


# dishes: admin.py




# dishes: apps.py

 CI Python Linter

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


Settings:

 ☒ 

Results:




All clear, no errors found

# dishes: models.py

 CI Python Linter

```
1 from django.core.validators import MinValueValidator
2 from decimal import Decimal
3 from django.db import models
4 from restaurants.models import Restaurant
5
6
7 class Dish(models.Model):
8     """
9     Represents a dish served at a restaurant, including its name, description,
10     price, image, dietary preferences, and creation/update timestamps.
11
12     Relationships:
13     - 'restaurant': The restaurant where the dish is served
14       (ForeignKey to Restaurant).
15     """
16     dietary_preferences_choices = [
17         ('vegetarian', 'Vegetarian'),
18         ('vegan', 'Vegan'),
19         ('pescatarian', 'Pescatarian'),
20         ('gluten-free', 'Gluten-free'),
21         ('halal', 'Halal'),
22         ('kosher', 'Kosher'),
23         ('other', 'Other'),
24     ]
25
26     restaurant = models.ForeignKey(Restaurant, on_delete=models.CASCADE)
27     name = models.CharField(max_length=255)
28     short_description = models.TextField(max_length=500)
29     price = models.DecimalField(
30         max_digits=6,
31         decimal_places=2,
32         validators=[MinValueValidator(Decimal('0'))]
33     )
34     image = models.ImageField(upload_to='images/', blank=True)
35     dietary_preference = models.CharField(
36         max_length=25, choices=dietary_preferences_choices
```

Settings:

Results:

All clear, no errors found

# dishes: serializers.py

 CI Python Linter

```
1 from rest_framework import serializers
2 from .models import Dish
3
4
5 class DishSerializer(serializers.ModelSerializer):
6     """
7     Serializer for the Dish model. Handles serialization and validation
8     of dish data.
9
10    Validation:
11    - 'image': Ensures the uploaded image is not larger than 2MB and its
12      dimensions do not exceed 4096px in width or height.
13    """
14    def validate_image(self, value):
15        if value.size > 1024 * 1024 * 2:
16            raise serializers.ValidationError(
17                'Image size larger than 2MB.'
18            )
19        if value.image.width > 4096:
20            raise serializers.ValidationError(
21                'Image width larger than 4096px.'
22            )
23        if value.image.height > 4096:
24            raise serializers.ValidationError(
25                'Image height larger than 4096px.'
26            )
27        return value
28
29    class Meta:
30        model = Dish
31        fields = '__all__'
32
33
34 class DishDetailSerializer(DishSerializer):
35     """
36     Serializer for detailed dish information. Inherits from DishSerializer.
```

Settings:

 ☒ 

Results:

All clear, no errors found

# dishes: tests.py



## CI Python Linter

```
120
121 - def test_retrieve_dish(self):
122     self.client.force_authenticate(user=self.user)
123     response = self.client.get(self.url)
124     self.assertEqual(response.status_code, status.HTTP_200_OK)
125     self.assertEqual(response.data['name'], 'Dish 1')
126
127 - def test_update_dish(self):
128     image = Image.new('RGB', (100, 100))
129     image_io = BytesIO()
130     image.save(image_io, format='JPEG')
131     image_io.seek(0)
132     uploaded_image = SimpleUploadedFile(
133         'test_image.jpg', image_io.read(), content_type='image/jpeg'
134     )
135     self.client.force_authenticate(user=self.user)
136 -     data = {
137         'restaurant': self.restaurant.id,
138         'name': 'Updated Dish',
139         'short_description': 'example',
140         'price': '13.50',
141         'image': uploaded_image,
142         'dietary_preference': 'vegetarian',
143     }
144     response = self.client.put(self.url, data, format='multipart')
145     self.assertEqual(response.status_code, status.HTTP_200_OK)
146     self.dish1.refresh_from_db()
147     self.assertEqual(self.dish1.name, 'Updated Dish')
148     self.assertEqual(self.dish1.price, 13.50)
149
150 - def test_delete_dish(self):
151     self.client.force_authenticate(user=self.user)
152     response = self.client.delete(self.url)
153     self.assertEqual(response.status_code, status.HTTP_204_NO_CONTENT)
154     self.assertEqual(Dish.objects.count(), 1)
155
```

Settings:



Results:

All clear, no errors found

# dishes: urls.py



# dishes: views.py

 CI Python Linter

```
1 from django.shortcuts import render
2 from rest_framework import permissions, generics, filters
3 from django_filters.rest_framework import DjangoFilterBackend
4 from .models import Dish
5 from .serializers import DishSerializer, DishDetailSerializer
6
7
8 class DishList(generics.ListCreateAPIView):
9     """
10     API endpoint for listing and creating dishes.
11
12     GET: Retrieve a list of dishes, optionally filtered by dietary preference
13         or restaurant, searched by name or description, and ordered by
14         specified fields.
15
16     POST: Create a new dish. Requires authentication.
17     """
18     serializer_class = DishSerializer
19     queryset = Dish.objects.all()
20
21     filter_backends = [
22         filters.OrderingFilter,
23         filters.SearchFilter,
24         DjangoFilterBackend,
25     ]
26     filterset_fields = [
27         'dietary_preference',
28         'restaurant',
29     ]
30     search_fields = [
31         'name',
32         'short_description',
33         'dietary_preference',
34     ]
35     ordering_fields = [
36         'price',
```

Settings:

 ☒ 

Results:

All clear, no errors found

# drf\_api: asgi.py

 CI Python Linter

```
1 import os
2 from django.core.asgi import get_asgi_application
3
4
5 os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'drf_api.settings')
6
7 application = get_asgi_application()
8
```

Settings:

 ☒ 

---

Results:

All clear, no errors found



# drf\_api: permissions.py



## CI Python Linter

```
1 from rest_framework import permissions
2
3
4 class IsOwnerOrReadOnly(permissions.BasePermission):
5     """
6     Custom permission class that allows:
7     - Read-only access to any user.
8     - Full access (create, update, delete) only to the owner of the object.
9
10    Ownership is determined by checking for either an 'owner' or 'created_by'
11    attribute on the object.
12    """
13    def has_object_permission(self, request, view, obj):
14        if request.method in permissions.SAFE_METHODS:
15            return True
16
17        has_owner = hasattr(obj, 'owner')
18        has_created_by = hasattr(obj, 'created_by')
19
20        if has_owner and obj.owner == request.user:
21            return True
22        elif has_created_by and obj.created_by == request.user:
23            return True
24        else:
25            return False
26
```


Settings:

Results:

All clear, no errors found

# drf\_api: settings.py



## CI Python Linter

```
150 - {
151     'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
152 },
153 - {
154     'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
155 },
156 - {
157     'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
158 },
159 ]
160
161
162 # Internationalization
163 # https://docs.djangoproject.com/en/4.2/topics/i18n/
164
165 LANGUAGE_CODE = 'en-us'
166
167 TIME_ZONE = 'UTC'
168
169 USE_I18N = True
170
171 USE_TZ = True
172
173
174 # Static files (CSS, JavaScript, Images)
175 # https://docs.djangoproject.com/en/4.2/howto/static-files/
176
177 STATIC_URL = 'static/'
178 STATIC_ROOT = BASE_DIR / 'staticfiles'
179 WHITENOISE_ROOT = BASE_DIR / 'staticfiles' / 'build'
180
181 # Default primary key field type
182 # https://docs.djangoproject.com/en/4.2/ref/settings/#default-auto-field
183
184 DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
185
```

### Settings:

🌙 ☒ 🌞

---

### Results:

57: E501 line too long (80 > 79 characters)  
148: E501 line too long (91 > 79 characters)  
151: E501 line too long (81 > 79 characters)  
154: E501 line too long (82 > 79 characters)  
157: E501 line too long (83 > 79 characters)

# drf\_api: urls.py

 CI Python Linter

```
1 from django.contrib import admin
2 from django.urls import path, include
3 from django.views.generic import TemplateView
4 from .views import logout_route
5
6 urlpatterns = [
7     path('', TemplateView.as_view(template_name='index.html')),
8     path('admin/', admin.site.urls),
9     path('api/api-auth/', include('rest_framework.urls')),
10    path('api/dj-rest-auth/logout/', logout_route),
11    path('api/dj-rest-auth/', include('dj_rest_auth.urls')),
12    path(
13        'api/dj-rest-auth/registration/',
14        include('dj_rest_auth.registration.urls')
15    ),
16    path('api/', include('profiles.urls')),
17    path('api/', include('restaurants.urls')),
18    path('api/', include('dishes.urls')),
19    path('api/', include('reviews.urls')),
20    path('api/', include('likes.urls')),
21 ]
22
23 handler404 = TemplateView.as_view(template_name='index.html')
24
```

Settings:

 ☒ 

Results:

All clear, no errors found

# drf\_api: views.py



code institute CI Python Linter

```
1 from rest_framework.decorators import api_view
2 from rest_framework.response import Response
3 from .settings import (
4     JWT_AUTH_COOKIE, JWT_AUTH_REFRESH_COOKIE, JWT_AUTH_SAMESITE,
5     JWT_AUTH_SECURE,
6 )
7
8
9 # dj-rest-auth logout view fix
10 @api_view(['POST'])
11 def logout_route(request):
12     response = Response()
13     response.set_cookie(
14         key=JWT_AUTH_COOKIE,
15         value='',
16         httponly=True,
17         expires='Thu, 01 Jan 1970 00:00:00 GMT',
18         max_age=0,
19         samesite=JWT_AUTH_SAMESITE,
20         secure=JWT_AUTH_SECURE,
21     )
22     response.set_cookie(
23         key=JWT_AUTH_REFRESH_COOKIE,
24         value='',
25         httponly=True,
26         expires='Thu, 01 Jan 1970 00:00:00 GMT',
27         max_age=0,
28         samesite=JWT_AUTH_SAMESITE,
29         secure=JWT_AUTH_SECURE,
30     )
31     return response
32
```

Settings:  
🌙 ☒ 🌞

Results:  
All clear, no errors found

# drf\_api: wsgi.py



The screenshot shows a web-based interface for a Python linter. The header includes the 'code institute' logo and the title 'CI Python Linter'. The main area is split into two panels. The left panel displays a Python code snippet for a Django DRF WSGI application. The right panel contains settings and results sections. The settings section has a toggle switch for linting, which is currently turned on. The results section shows a message indicating that no errors were found.

```
1 import os
2 from django.core.wsgi import get_wsgi_application
3
4 os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'drf_api.settings')
5
6 application = get_wsgi_application()
7
```

Settings:

☒ ☐ ☐

Results:

All clear, no errors found

# restaurants: admin.py



# restaurants: apps.py



# restaurants: models.py

 CI Python Linter

```
1 from django.db import models
2 from django.contrib.auth.models import User
3
4
5 class Restaurant(models.Model):
6     """
7     Represents a restaurant, storing its name, location, cuisine type,
8     a short description, image, and creation/update timestamps.
9
10    Relationships:
11    - 'created_by': The user who created this restaurant
12      (ForeignKey to User).
13    - 'dishes': Dishes served at this restaurant
14      (reverse relationship from the Dish model).
15    - 'likes': Users who have liked/favorited this restaurant
16      (reverse relationship from the Like model).
17    - 'reviews': Reviews associated with this restaurant
18      (reverse relationship from the Review model).
19    """
20
21    cuisine_type_choices = [
22        ('italian', 'Italian'),
23        ('indian', 'Indian'),
24        ('german', 'German'),
25        ('japanese', 'Japanese'),
26        ('chinese', 'Chinese'),
27        ('thai', 'Thai'),
28        ('mexican', 'Mexican'),
29        ('greek', 'Greek'),
30        ('french', 'French'),
31        ('spanish', 'Spanish'),
32        ('vietnamese', 'Vietnamese'),
33        ('korean', 'Korean'),
34        ('other', 'Other'),
35    ]
36
```

Settings:


  

Results:

All clear, no errors found






# restaurants: serializers.py

 CI Python Linter

```
1 from rest_framework import serializers
2 from .models import Restaurant
3 from likes.models import Like
4
5
6 class RestaurantSerializer(serializers.ModelSerializer):
7     """
8     Serializer for the Restaurant model. Handles serialization and validation
9     of restaurant data.
10
11     Fields:
12     - 'created_by': Read-only field representing the username of the user who
13       created the restaurant.
14     - 'is_owner': Serializer method field indicating whether the current user
15       is the owner of the restaurant.
16     - 'like_id': Serializer method field representing the ID of the like object
17       if the current user has liked the restaurant, otherwise None.
18     - 'like_count': Read-only field representing the total number of likes for
19       the restaurant.
20     - 'review_count': Read-only field representing the total number of reviews
21       for the restaurant.
22
23     Validation:
24     - 'image': Ensures the uploaded image is not larger than 2MB and its
25       dimensions do not exceed 4096px in width or height.
26
27     Methods:
28     - 'get_is_owner': Determines if the requesting user is the owner of
29       the restaurant.
30     - 'get_like_id': Retrieves the ID of the like object if the current user
31       has liked the restaurant.
32     """
33     created_by = serializers.ReadOnlyField(source='created_by.username')
34     is_owner = serializers.SerializerMethodField()
35     like_id = serializers.SerializerMethodField()
36     like_count = serializers.ReadOnlyField()
```

Settings:

Results:

All clear, no errors found

# restaurants: tests.py



The screenshot displays the Code Institute CI Python Linter interface. The main editor area contains a Python file named `tests.py` with the following code:

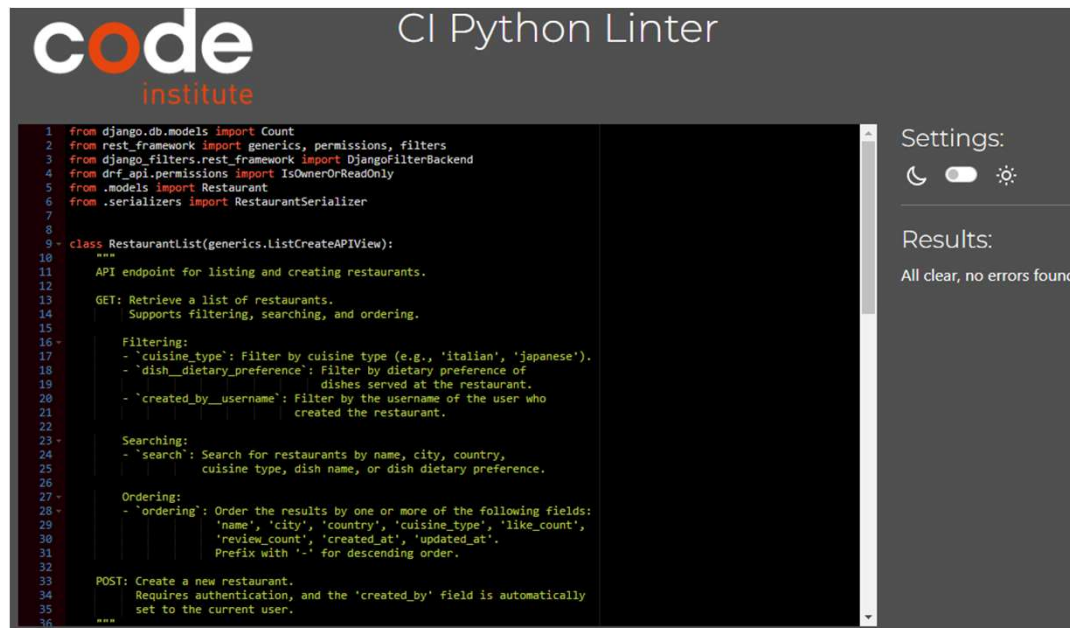
```
1 from django.test import TestCase
2 from django.contrib.auth.models import User
3 from django.core.files.uploadedfile import SimpleUploadedFile
4 from rest_framework.test import APIClient
5 from rest_framework import status
6 from .models import Restaurant
7 from PIL import Image
8 from io import BytesIO
9
10
11 class RestaurantListTestCase(TestCase):
12     def setUp(self):
13         self.user = User.objects.create_user(
14             username='testuser', password='testpassword'
15         )
16         self.restaurant1 = Restaurant.objects.create(
17             created_by=self.user,
18             name='Italian Restaurant',
19             city='Rome',
20             country='Italy',
21             image='uploaded_image',
22             short_description='example',
23             cuisine_type='italian',
24         )
25         self.restaurant2 = Restaurant.objects.create(
26             created_by=self.user,
27             name='Japanese Restaurant',
28             city='Tokyo',
29             country='Japan',
30             image='uploaded_image',
31             short_description='example',
32             cuisine_type='japanese',
33         )
34
35     self.client = APIClient()
36
```

On the right side, the **Settings:** panel shows a dark mode toggle switch that is currently turned on. Below it, the **Results:** panel displays the message "All clear, no errors found".

# restaurants: urls.py



# restaurants: views.py



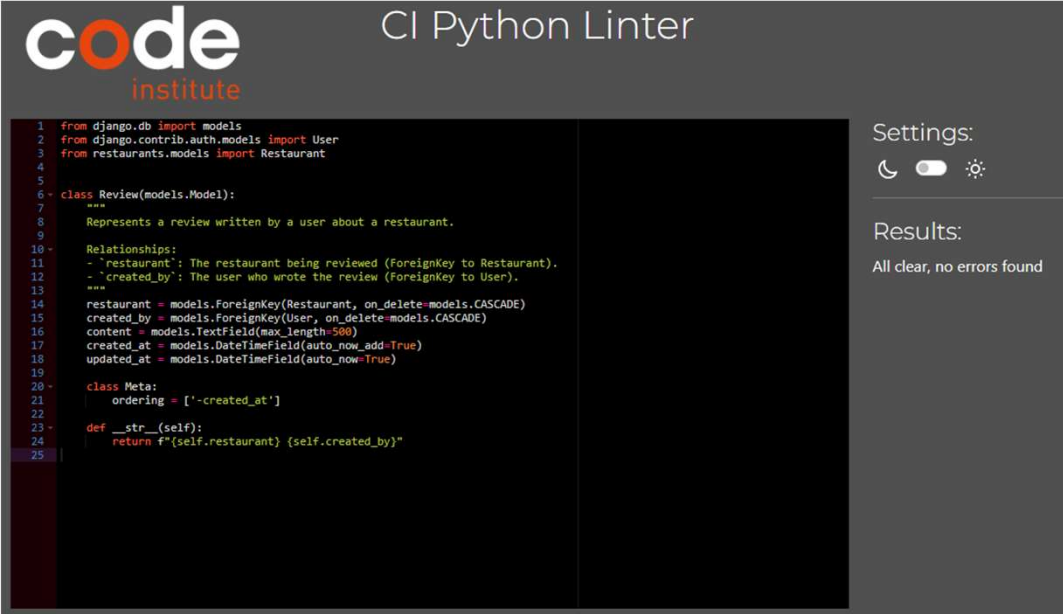
# reviews: admin.py



# reviews: apps.py



# reviews: models.py

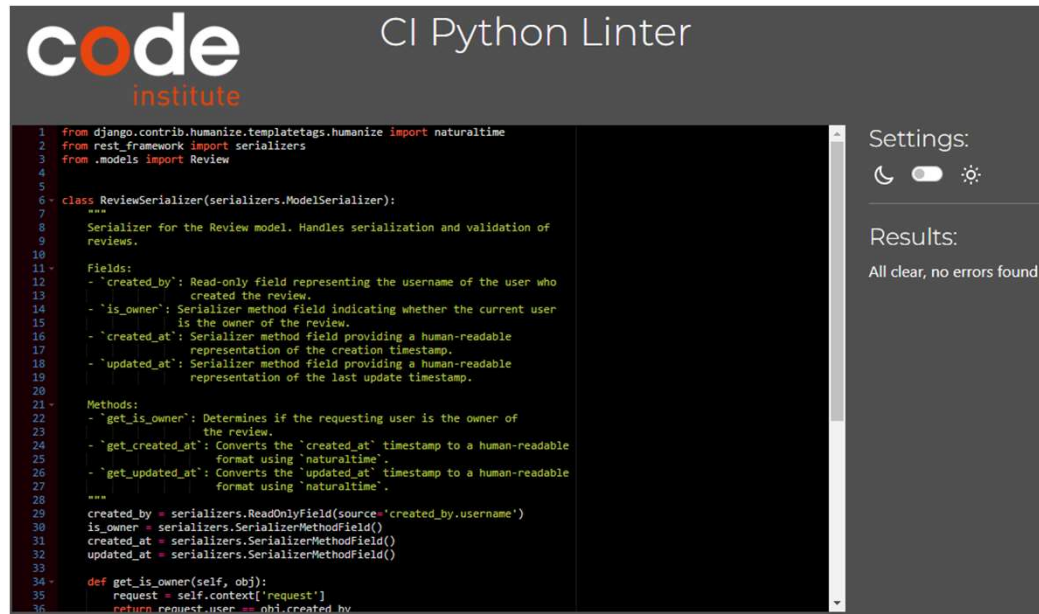


The screenshot shows the Code Institute CI Python Linter interface. The main area displays a Python model for a 'Review' class. The code is as follows:

```
1 from django.db import models
2 from django.contrib.auth.models import User
3 from restaurants.models import Restaurant
4
5
6 class Review(models.Model):
7     """
8     Represents a review written by a user about a restaurant.
9
10    Relationships:
11    - 'restaurant': The restaurant being reviewed (ForeignKey to Restaurant).
12    - 'created_by': The user who wrote the review (ForeignKey to User).
13    """
14    restaurant = models.ForeignKey(Restaurant, on_delete=models.CASCADE)
15    created_by = models.ForeignKey(User, on_delete=models.CASCADE)
16    content = models.TextField(max_length=500)
17    created_at = models.DateTimeField(auto_now_add=True)
18    updated_at = models.DateTimeField(auto_now=True)
19
20    class Meta:
21        ordering = ['-created_at']
22
23    def __str__(self):
24        return f"{self.restaurant} {self.created_by}"
25
```

On the right side of the interface, there are two sections: 'Settings:' with a dark mode toggle switch and a settings gear icon, and 'Results:' which displays the message 'All clear, no errors found'.

# reviews: serializers.py



The screenshot displays the Code Institute CI Python Linter interface. The main window shows a Python file named `reviews: serializers.py` with the following content:

```
1 from django.contrib.humanize.templatetags.humanize import naturaltime
2 from rest_framework import serializers
3 from .models import Review
4
5
6 class ReviewSerializer(serializers.ModelSerializer):
7     """
8     Serializer for the Review model. Handles serialization and validation of
9     reviews.
10
11     Fields:
12     - 'created_by': Read-only field representing the username of the user who
13       created the review.
14     - 'is_owner': Serializer method field indicating whether the current user
15       is the owner of the review.
16     - 'created_at': Serializer method field providing a human-readable
17       representation of the creation timestamp.
18     - 'updated_at': Serializer method field providing a human-readable
19       representation of the last update timestamp.
20
21     Methods:
22     - 'get_is_owner': Determines if the requesting user is the owner of
23       the review.
24     - 'get_created_at': Converts the 'created_at' timestamp to a human-readable
25       format using 'naturaltime'.
26     - 'get_updated_at': Converts the 'updated_at' timestamp to a human-readable
27       format using 'naturaltime'.
28     """
29     created_by = serializers.ReadOnlyField(source='created_by.username')
30     is_owner = serializers.SerializerMethodField()
31     created_at = serializers.SerializerMethodField()
32     updated_at = serializers.SerializerMethodField()
33
34     def get_is_owner(self, obj):
35         request = self.context['request']
36         return request.user == obj.created_by
```

On the right side of the interface, there is a **Settings:** section with a dark mode toggle (currently off) and a **Results:** section showing the message: "All clear, no errors found".



# reviews: tests.py



The screenshot displays the Code Institute CI Python Linter interface. On the left, a code editor shows a Django test file named `reviews: tests.py`. The code includes imports for `TestCase`, `User`, `APIClient`, `status`, `Review`, and `Restaurant`. It defines a `ReviewListTestCase` class with a `setUp` method that creates a test user, a restaurant, and two reviews. The right sidebar contains a 'Settings' section with a dark mode toggle and a 'Results' section showing 'All clear, no errors found'.

```
1 from django.test import TestCase
2 from django.contrib.auth.models import User
3 from rest_framework.test import APIClient
4 from rest_framework import status
5 from .models import Review
6 from restaurants.models import Restaurant
7
8
9 class ReviewListTestCase(TestCase):
10     def setUp(self):
11         self.user = User.objects.create_user(
12             username='testuser', password='testpassword'
13         )
14         self.restaurant = Restaurant.objects.create(
15             created_by=self.user,
16             name='Italian Restaurant',
17             city='Rome',
18             country='Italy',
19             image='uploaded_image',
20             short_description='example',
21             cuisine_type='italian',
22         )
23         self.review1 = Review.objects.create(
24             restaurant=self.restaurant,
25             created_by=self.user,
26             content='Great food!',
27         )
28         self.review2 = Review.objects.create(
29             restaurant=self.restaurant,
30             created_by=self.user,
31             content='Amazing service!',
32         )
33
34     self.client = APIClient()
35
36     self.url = '/reviews/'
```

code institute

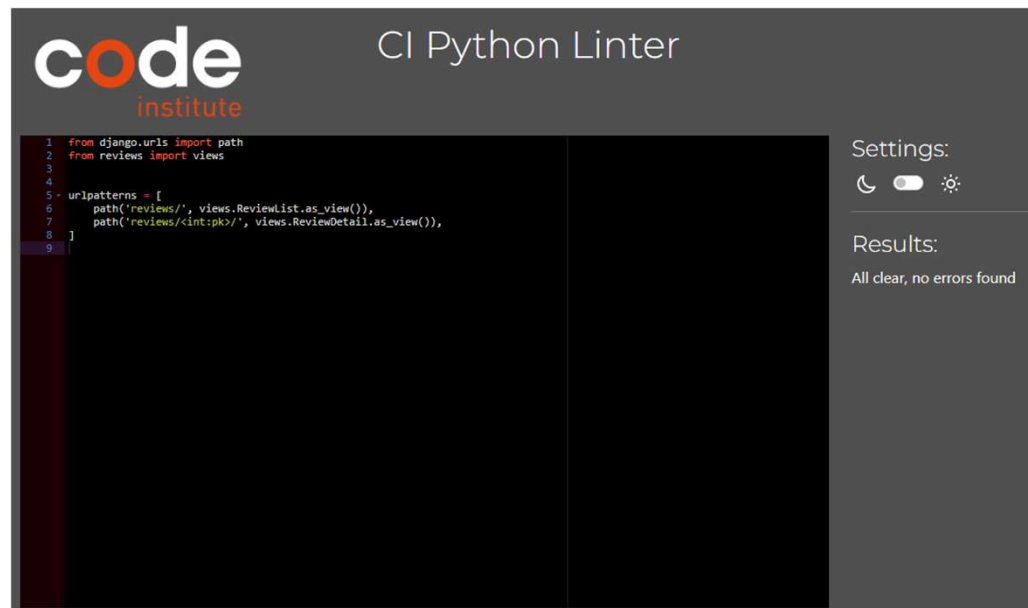
CI Python Linter

Settings:

Results:

All clear, no errors found

# reviews: urls.py



# reviews: views.py



code institute CI Python Linter

```
1 from rest_framework import generics, permissions, filters
2 from django_filters.rest_framework import DjangoFilterBackend
3 from drf_api.permissions import IsOwnerOrReadOnly
4 from .models import Review
5 from .serializers import ReviewSerializer, ReviewDetailSerializer
6
7
8 class ReviewList(generics.ListCreateAPIView):
9     """
10     API endpoint for listing and creating reviews.
11
12     GET: Retrieve a list of reviews. Supports filtering, searching, and
13         ordering.
14
15     Filtering:
16     - 'restaurant': Filter by the restaurant the reviews
17         are associated with.
18
19     Searching:
20     - 'search': Search for reviews by the username of the creator or
21         the content of the review.
22
23     Ordering:
24     - 'ordering': Order the results by 'updated_at'.
25         Prefix with '-' for descending order.
26
27     POST: Create a new review.
28         Requires authentication, and the 'created_by' field is automatically
29         set to the current user.
30     """
31     serializer_class = ReviewSerializer
32     permission_classes = [
33         permissions.IsAuthenticatedOrReadOnly
34     ]
35     queryset = Review.objects.all()
36     filter_backends = [
```

Settings:  
🌙 ☒ ☐ ☀️

Results:  
All clear, no errors found