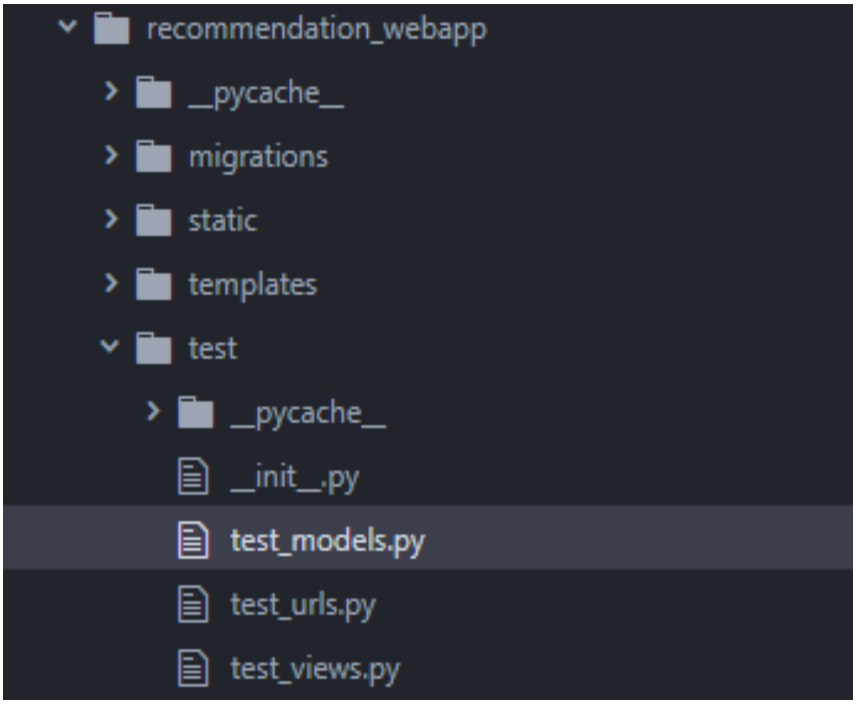
Unit Testing in Python

This document contains information regarding the unit testing performed in this project.



A test folder is created in the recommendation_webapp folder and the original test.py is deleted.

We rather create three separate python files to perform tests on URLs, views, models of this application.

While testing in Django, each time a test is performed, **a default empty database is created and it is destroyed at the end of the test case.**

All test methods must start with the term ‘test’.

Test_url.py

test_urls.py | test_models.py

```python
from django.test import SimpleTestCase
from django.urls import reverse, resolve
from recommendation_webapp.views import home, all_movies, book_recommendat

class TestUrls(SimpleTestCase):

    #Home Url Test
    def test_home_url(self):
        url = reverse('home')
        self.assertEquals(resolve(url).func, home)

    #Movie Quick Search Url test
    def test_all_movies_url(self):
        url = reverse('all_movies')
        self.assertEquals(resolve(url).func, all_movies)

    #Book recommendation Engine url test
    def test_book_recommendations_url(self):
        url = reverse('book_recommendations')
        self.assertEquals(resolve(url).func, book_recommendations)

    #Anime similarity filtering url test
    def test_anime_similar_filtering_url(self):
        url = reverse('anime_similar_filtering')
        self.assertEquals(resolve(url).func, anime_similar_filtering)

    #Movie pre filter url test
    def test_movie_pre_filter_url(self):
        url = reverse('movie_pre_filter')
        self.assertEquals(resolve(url).func, movie_pre_filter)
```

Here, different test cases check whether a particular URL accesses the right function defined for that URL.

Test_views.py

test_views.py | test_urls.py | test_models.py

```
from django.test import TestCase, Client
from django.urls import reverse
from recommendation_webapp.models import Movie,Book,Anime2

class TestViews(TestCase):

    def setUp(self):
        self.client = Client()

        #Inserting sample objects in the dataset
        Movie.objects.create(Title='Goodfellas',Genre='Crime',IMDB_Score='8.7')
        Movie.objects.create(Title='Captain America',Genre='Action',IMDB_Score='6.9')
        Movie.objects.create(Title='Hangover',Genre='Comedy',IMDB_Score='7.7')

        Book.objects.create(name='Harry Potter and the Chamber of Secrects',author='J.K.
        Book.objects.create(name='Barking Up the wrong tree',author='Eric Barker',categ
        Book.objects.create(name='The Immortals of Meluha',author='Amish Tripathi',categ

        Anime2.objects.create(title='Baki',genre='action')
        Anime2.objects.create(title='Dr. Stone',genre='adventure')
        Anime2.objects.create(title='Shin-Chan',genre='comedy')

    #Home function Check
    def test_home_get(self):
        response = self.client.get(reverse('home'))
        self.assertEquals(response.status_code,200)
        self.assertTemplateUsed(response,'index.html')
```

Since an empty database is created for every test case, we define a **setUp** method to be performed before every test case.

This method adds some sample objects in all three models so that various test cases can be performed on them.

Here, we check whether we hit the right status codes and also we check whether the appropriate template is rendered for the tested view with different parameters passed to it.

Test_models.py

test_models.py | test_views.py | test_urls.py

```python
from django.test import TestCase, Client
from recommendation_webapp.models import Movie,Anime2,Book

class TestModels(TestCase):

    #Test the Movie model
    def test_Movie_model(self):
        movie = Movie.objects.create(Title='Fight Club',Genre='Crime',IMDB_Score='8.7')
        max_length = Movie._meta.get_field('Title').max_length
        self.assertEquals(max_length, 500)
        self.assertEquals(str(movie),'Fight Club')

    #Test the Book model
    def test_Book_model(self):
        book = Book.objects.create(name='Harry Potter and the Chamber of Secrects',author=
        max_length = Book._meta.get_field('name').max_length
        self.assertEquals(max_length, 500)
        self.assertEquals(str(book),'Harry Potter and the Chamber of Secrects')

    #Test the Anime2 model
    def test_Anime2_model(self):
        anime = Anime2.objects.create(title='Dragon Ball',genre='action')
        max_length = Anime2._meta.get_field('title').max_length
        self.assertEquals(max_length, 500)
        self.assertEquals(str(anime),'Dragon Ball')
```

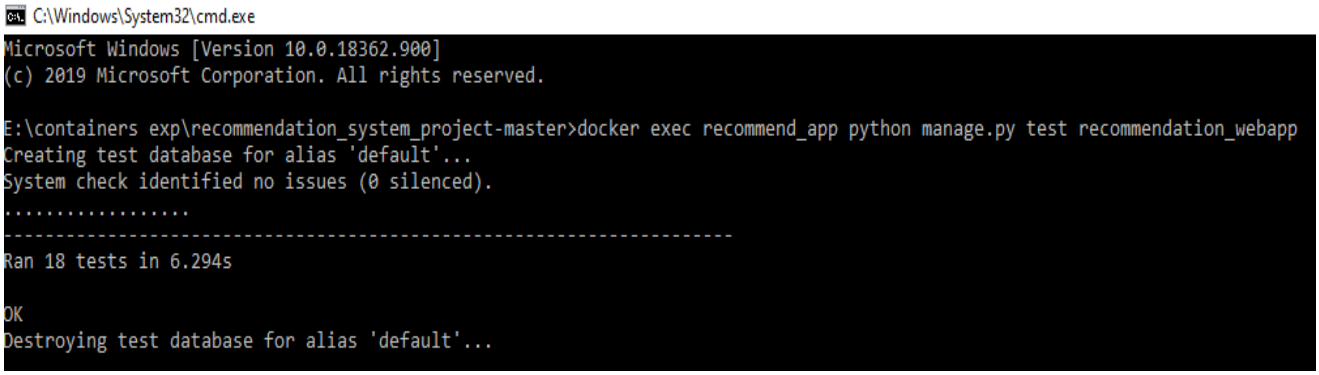
Here we test the three models: Movie, Book and Anime2

Sample objects can be added and verified in the models.

Also, we can test the models field parameters passed to the models like max length.

Running tests

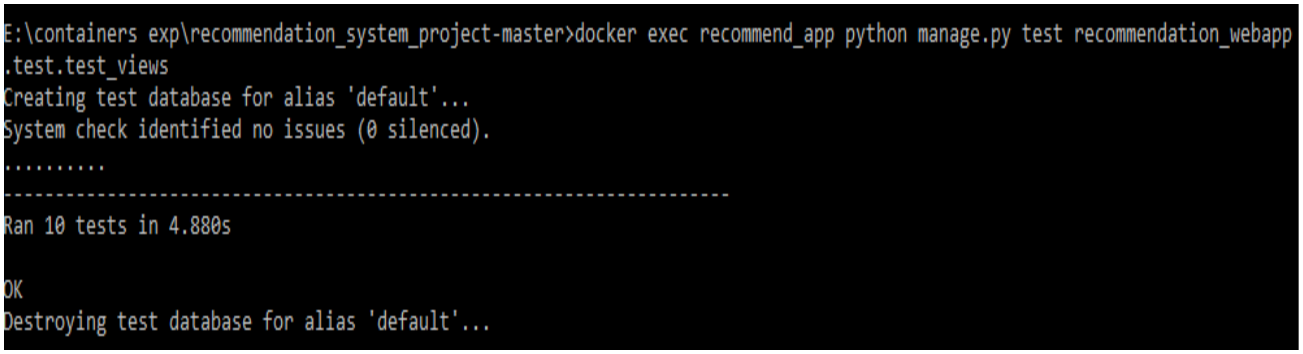
1. Running all tests



Make sure the containers are running when performing test.

Command: docker exec recommend_app python manage.py test recommendation_webapp

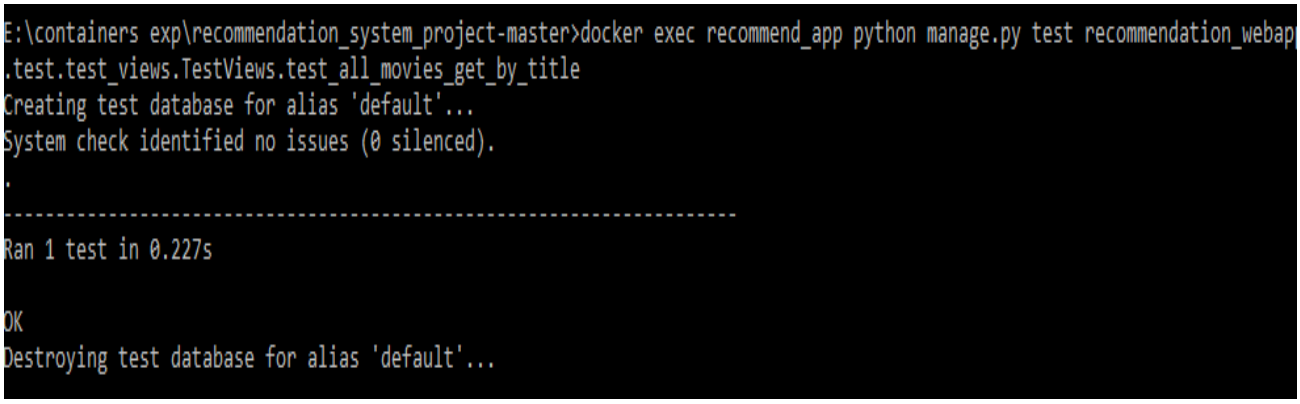
2. Run test on a particular python file



Command: docker exec recommend_app python manage.py test recommendation_webapp.test.test_views

Choose whichever file you prefer.

3. Run test on individual method



Command: docker exec recommend_app python manage.py test recommendation_webapp.test.test_views.TestViews.test_all_movies_get_by_title

Here, TestViews is the name of class mentioned in test_views.py

There are total 18 test cases written in this application to test different aspects of the code.

Modify the python files in the test folder to add or modify other test cases.

Prepare test cases considering what objects are created into the database for each test case.