



**TASK**

# **Django – Sticky Notes Application Part 2**

Visit our website


# Introduction

## WELCOME TO THE DJANGO – STICKY NOTES APPLICATION PART 2 TASK!

In this task, we are going to guide you through unit testing and debugging the bulletin board Django application. Django has some powerful built-in tools such as **TestCase** to facilitate this process. By effectively utilising Django's testing tools and best practices, you can significantly improve the quality and reliability of your applications. Let's get started!

### UNIT TESTING

As you know, unit testing is a fundamental aspect of software development that ensures individual components of your code function as expected. In the context of Django, this means testing isolated parts of your application like models, views, forms, and URL patterns.



**Extra resource**

Explore how to [write and run tests with Django](#) by referring to the Django documentation.

## Step 1: Testing for the bulletin board app

Create tests in the **posts/tests.py** file:

```
# posts/tests.py
from django.test import TestCase
from django.urls import reverse
from .models import Post, Author

class PostModelTest(TestCase):
    def setUp(self):
        # Create an Author object
        author = Author.objects.create(name='Test Author')
        # Create a Post object for testing
        Post.objects.create(title='Test Post', content='This is a test
post.', author=author)

    def test_post_has_title(self):
        # Test that a Post object has the expected title
        post = Post.objects.get(id=1)
        self.assertEqual(post.title, 'Test Post')

    def test_post_has_content(self):
        # Test that a Post object has the expected content
        post = Post.objects.get(id=1)
        self.assertEqual(post.content, 'This is a test post.')

class PostViewTest(TestCase):
    def setUp(self):
        # Create an Author object
        author = Author.objects.create(name='Test Author')
        # Create a Post object for testing views
        Post.objects.create(title='Test Post', content='This is a test
post.', author=author)

    def test_post_list_view(self):
        # Test the post-list view
        response = self.client.get(reverse('post_list'))
        self.assertEqual(response.status_code, 200)
        self.assertContains(response, 'Test Post')

    def test_post_detail_view(self):
        # Test the post-detail view
        post = Post.objects.get(id=1)
        response = self.client.get(reverse('post_detail',
args=[str(post.id)]))
        self.assertEqual(response.status_code, 200)
```

```
self.assertContains(response, 'Test Post')
self.assertContains(response, 'This is a test post.')
```

Explanation:

- **PostModelTest:** This class contains tests for the **Post** model. It checks whether a post has the expected title and content.
- **PostViewTest:** This class contains tests for views related to posts. It checks if the post list view displays a post, and if the post detail view shows the correct title and content.

These tests use Django's **TestCase** class and various assertion methods to check whether the behaviour of your models and views is as expected.

## Step 2: Run tests

To run the tests, run **manage.py** in the command line:

```
python manage.py test posts
```

You should see the following in your terminal:

```
Found 4 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
....
-----
Ran 4 tests in 0.052s

OK
Destroying test database for alias 'default'...
```

Explanation:

- **Found 4 test(s).**: This indicates that the test runner discovered and identified a total of four tests to run.
- **Creating test database for alias 'default'...**: Before running tests, Django creates a test database to ensure a clean and isolated environment for testing.
- **System check identified no issues (0 silenced).**: The system check ensures that there are no issues with your Django project that could interfere with the tests.
- **....**: Each dot represents a test case that was run, and in this case, there are four dots, indicating that four tests were executed.
- **Ran 4 tests in 0.052s**: This line shows the total number of tests executed (4) and the time it took to run them (0.052 seconds).
- **OK**: This indicates that all tests passed successfully. If there were failures or errors, this status would be different.
- **Destroying test database for alias 'default'...**: After running the tests, Django destroys the test database, cleaning up the temporary environment created for testing.

In summary, a test output of “OK” means that all four tests ran without issues and passed successfully. This is the expected outcome, indicating that your tests are providing the expected results for the defined test cases.

The above sections collectively illustrate the essential components of a Django application, specifically models, views, forms, and URL patterns.



## Practical task

In this task, you will further develop your sticky notes application from the previous task and test it. During this task, you are welcome to refactor your application and improve it if you feel there are any features you wish to improve upon. Please follow these steps:

1. Design and implement tests to ensure they cover the use cases of the sticky notes application.
2. Run the tests you've created to validate the correctness of your implementation.
3. Ensure that you also manually test your application to ensure that there are no obvious bugs.

4. Your completed sticky notes task manager application should be saved in a folder in your task folder. In this folder, you will need to include all the components of the project, including:
  - All your design diagrams
  - Unit tests
  - Project directory for the final app, including the source code
5. Please **exclude** the **venv** folder; it is unnecessary as the mentor reviewing your work can generate it. It's good practice to always delete generated files since they may make it difficult to set up your app on other machines and they take up a lot of storage space.
6. Add your completed sticky notes application to your GitHub account as part of your developer portfolio. Ensure your repository is public and remember to exclude the virtual environment folder before uploading. Share a link to your sticky notes repository in a text file called **sticky\_github.txt**.

**Important:** Be sure to upload all files required for the task submission inside your task folder and then click "Request review" on your dashboard.



## Share your thoughts

Please take some time to complete this short feedback [form](#) to help us ensure we provide you with the best possible learning experience.