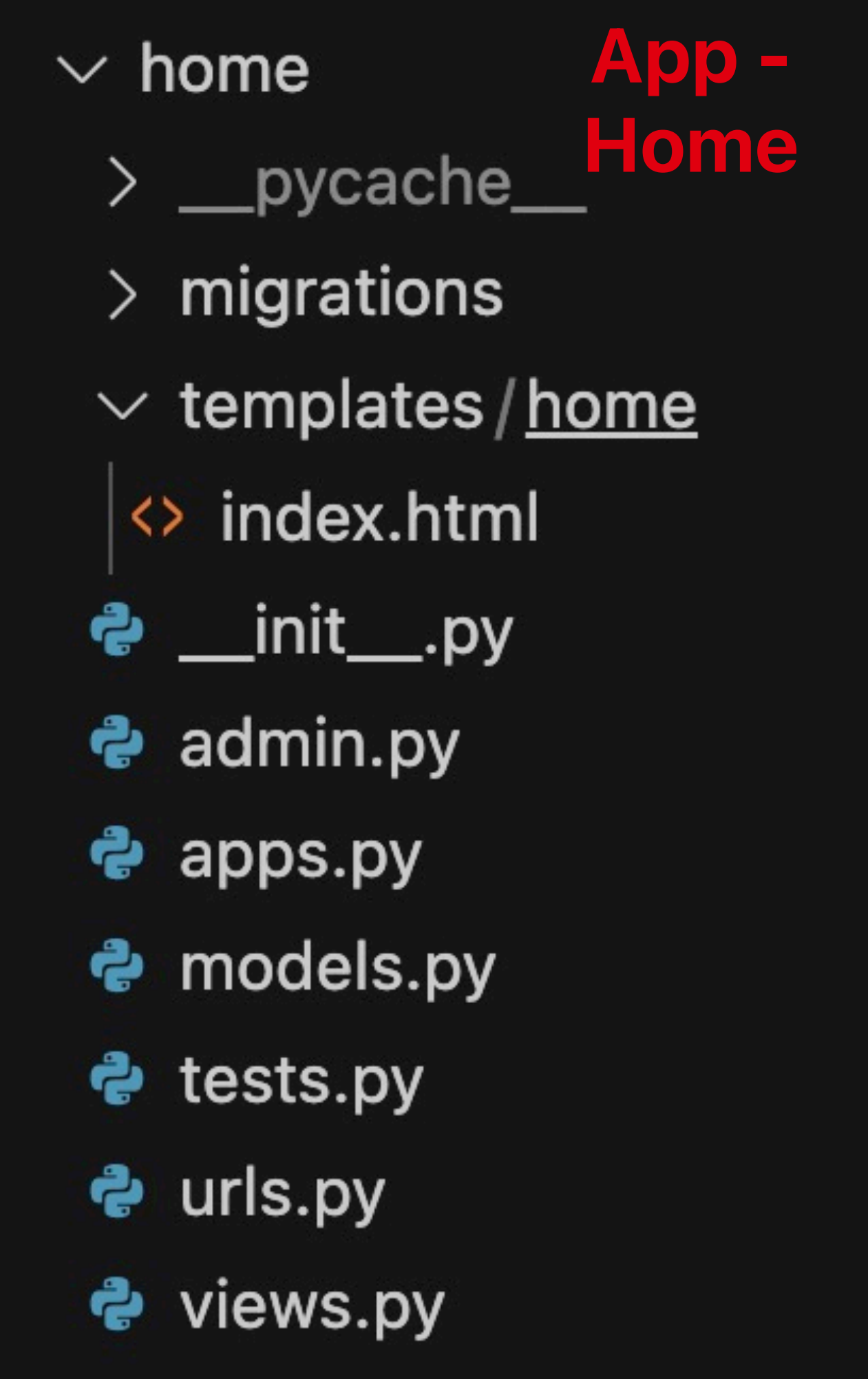
views.py

```python
from django.shortcuts import render, redirect
from django.contrib import messages
from .forms import ContactUsForm


def contact_us(request):
    """
    Handle the Contact Us form submission.

    - On GET request, display an empty contact form.
    - On POST request, validate and save the form.
      If valid, save the submission, show success message,
      and redirect to avoid form resubmission.
    - If the form is invalid, re-render the form with errors.

    Args:
        request (HttpRequest): The HTTP request object.

    Returns:
        HttpResponse: Rendered template with form context.
    """
    if request.method == 'POST':
        form = ContactUsForm(request.POST)
        if form.is_valid():
            # Save the valid form data to the database
            form.save()

            # Add a success message to be displayed to the user
            messages.success(
                request,
                "Thank you for contacting us! We'll respond as soon as we can."
```

Settings:

Results:

All clear, no errors found


App -
Home
home
__pycache__
migrations
templates/home
index.html
__init__.py
admin.py
apps.py
models.py
tests.py
urls.py
views.py

CI Python Linter

apps.py

```python
"""
App configuration for the 'home' app of The Cult Film Club project.
"""

from django.apps import AppConfig


class HomeConfig(AppConfig):
    # Specifies the type of primary key to use for models in this app
    default_auto_field = "django.db.models.BigAutoField"

    # Full Python path to the application
    name = "the_cult_film_club.apps.home"

    # Human-readable name for admin site and introspection tools
    verbose_name = "Home"

```

Settings:

Results:

All clear, no errors found

CI Python Linter

urls.py

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

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

Settings:

Results:

All clear, no errors found

CI Python Linter

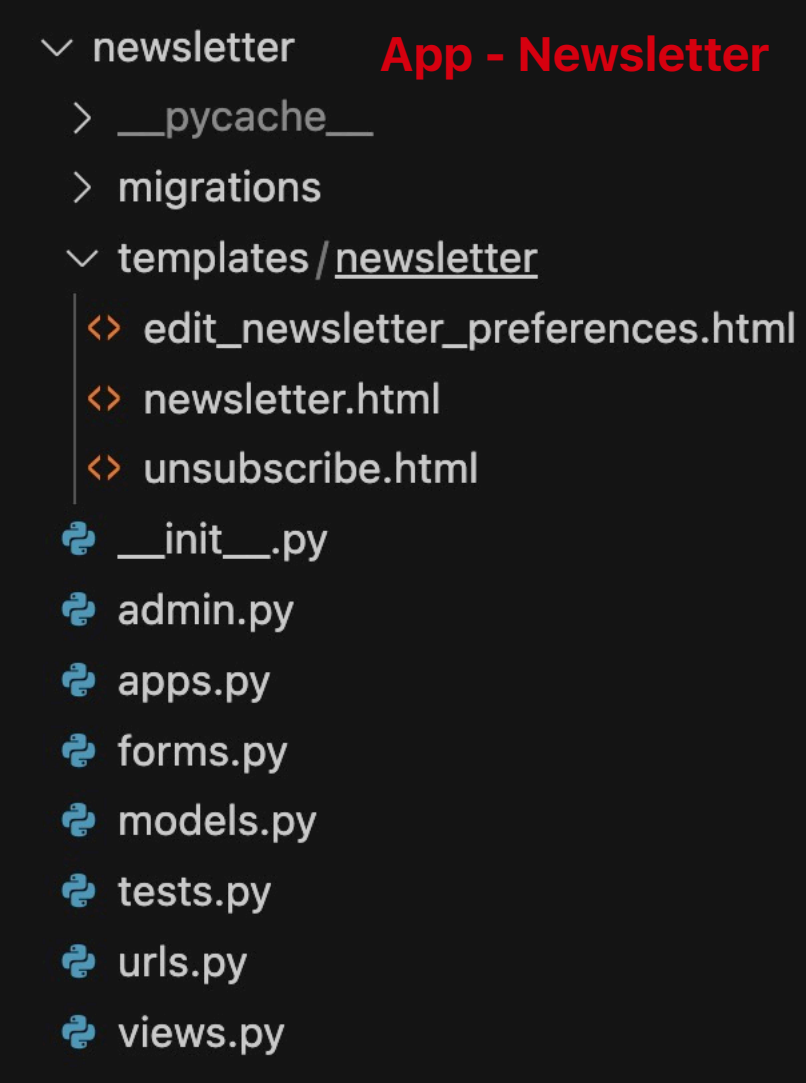
views.py

```python
from django.shortcuts import render
from the_cult_film_club.apps.releases.models import Releases


def index(request):
    """
    Display the home page for The Cult Film Club.

    This view fetches and displays 3 random cult film releases to
    highlight on the homepage.

    Args:
        request (HttpRequest): The incoming HTTP request.

    Returns:
        HttpResponse: The rendered home page with context data.
    """
    featured_releases = Releases.objects.order_by('?')[:3]
    return render(request, 'home/index.html', {
        'releases': featured_releases
    })

```

Settings:

Results:

All clear, no errors found


newsletter
App - Newsletter
__pycache__
migrations
templates/newsletter
edit_newsletter_preferences.html
newsletter.html
unsubscribe.html
__init__.py
admin.py
apps.py
forms.py
models.py
tests.py
urls.py
views.py

CI Python Linter

admin.py

```python
from django.contrib import admin
from the_cult_film_club.apps.newsletter.models import NewsletterSignup


@admin.register(NewsletterSignup)
class NewsletterSignupAdmin(admin.ModelAdmin):
    list_display = (
        'email',
        'display_genres',
        'date_joined',
        'unsubscribe_token',
    )
    list_filter = ('date_joined',)
    search_fields = ('email',)
    ordering = ['-date_joined']

    def display_genres(self, obj):
        """Display genres as a comma-separated list with better formatting."""
        return ", ".join(obj.genres.split(',')) if obj.genres else "-"
    display_genres.short_description = 'Genres'

```

Settings:

Results:

All clear, no errors found

CI Python Linter

apps.py

```python
from django.apps import AppConfig


class NewsletterConfig(AppConfig):
    """
    Configuration for the Newsletter app.
    """
    default_auto_field = "django.db.models.BigAutoField"
    name = "the_cult_film_club.apps.newsletter"
    verbose_name = "Newsletter"

```

Settings:

Results:

All clear, no errors found

CI Python Linter

forms.py

```python
from django import forms
from .models import NewsletterSignup
from the_cult_film_club.apps.releases.models import Releases


def get_genre_choices():
    """
    Return a list of distinct non-empty genre values from Releases
    for use in the newsletter signup form.
    """
    genres = (
        Releases.objects
        .exclude(genre__isnull=True)
        .exclude(genre__exact="")
        .values_list('genre', flat=True)
        .distinct()
    )
    return [(genre, genre) for genre in genres]


class NewsletterSignupForm(forms.ModelForm):
    genres = forms.MultipleChoiceField(
        choices=[],
        widget=forms.CheckboxSelectMultiple,
        required=True,
        label="I am interested in the following topics:"
    )

    class Meta:
        model = NewsletterSignup
        fields = ['email', 'genres']
```

Settings:

Results:

All clear, no errors found

CI Python Linter

models.py

```python
from django.db import models
import uuid


class NewsletterSignup(models.Model):
    """
    Stores newsletter subscriber details such as
    email, optional genre preferences, and an unsubscribe token.
    """
    email = models.EmailField(
        unique=True,
        help_text="Subscriber's email address"
    )
    genres = models.CharField(
        max_length=500,
        blank=True,
        help_text="Comma-separated list of preferred genres"
    )
    date_joined = models.DateTimeField(
        auto_now_add=True,
        help_text="Timestamp when the subscription was created"
    )
    unsubscribe_token = models.UUIDField(
        default=uuid.uuid4,
        editable=False,
        unique=True,
        help_text="Token used to securely unsubscribe via email link"
    )

    class Meta:
        verbose_name = "Subscriber"
```

Settings:

Results:

All clear, no errors found

CI Python Linter

urls.py

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

urlpatterns = [
    # Newsletter Signup Page
    path(
        '',
        views.newsletter_signup,
        name='newsletter_signup'
    ),

    # Unsubscribe via secure token
    path(
        'newsletter/unsubscribe/<uuid:token>/',
        views.unsubscribe,
        name='newsletter_unsubscribe'
    ),

    # Request unsubscribe link via email
    path(
        'newsletter/unsubscribe-request/',
        views.newsletter_unsubscribe_request,
        name='newsletter_unsubscribe_request'
    ),
    path(
        'newsletter/edit/<uuid:token>/',
        views.edit_newsletter_preferences,
        name='edit_newsletter_preferences'
    ),
]

```

Settings:

Results:

All clear, no errors found

CI Python Linter

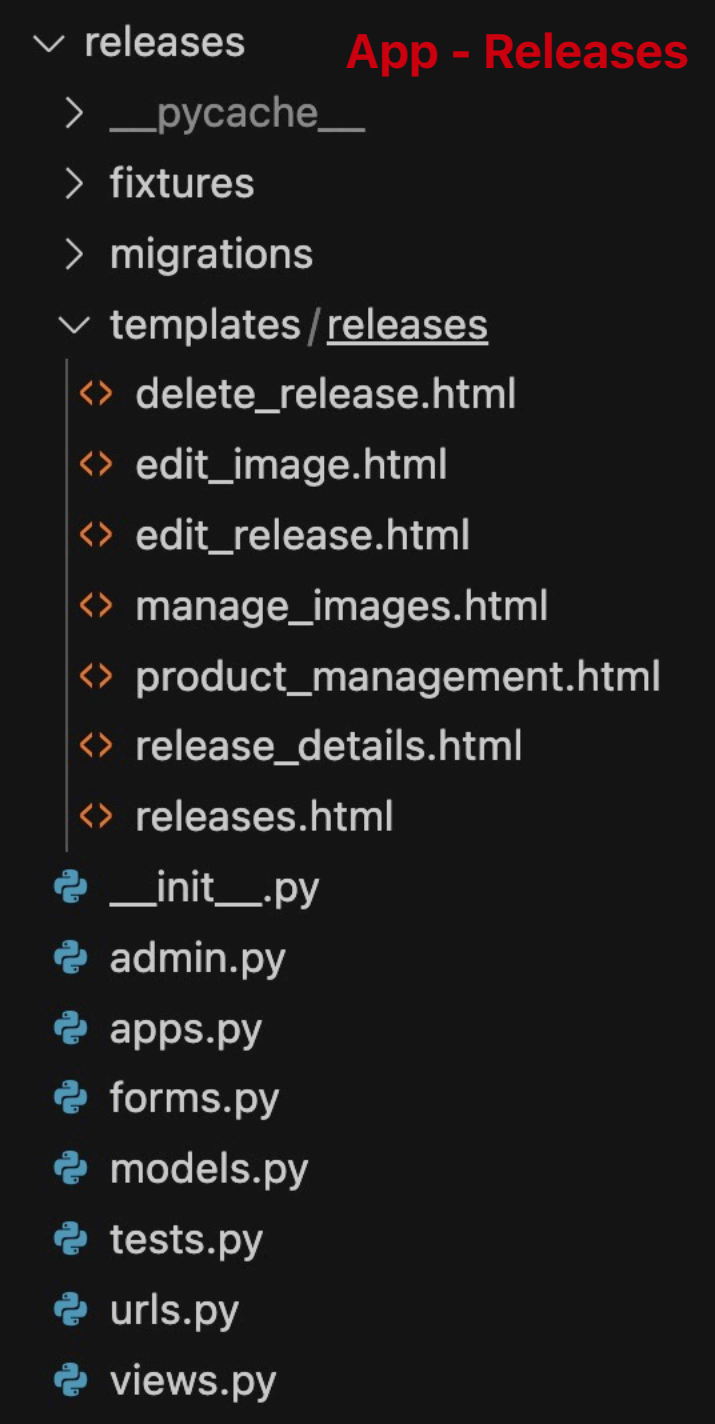
views.py

```python
from django.shortcuts import render, redirect, reverse, get_object_or_404
from django.contrib import messages
from django.core.mail import send_mail

from .forms import NewsletterSignupForm
from .models import NewsletterSignup


def newsletter_signup(request):
    """
    Handle newsletter subscription form rendering and submission.
    If the user is authenticated, their current subscription is checked.
    """
    subscriber = None
    if request.user.is_authenticated:
        subscriber = (
            NewsletterSignup.objects
            .filter(email=request.user.email)
            .first()
        )

    if request.method == 'POST':
        form = NewsletterSignupForm(request.POST, instance=subscriber)
        if form.is_valid():
            form.save()
            messages.success(
                request,
                (
                    "Thanks! You've been subscribed to The Cult Film Club "
                    "newsletter."
                )
```

Settings:

Results:

All clear, no errors found


releases
App - Releases
__pycache__
fixtures
migrations
templates/releases
delete_release.html
edit_image.html
edit_release.html
manage_images.html
product_management.html
release_details.html
releases.html
__init__.py
admin.py
apps.py
forms.py
models.py
tests.py
urls.py
views.py

CI Python Linter

admin.py

```python
from django.contrib import admin
from django.utils.html import format_html
from django import forms
from django_ckeditor_5.widgets import CKEditor5Widget
from .models import Releases, Images, Rating


class ReleasesAdminForm(forms.ModelForm):
    """Custom form for Releases admin with CKEditor widgets for text fields."""
    class Meta:
        model = Releases
        fields = "__all__"
        widgets = {
            "description": CKEditor5Widget(config_name="default"),
            "special_features": CKEditor5Widget(config_name="default"),
        }


class RatingAdminForm(forms.ModelForm):
    """Custom form for Rating admin with CKEditor widget for review."""
    class Meta:
        model = Rating
        fields = "__all__"
        widgets = {
            "review": CKEditor5Widget(config_name="default"),
        }


class ImageInline(admin.TabularInline):
    """Inline admin to manage Images related to a Release."""
    model = Images
```

Settings:

Results:

All clear, no errors found

CI Python Linter

apps.py

```python
from django.apps import AppConfig


class ReleasesConfig(AppConfig):
    """
    Configuration for the Releases app, handling film releases and related
    models.
    """
    default_auto_field = "django.db.models.BigAutoField"
    name = "the_cult_film_club.apps.releases"
    verbose_name = "Films"

```

Settings:

Results:

All clear, no errors found

CI Python Linter

forms.py

```python
from django import forms
from .models import Releases, Images, Rating
from django_ckeditor_5.widgets import CKEditor5Widget


class ReleaseForm(forms.ModelForm):
    """
    Form for creating or adding new Releases with selected fields.
    Uses CKEditor5Widget for rich text fields.
    """
    class Meta:
        model = Releases
        fields = [
            'title', 'release_date', 'director', 'description', 'genre',
            'subgenre', 'resolution', 'special_features', 'edition',
            'censor_status', 'packaging', 'copies_available', 'price'
        ]
        widgets = {
            'description': CKEditor5Widget(config_name="extends"),
            'special_features': CKEditor5Widget(config_name="extends"),
            'release_date': forms.DateInput(
                attrs={'type': 'date'},
                format='%Y-%m-%d'
            ),
            'price': forms.NumberInput(attrs={'step': '0.01', 'min': 0}),
            'copies_available': forms.NumberInput(attrs={'min': 0}),
        }
        labels = {
            'title': 'Title',
            'release_date': 'Release Date',
            'director': 'Director',
```

Settings:

Results:

All clear, no errors found

CI Python Linter

models.py

```python
from django.db import models
from django.contrib.auth.models import User
from django.db.models import Avg
from cloudinary.models import CloudinaryField
from django.core.validators import MaxValueValidator, MinValueValidator
from django_ckeditor_5.fields import CKEditor5Field


class Releases(models.Model):
    class Meta:
        verbose_name = "Release"
        verbose_name_plural = "Releases"
    title = models.CharField(max_length=100, blank=False, null=False)
    release_date = models.DateField(blank=False, null=False)
    director = models.CharField(max_length=100, blank=True, null=True)
    description = CKEditor5Field(max_length=1000, blank=True, null=True)
    genre = models.CharField(max_length=50, blank=True, null=True)
    subgenre = models.CharField(max_length=50, blank=True, null=True)
    resolution = models.CharField(max_length=10, blank=True, null=True)
    special_features = CKEditor5Field(max_length=2000, blank=True, null=True)
    edition = models.CharField(max_length=50, blank=True, null=True)
    censor_status = models.CharField(max_length=500, blank=True, null=True)
    packaging = models.CharField(max_length=500, blank=True, null=True)
    copies_available = models.IntegerField(blank=False, null=False, default=0)
    price = models.DecimalField(
        max_digits=10,
        decimal_places=2,
        blank=False,
        null=False
    )

```

Settings:

Results:

All clear, no errors found

CI Python Linter

urls.py

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


urlpatterns = [
    path('', views.releases, name='releases'),
    path(
        'release_details/<int:release_id>/',
        views.release_details,
        name='release_details'
    ),
    path("ckeditor5/", include("django_ckeditor_5.urls")),
    path('manage/', views.product_management, name='product_management'),
    path('edit/<int:release_id>/', views.edit_release, name='edit_release'),
    path(
        'delete/<int:release_id>/',
        views.delete_release,
        name='delete_release'
    ),
    path(
        'images/<int:release_id>/',
        views.manage_images,
        name='manage_images'
    ),
    path('images/<int:image_id>/edit/', views.edit_image, name='edit_image'),
    path(
        'images/delete/<int:image_id>/',
        views.delete_image,
        name='delete_image'
    ),
    path(
```

Settings:

Results:

All clear, no errors found

CI Python Linter

views.py

```python
import math
from django.shortcuts import render, get_object_or_404, reverse, redirect
from django.contrib import messages
from django.db.models import Q, Avg, F, Value
from django.core.paginator import Paginator
from django.db.models.functions import (
    Length, Substr, Reverse, StrIndex, ExtractYear
)
from django.contrib.auth.decorators import login_required
from django.utils.safestring import mark_safe

from the_cult_film_club.apps.releases.models import Releases, Rating, Images
from the_cult_film_club.apps.account.models import Wishlist, WishlistItem
from the_cult_film_club.apps.account.forms import WishlistItemForm
from .forms import ReleaseForm, ReleaseEditForm, ImageForm, RatingForm
from functools import wraps
from django.core.exceptions import PermissionDenied
from django.contrib.auth.views import redirect_to_login


def superuser_required(view_func):
    """
    Decorator for views that ensures the user is both authenticated and a
    superuser.

    - If the user is not authenticated, redirects them to the login page.
    - If the user is authenticated but not a superuser, raises a
      PermissionDenied (403) error.
    - If the user is a superuser, proceeds to the view as normal.

    Example usage:
```

Settings:

Results:

All clear, no errors found