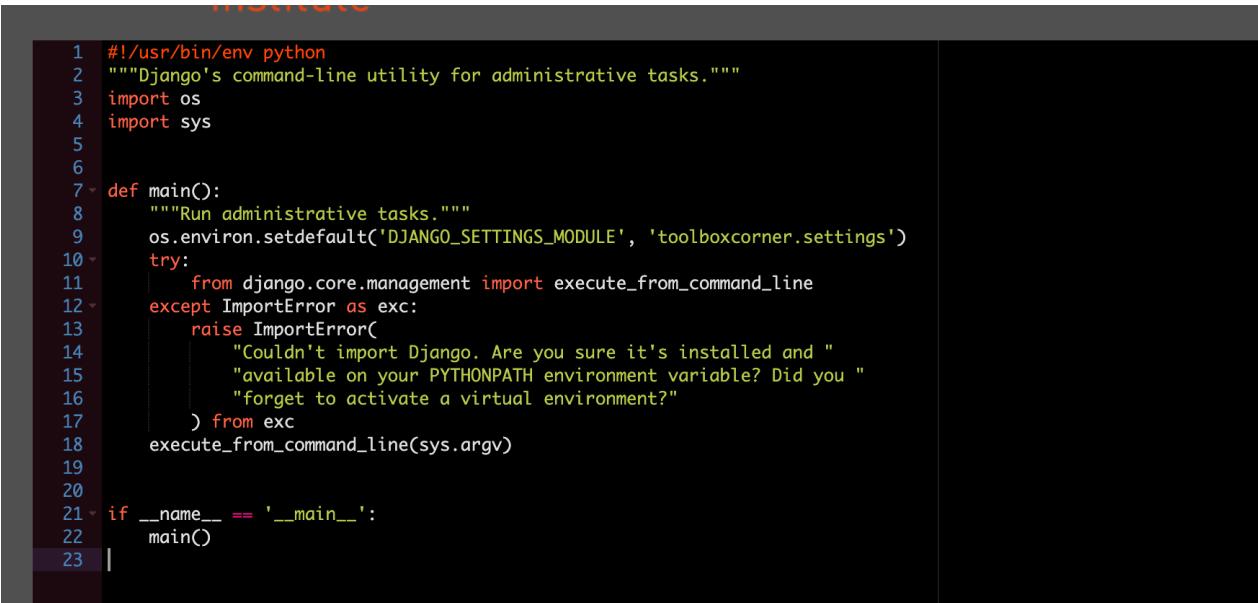


# Manage.py



The screenshot shows a code editor window with a dark background. On the left, the file `manage.py` is displayed, containing Python code for a Django command-line utility. The code includes imports for `os` and `sys`, a `main()` function, and logic for handling command-line arguments. On the right, there are two sections: "Settings" which includes a toggle switch for light/dark mode, and "Results" which displays the message "All clear, no errors found".

```
1 #!/usr/bin/env python
2 """Django's command-line utility for administrative tasks."""
3 import os
4 import sys
5
6 def main():
7     """Run administrative tasks."""
8     os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'toolboxcorner.settings')
9     try:
10         from django.core.management import execute_from_command_line
11     except ImportError as exc:
12         raise ImportError(
13             "Couldn't import Django. Are you sure it's installed and "
14             "available on your PYTHONPATH environment variable? Did you "
15             "forget to activate a virtual environment?"
16         ) from exc
17     execute_from_command_line(sys.argv)
18
19
20
21 if __name__ == '__main__':
22     main()
23 |
```

Settings:

Results:  
All clear, no errors found

# Website – Views.py

```
201     for form in image_formset:
202         if form.cleaned_data.get('DELETE'):
203             if form.instance.pk:
204                 form.instance.delete()
205         elif form.cleaned_data.get('image'):
206             image_instance = form.save(commit=False)
207             image_instance.product = product
208             image_instance.save()
209
210             messages.success(request, 'Product updated successfully!')
211             return redirect(reverse('product_detail', args=[product.id]))
212     else:
213         messages.error(request, 'Please correct the errors below.')
214 else:
215     form = ProductForm(instance=product)
216     image_formset = ProductImageFormSet(queryset=product.images.all())
217
218     return render(
219         request, 'website/edit_product.html',
220         {'form': form, 'image_formset': image_formset, 'product': product})
221
222
223
224 @staff_member_required
225 def delete_product(request, product_id):
226     """
227     Allows staff members to delete products.
228     Handles both GET (display confirmation)
229     and POST (process deletion) requests.
230     """
231     product = get_object_or_404(Product, id=product_id)
232
233     if request.method == 'POST':
234         product.delete()
235         messages.success(request, 'Product deleted successfully!')
```

Settings:



Results:

All clear, no errors found

# Website – urls.py



```
1 from django.urls import path
2 from . import views
3 from django.urls import path, include
4 from .views import edit_product, search_results
5
6 urlpatterns = [
7     path('', views.welcome, name='welcome'),
8     path('products', views.home, name='home'),
9     path('products/category/<int:category_id>/',
10         views.category_products, name='category_products'),
11     path('product/<int:product_id>', views.product_detail,
12         name='product_detail'),
13     path('product/add/', views.add_product, name='add_product'),
14     path('edit/<int:product_id>', edit_product, name='edit_product'),
15     path('delete/<int:product_id>',
16         views.delete_product, name='delete_product'),
17     path('bag/', include('bag.urls')),
18     path('search/', search_results, name='search_results'),
19 ]
20
```

Settings:

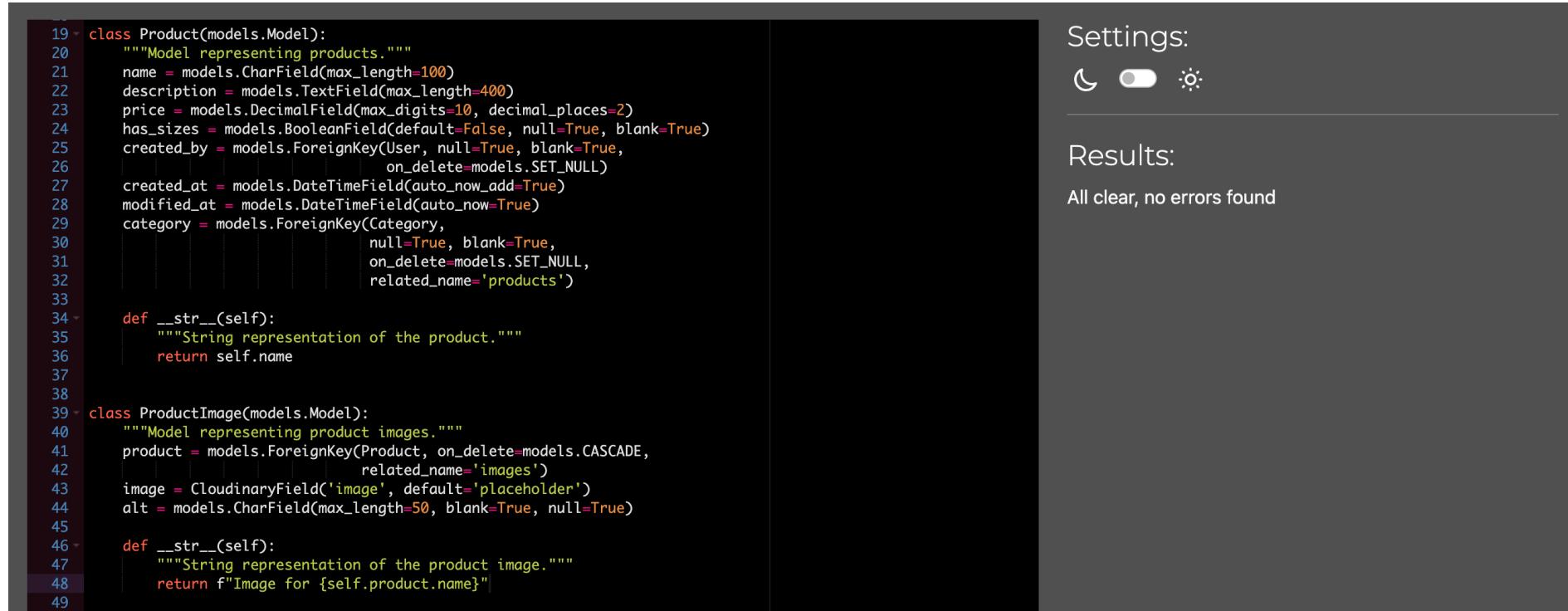
🌙 ⏵ ☀️

---

Results:

All clear, no errors found

# Website – Models.py



The screenshot shows a code editor interface with a dark theme. On the left, there is a vertical toolbar with icons for file operations like new, open, save, and close. The main area displays two Python class definitions:

```
19 - class Product(models.Model):
20     """Model representing products."""
21     name = models.CharField(max_length=100)
22     description = models.TextField(max_length=400)
23     price = models.DecimalField(max_digits=10, decimal_places=2)
24     has_sizes = models.BooleanField(default=False, null=True, blank=True)
25     created_by = models.ForeignKey(User, null=True, blank=True,
26                                     on_delete=models.SET_NULL)
27     created_at = models.DateTimeField(auto_now_add=True)
28     modified_at = models.DateTimeField(auto_now=True)
29     category = models.ForeignKey(Category,
30                                  null=True, blank=True,
31                                  on_delete=models.SET_NULL,
32                                  related_name='products')
33
34 -     def __str__(self):
35         """String representation of the product."""
36         return self.name
37
38
39 - class ProductImage(models.Model):
40     """Model representing product images."""
41     product = models.ForeignKey(Product, on_delete=models.CASCADE,
42                                related_name='images')
43     image = CloudinaryField('image', default='placeholder')
44     alt = models.CharField(max_length=50, blank=True, null=True)
45
46 -     def __str__(self):
47         """String representation of the product image."""
48         return f'Image for {self.product.name}'
49
```

On the right side of the interface, there are sections for "Settings" and "Results". The "Settings" section includes a toggle switch for light/dark mode and a brightness slider. The "Results" section displays the message "All clear, no errors found".

# Website – forms.py

```
 1 from django import forms
 2 from .models import Product, ProductImage, Category
 3 from django.forms import modelformset_factory
 4
 5
 6 class ProductForm(forms.ModelForm):
 7     """
 8     Form for creating and updating Product instances.
 9     """
10    class Meta:
11        model = Product
12        fields = ['name', 'description', 'price', 'has_sizes', 'category']
13
14
15 class ProductImageForm(forms.ModelForm):
16     """
17     Form for creating and updating ProductImage instances.
18     """
19    class Meta:
20        model = ProductImage
21        fields = ['image', 'alt']
22
23 # Define the formset for ProductImage
24
25
26 ProductImageFormSet = modelformset_factory(ProductImage,
27                                             form=ProductImageForm,
28                                             extra=1, can_delete=True)
29
```

Settings:



Results:

All clear, no errors found

# Website – contexts.py

```
 1 from .models import Category
 2
 3
 4 def categories(request):
 5     """
 6         Context processor to add categories to all templates.
 7         This function retrieves all Category objects from the
 8         database and returns them in a dictionary. The dictionary
 9         is added to the context of all templates rendered during
10         the request-response cycle, making the categories available
11         globally in all templates.
12     """
13     return {
14         'categories': Category.objects.all()
15     }
16 |
```

Settings:



Results:

All clear, no errors found

# Website – apps.py

```
 1 from django.apps import AppConfig  
 2  
 3  
 4 class WebsiteConfig(AppConfig):  
 5     default_auto_field = 'django.db.models.BigAutoField'  
 6     name = 'website'  
 7 |
```

Settings:



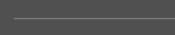
Results:

All clear, no errors found

# Website – admin.py

```
 1 from django.contrib import admin
 2 from .models import Product, ProductImage, Category
 3
 4
 5 class ProductImageInline(admin.TabularInline):
 6     """
 7         Defines the inline representation of ProductImage
 8         model in the admin interface.
 9         Allows adding/editing images directly in the Product admin page.
10     """
11     model = ProductImage
12
13
14 @admin.register(Product)
15 class ProductAdmin(admin.ModelAdmin):
16     """
17         Defines the admin interface for the Product model.
18     """
19     list_display = ('name', 'price', 'created_by', 'created_at')
20     list_filter = ('created_at',)
21     search_fields = ('name',)
22     inlines = [ProductImageInline]
23
24 |
25 @admin.register(Category)
26 class CategoryAdmin(admin.ModelAdmin):
27     """
28         Defines the admin interface for the Category model.
29     """
30     list_display = ('name',)
```

Settings:



Results:

All clear, no errors found

# Website – templatetags – custom\_filters.py

```
1 from django import template
2 import re
3
4 register = template.Library()
5
6 |
7 @register.filter
8 def first_sentence(value):
9     """
10     Returns the first sentence of the string,
11     including the ending punctuation.
12     """
13     if not isinstance(value, str):
14         return value
15     """
16     Use a regular expression to find the first
17     occurrence of a period or exclamation mark
18     """
19     match = re.search(r'[.!]$', value)
20     if match:
21         end_pos = match.end()
22         return value[:end_pos]
23     return value
24
```

Settings:



Results:

All clear, no errors found

# Tooboxcorner – wsgi.py

```
1 """
2 WSGI config for toolboxcorner project.
3
4 It exposes the WSGI callable as a module-level variable named ``application``.
5
6 For more information on this file, see
7 https://docs.djangoproject.com/en/3.2/howto/deployment/wsgi/
8 """
9
10 import os
11
12 from django.core.wsgi import get_wsgi_application
13
14 os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'toolboxcorner.settings')
15
16 application = get_wsgi_application()
17 |
```

Settings:



Results:

All clear, no errors found

# Toolboxcorner – views.py

The screenshot shows a code editor interface with a dark theme. On the left, a vertical toolbar has icons for file operations (New, Open, Save, etc.) and a search bar. The main area displays the following Python code:

```
1 from django.shortcuts import render
2
3 def handler404(request, exception):
4     """ Error Handler 404 - Page Not Found """
5     return render(request, "errors/404.html", status=404)
6
```

To the right of the code editor is a sidebar with two sections: "Settings:" and "Results:". The "Settings:" section contains three icons: a moon (dark mode), a toggle switch, and a gear. The "Results:" section displays the message "All clear, no errors found".

# Toolboxcorner – urls.py

Code content:

```
5 Examples:
6 Function views
7   1. Add an import: from my_app import views
8     2. Add a URL to urlpatterns: path('', views.home, name='home')
9 Class-based views
10  1. Add an import: from other_app.views import Home
11    2. Add a URL to urlpatterns: path('', Home.as_view(), name='home')
12 Including another URLconf
13  1. Import the include() function: from django.urls import include, path
14    2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))
15 """
16
17 from django.contrib import admin
18 from django.urls import path, include
19 from django.conf import settings
20 from django.conf.urls.static import static
21 from .views import handler404
22
23 urlpatterns = [
24     path('admin/', admin.site.urls),
25     path('accounts/', include('allauth.urls')),
26     path('', include('website.urls')),
27     path('products/', include('website.urls')),
28     path('contact/', include('contact.urls')),
29     path('reviews/', include('review.urls')),
30     path('bag/', include('bag.urls')),
31     path('checkout/', include('checkout.urls')),
32 ] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
33
34 handler404 = 'toolboxcorner.views.handler404'
35 |
```

Settings:

🌙 ⏵ ☀️

---

Results:

All clear, no errors found

# Toolboxcorner – settings.py

```
1 """
2 Django settings for toolboxcorner project.
3
4 Generated by 'django-admin startproject' using Django 3.2.9.
5
6 For more information on this file, see
7 https://docs.djangoproject.com/en/3.2/topics/settings/
8
9 For the full list of settings and their values, see
10 https://docs.djangoproject.com/en/3.2/ref/settings/
11 """
12
13 from pathlib import Path
14 import os
15 import dj_database_url
16 if os.path.isfile('env.py'):
17     import env
18
19 from django.contrib.messages import constants as messages
20
21 # Build paths inside the project like this: os.path.join(BASE_DIR, ...)
22
23 BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
24 |
25
26 # Quick-start development settings - unsuitable for production
27 # See https://docs.djangoproject.com/en/3.2/howto/deployment/checklist/
28
29 # SECURITY WARNING: keep the secret key used in production secret!
30 SECRET_KEY = os.environ.get('SECRET_KEY', '')
31
```

Settings:



Results:

[36: E501 line too long \(85 > 79 characters\)](#)  
[148: E501 line too long \(83 > 79 characters\)](#)

# Toolboxcorner – asgi.py

```
1 """
2 ASGI config for toolboxcorner project.
3
4 It exposes the ASGI callable as a module-level variable named ``application``.
5
6 For more information on this file, see
7 https://docs.djangoproject.com/en/3.2/howto/deployment/asgi/
8 """
9
10 import os
11
12 from django.core.asgi import get_asgi_application
13
14 os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'toolboxcorner.settings')
15
16 application = get_asgi_application()
17
```

Settings:



Results:

All clear, no errors found

# Review - views.py

```
 1 from django.shortcuts import render, get_object_or_404, redirect
 2 from django.contrib.auth.decorators import login_required
 3 from django.http import HttpResponseRedirect
 4 from .models import Review
 5 from .forms import ReviewForm
 6 from website.models import Product
 7 from django.db.models import Avg
 8
 9
10 @login_required
11 def add_review(request, product_id):
12     """
13     Handles the submission of a new review for a given product.
14     """
15     product = get_object_or_404(Product, id=product_id)
16     if request.method == 'POST':
17         form = ReviewForm(request.POST)
18         if form.is_valid():
19             review = form.save(commit=False)
20             review.product = product
21             review.user = request.user
22             review.save()
23             return redirect('product_detail', product.id)
24     else:
25         form = ReviewForm()
26     return render(request,
27                  'review/add_review.html',
28                  {'form': form, 'product': product})
29
30
31 @login_required
```

Settings:



Results:

All clear, no errors found

# Review – urls.py

```
 1 from django.urls import path
 2 from . import views
 3
 4
 5 urlpatterns = [
 6     path('add/<int:product_id>', views.add_review, name='add_review'),
 7     path('edit/<int:review_id>', views.edit_review, name='edit_review'),
 8     path('delete/<int:review_id>', views.delete_review, name='delete_review'),
 9     path('product/<int:product_id>', views.review_list, name='review_list'),
10 ]
11
```

Settings:



Results:

All clear, no errors found

# Review – models.py

```
 1 from django.db import models
 2 from django.conf import settings
 3 from website.models import Product
 4
 5
 6 class Review(models.Model):
 7     """
 8         Model to represent a product review.
 9     """
10     STAR_CHOICES = [(i, str(i)) for i in range(6)]
11
12     product = models.ForeignKey(Product, on_delete=models.CASCADE)
13     user = models.ForeignKey(settings.AUTH_USER_MODEL,
14                             on_delete=models.CASCADE)
15     content = models.TextField(max_length=500, blank=True)
16     stars = models.IntegerField(choices=STAR_CHOICES)
17     created_at = models.DateTimeField(auto_now_add=True)
18     updated_at = models.DateTimeField(auto_now=True)
19
20     class Meta:
21         """
22             Meta class to enforce unique reviews per product-user pair.
23         """
24         unique_together = ('product', 'user')
25
26     def __str__(self):
27         """
28             String representation of the review.
29         """
30         return f'Review for {self.product.name} by {self.user.username}'
31
```

Settings:



Results:

All clear, no errors found

# Review – forms.py

```
 1 from django import forms
 2 from .models import Review
 3
 4 |
 5 class ReviewForm(forms.ModelForm):
 6     """
 7     Form to create or edit a review.
 8     """
 9     class Meta:
10         """
11         Meta class to specify the model and fields used in the form.
12         """
13         model = Review
14         fields = ['content', 'stars']
```

Settings:



Results:

All clear, no errors found

# Review – apps.py

The screenshot shows a code editor on the left and a settings panel on the right.

**Code:**

```
1 from django.apps import AppConfig  
2  
3  
4 class ReviewConfig(AppConfig):  
5     default_auto_field = 'django.db.models.BigAutoField'  
6     name = 'review'  
7
```

**Settings:**

⌚ ⚡ ⚙

---

**Results:**

All clear, no errors found

# Review - admin.py

```
 1 from django.contrib import admin
 2 from .models import Review
 3
 4
 5 @admin.register(Review)
 6 class ReviewAdmin(admin.ModelAdmin):
 7     """Admin configuration for the Review model."""
 8
 9     list_display = ('product', 'user', 'stars', 'created_at')
10     list_filter = ('stars', 'created_at')
11     search_fields = ('content',)
```

Settings:



Results:

All clear, no errors found

# Contact – views.py

```
 1 from django.shortcuts import render, redirect
 2 from django.contrib import messages
 3 from .forms import ContactForm
 4
 5
 6 def contact_view(request):
 7     """Handle the contact form submission and render the contact view."""
 8     if request.method == 'POST':
 9         form = ContactForm(request.POST)
10         if form.is_valid():
11             form.save()
12             messages.success(request,
13                             'Your message has been sent successfully!')
14             return redirect('contact')
15     else:
16         form = ContactForm()
17
18     return render(request, 'contact/contact.html', {'form': form})
19 |
```

Settings:



Results:

All clear, no errors found

# Contact – urls.py

```
1 from django.urls import path
2 from .views import contact_view
3
4
5 urlpatterns = [
6     path('', contact_view, name='contact'),
7 ]
8
```

Settings:



Results:

All clear, no errors found

# Contact – models.py

```
 1 from django.db import models
 2 from django.core.validators import MinLengthValidator, MaxLengthValidator
 3 from django.utils import timezone
 4
 5
 6 class Contact(models.Model):
 7     """Model to store contact form submissions."""
 8     SUBJECT_CHOICES = [
 9         ('problems', 'Problems'),
10         ('product_info', 'Product Information'),
11         ('general_info', 'General Information'),
12         ('work_with_us', 'Work with Us'),
13         ('other', 'Other'),
14     ]
15
16     subject = models.CharField(max_length=50, choices=SUPERJECT_CHOICES)
17     content = models.TextField(validators=[MinLengthValidator(10),
18                                           MaxLengthValidator(500)])
19     email = models.EmailField()
20     created_at = models.DateTimeField(default=timezone.now)
21
22     def __str__(self):
23         return f'{self.get_subject_display()} - {self.email}'
24
```

Settings:



Results:

All clear, no errors found

# Contact – forms.py

```
1 from django import forms
2 from .models import Contact
3
4 class ContactForm(forms.ModelForm):
5     """Form for contacting purposes."""
6
7     class Meta:
8         model = Contact
9         fields = ['subject', 'content', 'email']
10
```

Settings:



Results:

All clear, no errors found

# Contact – apps.py

```
1 from django.apps import AppConfig  
2  
3  
4 class ContactConfig(AppConfig):  
5     default_auto_field = 'django.db.models.BigAutoField'  
6     name = 'contact'  
7 |
```

Settings:



Results:

All clear, no errors found

# Contact – admin.py

```
 1 from django.contrib import admin
 2 from .models import Contact
 3
 4
 5 @admin.register(Contact)
 6 class ContactAdmin(admin.ModelAdmin):
 7     """Admin configuration for the Contact model."""
 8
 9     list_display = ('subject', 'email', 'content', 'created_at')
10     list_filter = ('subject', 'created_at')
11     search_fields = ('email', 'content')
12
```

Settings:



Results:

All clear, no errors found

# Checkout – webhooks.py

```
 1 from django.conf import settings
 2 from django.http import HttpResponseRedirect
 3 from django.views.decorators.http import require_POST
 4 from django.views.decorators.csrf import csrf_exempt
 5
 6 from checkout.webhook_handler import StripeWH_Handler
 7
 8 import stripe
 9
10
11 @require_POST
12 @csrf_exempt
13 def webhook(request):
14     """Listen for webhooks from Stripe"""
15     # Setup
16     wh_secret = settings.STRIPE_WH_SECRET
17     stripe.api_key = settings.STRIPE_SECRET_KEY
18
19     # Get the webhook data and verify its signature
20     payload = request.body
21     sig_header = request.META['HTTP_STRIPE_SIGNATURE']
22     event = None
23
24     try:
25         event = stripe.Webhook.construct_event(
26             payload, sig_header, wh_secret
27         )
28     except ValueError as e:
29         # Invalid payload
30         return HttpResponseRedirect(status=400)
31     except stripe.error.SignatureVerificationError as e:
```

Settings:



Results:

All clear, no errors found

# Checkout – webhook\_handler.py

```
 1 from django.http import HttpResponse
 2
 3 from .models import Order, OrderLineItem
 4 from website.models import Product
 5
 6 import json
 7 import time
 8 import stripe
 9
10
11 class StripeWH_Handler:
12     """Handle Stripe webhooks"""
13
14     def __init__(self, request):
15         self.request = request
16
17     def handle_event(self, event):
18         """
19             Handle a generic/unknown/unexpected webhook event
20         """
21         return JsonResponse(
22             content=f'Unhandled webhook received: {event["type"]}',
23             status=200
24         )
25
26     def handle_payment_intent_succeeded(self, event):
27         """
28             Handle the payment_intent.succeeded webhook from Stripe
29         """
30         intent = event.data.object
31         pid = intent.id
```

Settings:



Results:

[106: E501 line too long \(81 > 79 characters\)](#)

# Checkout – views.py

```
 1 from django.shortcuts import render, redirect, reverse
 2 from django.shortcuts import get_object_or_404, HttpResponseRedirect
 3 from django.views.decorators.http import require_POST
 4 from django.contrib import messages
 5 from django.conf import settings
 6 import logging
 7
 8 from .forms import OrderForm
 9 from .models import Order, OrderLineItem
10 from website.models import Product
11 from bag.contexts import bag_contents
12
13 import stripe
14 import json
15
16
17 @require_POST
18 def cache_checkout_data(request):
19     """
20     Cache checkout data to associate with Stripe PaymentIntent.
21     """
22     try:
23         pid = request.POST.get('client_secret').split('_secret')[0]
24         stripe.api_key = settings.STRIPE_SECRET_KEY
25         stripe.PaymentIntent.modify(pid, metadata={
26             'bag': json.dumps(request.session.get('bag', {})),
27             'save_info': request.POST.get('save_info'),
28             'username': request.user,
29         })
30         return HttpResponseRedirect(status=200)
31     except Exception as e:
```

Settings:



Results:

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

# Checkout - urls.py

```
 1 from django.urls import path
 2 from . import views
 3 from .webhooks import webhook
 4
 5 urlpatterns = [
 6     path('', views.checkout, name='checkout'),
 7     path('checkout_success/<order_number>', views.checkout_success,
 8          name='checkout_success'),
 9     path('cache_checkout_data/', views.cache_checkout_data,
10          name='cache_checkout_data'),
11     path('wh/', webhook, name='webhook'),
12 ]
13 ]
```

Settings:



Results:

All clear, no errors found

# Checkout – signals.py

```
 1 from django.db.models.signals import post_save, post_delete
 2 from django.dispatch import receiver
 3
 4 from .models import OrderLineItem
 5
 6
 7 @receiver(post_save, sender=OrderLineItem)
 8 def update_on_save(sender, instance, created, **kwargs):
 9     """
10     Update order total on lineitem update/create
11     """
12     instance.order.update_total()
13
14
15 @receiver(post_delete, sender=OrderLineItem)
16 def update_on_delete(sender, instance, **kwargs):
17     """
18     Update order total on lineitem delete
19     """
20     instance.order.update_total()
21 |
```

Settings:



Results:

All clear, no errors found

# Checkout – models.py

```
58     if not self.order_number:
59         self.order_number = self._generate_order_number()
60     super().save(*args, **kwargs)
61
62     def __str__(self):
63         return self.order_number
64
65
66     class OrderLineItem(models.Model):
67         order = models.ForeignKey(Order, null=False,
68             blank=False, on_delete=models.CASCADE,
69             related_name='lineitems')
70         product = models.ForeignKey(Product, null=False,
71             blank=False, on_delete=models.CASCADE)
72         product_size = models.CharField(max_length=2, null=True, blank=True)
73         quantity = models.IntegerField(null=False, blank=False, default=0)
74         lineitem_total = models.DecimalField(max_digits=6,
75             decimal_places=2, null=False,
76             blank=False, editable=False)
77
78     def save(self, *args, **kwargs):
79         """
80             Override the original save method to set the lineitem total
81             and update the order total.
82         """
83         self.lineitem_total = self.product.price * self.quantity
84         super().save(*args, **kwargs)
85
86     def __str__(self):
87         return f'NAME {self.product.name} on order {self.order.order_number}'
88 |
```

Settings:



Results:

[45: E501 line too long \(102 > 79 characters\)](#)

[47: E501 line too long \(95 > 79 characters\)](#)

# Checkout – forms.py

```
10     'town_or_city', 'postcode', 'country',
11     'county',)
12
13 def __init__(self, *args, **kwargs):
14     """
15     Add placeholders and classes, remove auto-generated
16     labels and set autofocus on first field
17     """
18     super().__init__(*args, **kwargs)
19     placeholders = {
20         'full_name': 'Full Name',
21         'email': 'Email Address',
22         'phone_number': 'Phone Number',
23         'postcode': 'Postal Code',
24         'town_or_city': 'Town or City',
25         'street_address1': 'Street Address 1',
26         'street_address2': 'Street Address 2',
27         'county': 'County, State or Locality',
28     }
29
30     self.fields['full_name'].widget.attrs['autofocus'] = True
31     for field in self.fields:
32         if field != 'country':
33             if self.fields[field].required:
34                 placeholder = f'{placeholders[field]} *'
35             else:
36                 placeholder = placeholders[field]
37             self.fields[field].widget.attrs['placeholder'] = placeholder
38             self.fields[field].widget.attrs['class'] = 'stripe-style-input'
39             self.fields[field].label = False
40
```

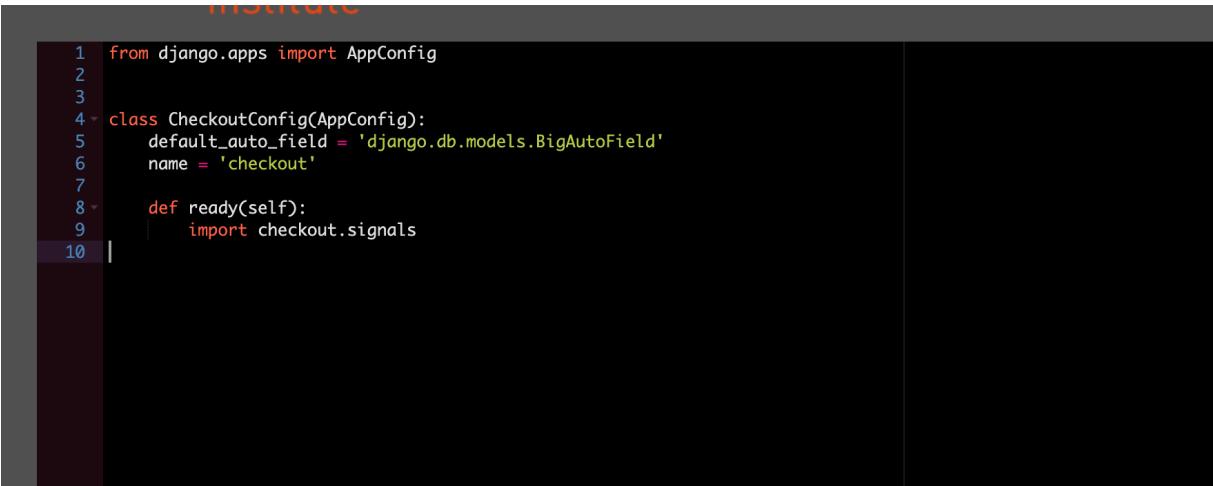
Settings:



Results:

All clear, no errors found

# Checkout – apps.py



The screenshot shows a code editor with a dark theme. On the left, there is a vertical file navigation bar with icons for Home, Projects, and Settings. The main area displays the contents of an `apps.py` file:

```
from django.apps import AppConfig
class CheckoutConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'checkout'
    def ready(self):
        import checkout.signals
```

The code is color-coded: `from`, `class`, `def`, and `import` are in red; `AppConfig`, `default_auto_field`, `name`, and `signals` are in green; and `django` and `models` are in blue.

On the right side of the interface, there are two sections: **Settings:** which includes a dark mode switch, a toggle switch, and a brightness icon; and **Results:** which displays the message "All clear, no errors found".

# Checkout – admin.py

The screenshot shows a code editor with a dark theme. On the left, the file `admin.py` contains the following Python code:

```
3
4
5 class OrderLineItemAdminInline(admin.TabularInline):
6     model = OrderLineItem
7     readonly_fields = ('lineitem_total',)
8
9
10 class OrderAdmin(admin.ModelAdmin):
11     inlines = (OrderLineItemAdminInline,)
12
13     readonly_fields = ('order_number', 'date',
14                         'delivery_cost', 'order_total',
15                         'grand_total', 'original_bag',
16                         'stripe_pid')
17
18     fields = ('order_number', 'date', 'full_name',
19               'email', 'phone_number', 'country',
20               'postcode', 'town_or_city', 'street_address1',
21               'street_address2', 'county', 'delivery_cost',
22               'order_total', 'grand_total', 'original_bag',
23               'stripe_pid')
24
25     list_display = ('order_number', 'date', 'full_name',
26                     'order_total', 'delivery_cost',
27                     'grand_total',)
28
29     ordering = ('-date',)
30
31
32 admin.site.register(Order, OrderAdmin)
33
```

The right side of the interface has a dark sidebar with the following sections:

- Settings:** Includes icons for moon (light/dark mode), a toggle switch, and a sun (bright mode).
- Results:** Displays the message "All clear, no errors found".

# Bag – views.py

```
 1 from django.shortcuts import render, redirect, reverse
 2 from django.shortcuts import HttpResponseRedirect, get_object_or_404
 3 from django.contrib import messages
 4 from website.models import Product
 5
 6
 7 def view_bag(request):
 8     """ A view that renders the bag contents page """
 9
10     return render(request, 'bag/bag.html')
11
12
13 def add_to_bag(request, item_id):
14     """ Add a quantity of the specified product to the shopping bag """
15     product = get_object_or_404(Product, pk=item_id)
16     quantity = int(request.POST.get('quantity'))
17     redirect_url = request.POST.get('redirect_url')
18     size = None
19     if 'product_size' in request.POST:
20         size = request.POST['product_size']
21     bag = request.session.get('bag', {})
22
23     if size:
24         if item_id in list(bag.keys()):
25             if size in bag[item_id]['items_by_size'].keys():
26                 bag[item_id]['items_by_size'][size] += quantity
27             else:
28                 bag[item_id]['items_by_size'][size] = quantity
29         else:
30             bag[item_id] = {'items_by_size': {size: quantity}}
31     else:
```

Settings:



Results:

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

# Bag – urls.py

```
 1 from django.urls import path
 2 from . import views
 3
 4 urlpatterns = [
 5     path('', views.view_bag, name='view_bag'),
 6     path('add/<item_id>/', views.add_to_bag, name='add_to_bag'),
 7     path('adjust/<item_id>/', views.adjust_bag, name='adjust_bag'),
 8     path('remove/<item_id>/', views.remove_from_bag, name='remove_from_bag'),
 9 ]
10 |
```

Settings:



Results:

All clear, no errors found

# Bag – contexts.py

```
26     for size, quantity in item_data['items_by_size'].items():
27         total += quantity * product.price
28         product_count += quantity
29         bag_items.append({
30             'item_id': item_id,
31             'quantity': quantity,
32             'product': product,
33             'size': size,
34         })
35
36     if total < settings.FREE_DELIVERY_THRESHOLD:
37         delivery = total * Decimal(settings.STANDARD_DELIVERY_PERCENTAGE / 100)
38         free_delivery_delta = settings.FREE_DELIVERY_THRESHOLD - total
39     else:
40         delivery = 0
41         free_delivery_delta = 0
42
43     grand_total = delivery + total
44
45     context = {
46         'bag_items': bag_items,
47         'total': total,
48         'product_count': product_count,
49         'delivery': delivery,
50         'free_delivery_delta': free_delivery_delta,
51         'free_delivery_threshold': settings.FREE_DELIVERY_THRESHOLD,
52         'grand_total': grand_total,
53     }
54
55     return context
56 
```

Settings:



Results:

All clear, no errors found

# Bag – apps.py

```
1 from django.apps import AppConfig  
2  
3  
4 class BagConfig(AppConfig):  
5     default_auto_field = 'django.db.models.BigAutoField'  
6     name = 'bag'  
7
```

Settings:



Results:

All clear, no errors found

# Bag – templatetags – bag\_tools.py

```
1 from django import template
2
3
4 register = template.Library()
5
6 @register.filter(name='calc_subtotal')
7 def calc_subtotal(price, quantity):
8     return price * quantity
9
```

Settings:



Results:

All clear, no errors found