

For the validation process the CI Python Linter from Code Institute was used.
<https://pep8ci.herokuapp.com/>

newsfeed: views.py



CI Python Linter

```
276
277 @login_required
278 @user_passes_test(lambda u: u.groups.filter(name='Moderator').exists())
279 def approve_comment(request, slug, comment_id):
280     """
281     Approves a comment, making it visible on the article detail page.
282
283     Retrieves:
284     - The comment object to be approved based on the comment_id.
285
286     Handles:
287     - Sets the comment's 'approved' status to True.
288     - Saves the updated comment.
289     - Adds a success message to the request.
290
291     Arguments:
292     - slug: The slug of the article the comment belongs to
293           (not directly used in the current implementation).
294     - comment_id: The ID of the comment to be approved.
295
296     Returns:
297     - HttpResponseRedirect: Redirects to the article detail page after
298                           processing the request.
299
300     Permissions:
301     - Requires the user to be logged in and belong to the 'Moderator' group.
302     """
303     comment = get_object_or_404(Comment, id=comment_id)
304     comment.approved = True
305     comment.save()
306     messages.add_message(
307         request, messages.SUCCESS,
308         'Approval was successful!'
309     )
310     return redirect('article_detail', slug=comment.article.slug)
311
```

Settings:



Results:

All clear, no errors found

newsfeed: urls.py



CI Python Linter

```
1 from django.urls import path
2 from . import views
3
4 urlpatterns = [
5     path('', views.article_list, name='home'),
6     path('about/', views.about, name='about'),
7     path('<slug:slug>/', views.article_detail, name='article_detail'),
8     path(
9         '<slug:slug>/edit_comment/<int:comment_id>',
10        views.comment_edit, name='comment_edit'
11    ),
12     path('<slug:slug>/delete_comment/<int:comment_id>',
13        views.comment_delete, name='comment_delete'),
14     path('<slug:slug>/like/', views.article_like, name='article_like'),
15     path(
16         '<slug:slug>/approve_comment/<int:comment_id>/',
17        views.approve_comment, name='approve_comment'
18    ),
19 ]
```


Settings:



Results:



All clear, no errors found

newsfeed: models.py

 CI Python Linter

```
81 ordering = ["-created_on"]
82
83 - def __str__(self):
84     """
85     Returns a string representation of the comment in the format:
86     "Comment on - [article title] - by - [author username] -".
87     """
88     return f"Comment on - {self.article.title} - by - {self.author} -"
89
90
91 - class Like(models.Model):
92     """
93     Represents a "like" action by a user on an article or comment.
94
95     Relationships:
96     - `user`: The user who liked the content (ForeignKey to User).
97     - `content_type`: The type of content that was liked
98       (ForeignKey to ContentType).
99     - `content_object`: The actual object that was liked
100       (either an Article or a Comment would be possible),
101       determined dynamically using
102       `content_type` and `object_id`.
103     """
104     user = models.ForeignKey(
105         User, on_delete=models.SET_NULL, null=True, blank=True)
106     content_type = models.ForeignKey(ContentType, on_delete=models.CASCADE)
107     object_id = models.PositiveIntegerField()
108     content_object = GenericForeignKey('content_type', 'object_id')
109
110 - def __str__(self):
111     """
112     Returns a string representation of the like in the format:
113     "Like by [user username] on [liked object __str__]".
114     """
115     return f"Like by {self.user} on {self.content_object}"
116
```

Settings:

 ☐ 

Results:

All clear, no errors found

newsfeed: forms.py



CI Python Linter

```
1 from django import forms
2 from .models import Comment
3
4
5 class CommentForm(forms.ModelForm):
6     """
7     A Django ModelForm for creating and editing comments.
8     """
9     class Meta:
10         model = Comment
11         fields = ['content', ]
12         widgets = {
13             'content': forms.Textarea(
14                 attrs={'class': 'form-control', 'id': 'id_body'})
15         }
16
```

Settings:

 ☒ 

Results:

All clear, no errors found

newsfeed: apps.py



newsfeed: admin.py

 CI Python Linter

```
1 from django.contrib import admin
2 from django_summernote.admin import SummernoteModelAdmin
3 from .models import Article, Comment, Like
4
5
6 @admin.register(Article)
7 class ArticleAdmin(SummernoteModelAdmin):
8     """
9     Customizes the Django admin interface for the Article model.
10    """
11    list_display = ('title', 'author', 'created_on', 'status',)
12    list_filter = ('status', 'created_on', 'author')
13    prepopulated_fields = {'slug': ('title',)}
14    summernote_fields = ('content', 'excerpt',)
15
16
17 @admin.register(Comment)
18 class CommentAdmin(SummernoteModelAdmin):
19     """
20     Customizes the Django admin interface for the Comment model.
21    """
22    list_display = ('author', 'article', 'created_on', 'approved')
23    list_filter = ('author', 'created_on', 'approved')
24
25
26 # Registration of Like model
27 admin.site.register(Like)
28
```

Settings:

Results:

All clear, no errors found

newsfeed: test_forms.py



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

```
1 from django.test import TestCase
2 from .forms import CommentForm
3
4
5 class TestCommentForm(TestCase):
6     """All tests for the comment form"""
7     def test_form_is_valid(self):
8         """Test to check the comment form is valid"""
9         comment_form = CommentForm({'content': 'Great Post!'})
10        self.assertTrue(comment_form.is_valid())
11
12    def test_form_is_invalid(self):
13        """Test if comment form is correctly invalid"""
14        comment_form = CommentForm({'content': ''})
15        self.assertFalse(comment_form.is_valid())
16
```

On the right side, the **Settings:** section includes a dark mode toggle (currently on) and a sun icon. The **Results:** section shows the message: "All clear, no errors found".

newsfeed: test_views.py



The screenshot displays the Code Institute CI Python Linter interface. The main area shows a Python file with test code. The code includes a class method for saving an article and a test function for approving a comment. The right sidebar shows the settings and results, indicating that all checks passed with no errors found.

```
235         title="Article title",
236         slug="article-title",
237         author=self.user,
238         is_breaking_news=True,
239         excerpt="Article Excerpt",
240         content="Article Content",
241         status=1,
242     )
243     self.article.save()
244
245     self.comment = Comment.objects.create(
246         article=self.article,
247         author=self.user,
248         content='Test comment',
249         approved=False
250     )
251
252     def test_approve_comment_as_moderator(self):
253         """Test to see if the moderator can approve a comment"""
254         self.client.login(
255             username='moderator',
256             password='moderatorpassword'
257         )
258
259         response = self.client.post(
260             reverse(
261                 'approve_comment',
262                 args=[self.article.slug, self.comment.id]))
263
264         self.assertEqual(response.status_code, 302)
265         self.assertRedirects(
266             response, reverse('article_detail', args=[self.article.slug])
267         )
268         updated_comment = Comment.objects.get(pk=self.comment.id)
269         self.assertTrue(updated_comment.approved)
270
```

code institute

CI Python Linter

Settings:

Results:

All clear, no errors found

good_world_news: wsgi.py



good_world_news: urls.py

 CI Python Linter

```
1 from django.contrib import admin
2 from django.urls import path, include
3
4 urlpatterns = [
5     path('admin/', admin.site.urls),
6     path("accounts/", include("allauth.urls")),
7     path('summernote/', include('django_summernote.urls')),
8     path('contact/', include('contact.urls'), name='contact'),
9     path('', include("newsfeed.urls"), name='newsfeed-urls'),
10 ]
11
```


Settings:

Results:



All clear, no errors found

good_world_news: settings.py

 CI Python Linter

```
124         'NAME': 'django.contrib.auth.password_validation.' +
125             'NumericPasswordValidator',
126     },
127 ]
128
129 ACCOUNT_EMAIL_VERIFICATION = 'none'
130
131
132 # Internationalization
133 # https://docs.djangoproject.com/en/4.2/topics/i18n/
134
135 LANGUAGE_CODE = 'en-us'
136
137 TIME_ZONE = 'UTC'
138
139 USE_I18N = True
140
141 USE_TZ = True
142
143
144 # Static files (CSS, JavaScript, Images)
145 # https://docs.djangoproject.com/en/4.2/howto/static-files/
146
147 STATIC_ROOT = os.path.join(BASE_DIR, 'staticfiles')
148 STATIC_URL = 'static/'
149 STATICFILES_DIRS = [os.path.join(BASE_DIR, 'static'), ]
150 STATICFILES_STORAGE = 'whitenoise.storage.CompressedManifestStaticFilesStorage'
151
152 # Default primary key field type
153 # https://docs.djangoproject.com/en/4.2/ref/settings/#default-auto-field
154
155 DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
156
157 # Custom adapter to redirect different user groups after logging in
158 ACCOUNT_ADAPTER = 'good_world_news.adapters.CustomAccountAdapter'
159
```

Settings:

 ☒ 

Results:

All clear, no errors found

good_world_news: asgi.py

 CI Python Linter

```
1 """
2 ASGI config for good_world_news 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/4.2/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', 'good_world_news.settings')
15
16 application = get_asgi_application()
17
```

Settings:

Results:

All clear, no errors found

contact: views.py

 CI Python Linter

```
1 from django.shortcuts import render, redirect
2 from django.contrib import messages
3 from .forms import ContactForm
4
5
6 def contact_form(request):
7     """
8     Handles the rendering and submission of the contact form.
9
10    - GET requests: Renders the contact form template with an empty
11      form instance.
12    - POST requests: Processes the submitted contact form data.
13      - If the form is valid:
14        - Saves the form data to the database.
15        - Adds a success message to the request.
16        - Redirects to the homepage.
17
18    Template:
19    - Renders the 'contact/contact.html' template.
20    """
21    if request.method == 'POST':
22        form = ContactForm(request.POST)
23        if form.is_valid():
24            form.save()
25
26            messages.success(request, 'Your messages was sent successfully.')
27            return redirect('home')
28    else:
29        form = ContactForm()
30    return render(request, 'contact/contact.html', {'form': form})
31
32
```

Settings:

Results:

All clear, no errors found

contact: urls.py



contact: models.py



The screenshot displays the Code Institute CI Python Linter interface. On the left, a code editor shows the content of `contact: models.py`. The code defines a `Contact` model with fields for `name`, `email`, `reason`, `message`, and `created_on`. It also includes a `Meta` class for ordering and a `__str__` method. On the right, the 'Settings' panel shows a dark theme and a toggle switch. The 'Results' panel indicates 'All clear, no errors found'.

```
1 from django.db import models
2 from django.core.validators import MinLengthValidator
3
4 CHOICES = [
5     ('question', 'Question'),
6     ('feedback', 'Feedback'),
7     ('collaboration', 'Collaboration Request'),
8     ('other', 'Other')
9 ]
10
11
12 class Contact(models.Model):
13     """
14     Represents a contact message submitted by a user.
15     """
16     name = models.CharField(validators=[MinLengthValidator(3)], max_length=200)
17     email = models.EmailField()
18     reason = models.CharField(
19         max_length=30, choices=CHOICES, default='question')
20     message = models.TextField(
21         validators=[MinLengthValidator(10)], max_length=500)
22     created_on = models.DateTimeField(auto_now_add=True)
23
24 class Meta:
25     """
26     Orders contact messages by their creation date in descending order.
27     """
28     ordering = ["-created_on"]
29
30 def __str__(self):
31     """
32     Returns a string representation of the contact message in the format:
33     "[reason] by [name]".
34     """
35     return f"{self.reason} by {self.name}"
36
```

code institute CI Python Linter

Settings:
🌙 ☒

Results:
All clear, no errors found

contact: forms.py

 CI Python Linter

```
1 from django import forms
2 from .models import Contact
3
4
5 - class ContactForm(forms.ModelForm):
6     """
7     A Django ModelForm for creating and editing contact messages.
8
9     Fields:
10     - name: The name of the person submitting the message
11         (single-line text input).
12     - email: The email address of the person submitting the message
13         (email input with validation).
14     - reason: The reason for the contact, selected from a dropdown menu
15         (choices defined in the Contact model).
16     - message: The main content of the contact message (textarea).
17     """
18
19     class Meta:
20         model = Contact
21         fields = ['name', 'email', 'reason', 'message']
22         widgets = {
23             'name': forms.TextInput(attrs={'class': 'form-control'}),
24             'email': forms.EmailInput(attrs={'class': 'form-control'}),
25             'reason': forms.Select(attrs={'class': 'form-select'}),
26             'message': forms.Textarea(attrs={'class': 'form-control'})
27         }
```

Settings:

 ☒ 

Results:

All clear, no errors found

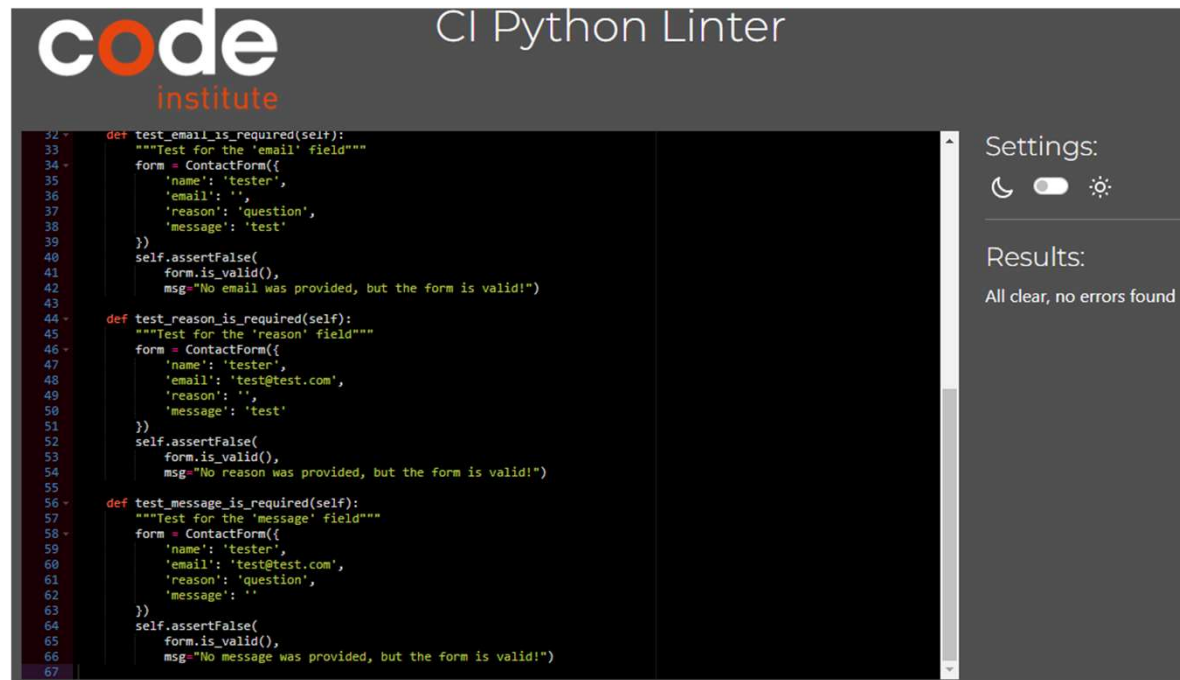
contact: apps.py



contact: admin.py



contact: test_forms.py



The screenshot displays the Code Institute CI Python Linter interface. The main area shows a Python test file with three test functions: `test_email_is_required`, `test_reason_is_required`, and `test_message_is_required`. Each function creates a `ContactForm` instance and asserts that `form.is_valid()` returns `False` with a specific error message. The right sidebar contains a 'Settings' section with a dark mode toggle and a 'Results' section showing 'All clear, no errors found'.

```
32 - def test_email_is_required(self):
33     """Test for the 'email' field"""
34 -     form = ContactForm({
35         'name': 'tester',
36         'email': '',
37         'reason': 'question',
38         'message': 'test'
39     })
40     self.assertFalse(
41         form.is_valid(),
42         msg="No email was provided, but the form is valid!")
43
44 - def test_reason_is_required(self):
45     """Test for the 'reason' field"""
46 -     form = ContactForm({
47         'name': 'tester',
48         'email': 'test@test.com',
49         'reason': '',
50         'message': 'test'
51     })
52     self.assertFalse(
53         form.is_valid(),
54         msg="No reason was provided, but the form is valid!")
55
56 - def test_message_is_required(self):
57     """Test for the 'message' field"""
58 -     form = ContactForm({
59         'name': 'tester',
60         'email': 'test@test.com',
61         'reason': 'question',
62         'message': ''
63     })
64     self.assertFalse(
65         form.is_valid(),
66         msg="No message was provided, but the form is valid!")
67
```

code institute


CI Python Linter

Settings:

Results:

All clear, no errors found

contact: test_views.py



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

```
1 from django.test import TestCase
2 from django.urls import reverse
3 from .forms import ContactForm
4
5
6 class TestContactView(TestCase):
7     """All tests for the contact view"""
8     def test_render_contact_form(self):
9         """Test for rendering the contact form correctly"""
10        response = self.client.get(reverse('contact'))
11        self.assertEqual(response.status_code, 200)
12        self.assertIsInstance(
13            response.context['form'], ContactForm
14        )
15
16    def test_successful_contact_form_submission(self):
17        """Test for someone trying to contact the website owner"""
18        contact_data = {
19            'name': 'test name',
20            'email': 'test@test.com',
21            'reason': 'question',
22            'message': 'This is my message'
23        }
24        response = self.client.post(
25            reverse('contact'), contact_data, follow=True)
26        self.assertRedirects(response, reverse('home'))
27        self.assertIn(
28            b"Your message was sent successfully.",
29            response.content)
30
```

On the right side, the **Settings:** section includes icons for a dark theme, a toggle switch, and a sun icon. The **Results:** section shows the message: "All clear, no errors found".