

## IT314 - Software Engineering

### **WeatherWise – A Weather Prediction Site**



Group No. 22

**Unit Testing File**

## Unit Testing using **AAA Method**:



*image source: medium.com*

The Arrange, Act, Assert (AAA) pattern is a common way to write unit tests. It's a best practice that helps organize tests by breaking them down into three steps:

- **Arrange:** Set up the test by initializing objects and preparing the data and prerequisites.
- **Act:** Perform the actual work of the test.
- **Assert:** Verify that the result is what was expected.

## GitHub Link of code files(.py):

[Source code of Unit Testing using \(Pytest\)](#)

## Framework Used: Pytest

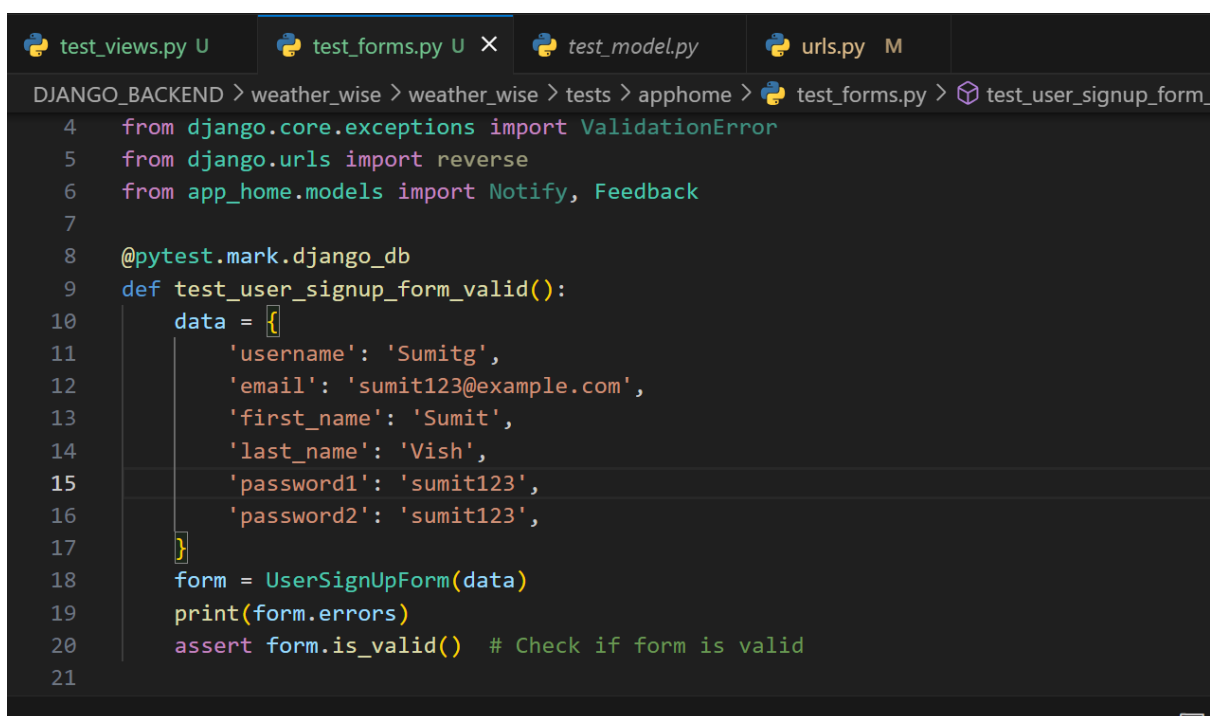
We used Pytest Unit Testing Framework to make unit tests and also used python Django's library "unittest" wherever required.

Pytest is a testing framework for Python that makes it easy to write simple and scalable test cases. It offers features like fixtures, parameterized tests, and detailed error reporting.

With Pytest, we were able to efficiently test our code and ensure its reliability.

### 1) User Signup Testing

#### Case 1: password is too similar to username



```
test_views.py U test_forms.py U X test_model.py urls.py M
DJANGO_BACKEND > weather_wise > weather_wise > tests > apphome > test_forms.py > test_user_signup_form_
4 from django.core.exceptions import ValidationError
5 from django.urls import reverse
6 from app_home.models import Notify, Feedback
7
8 @pytest.mark.django_db
9 def test_user_signup_form_valid():
10     data = {
11         'username': 'Sumitg',
12         'email': 'sumit123@example.com',
13         'first_name': 'Sumit',
14         'last_name': 'Vish',
15         'password1': 'sumit123',
16         'password2': 'sumit123',
17     }
18     form = UserSignUpForm(data)
19     print(form.errors)
20     assert form.is_valid() # Check if form is valid
21
```

**Verdict:** Test Failed as expected (User denied sign in)

```
===== short test summary info =====
FAILED weather_wise/tests/apphome/test_forms.py::test_user_signup_form_valid - assert False
===== 1 failed, 21 passed, 3 xfailed in 9.59s =====
```

```
@pytest.mark.django_db
def test_user_signup_form_valid():
    data = {
        'username': 'Sumitg',
        'email': 'sumit123@example.com',
        'first_name': 'Sumit',
        'last_name': 'Vish',
        'password1': 'sumit123',
        'password2': 'sumit123',
    }
    form = UserSignUpForm(data)
    print(form.errors)
    > assert form.is_valid() # Check if form is valid
E   assert False
E   + where False = is_valid()
E   +   where is_valid = <UserSignUpForm bound=True, valid=False, fields=(username;email;first_name;last_name;password1;password2)>.is_valid

weather_wise\tests\apphome\test_forms.py:20: AssertionError
----- Captured stdout call -----
<ul class="errorlist"><li>password2<ul class="errorlist"><li>The password is too similar to the username.</li></ul></li></ul>
```

**Case 2:** Valid credentials

```
test_views.py U test_forms.py U X test_model.py urls.py M
DJANGO_BACKEND > weather_wise > weather_wise > tests > apphome > test_forms.py >
4   from django.core.exceptions import ValidationError
5   from django.urls import reverse
6   from app_home.models import Notify, Feedback
7
8   @pytest.mark.django_db
9   def test_user_signup_form_valid():
10      data = {
11          'username': 'Sumitg',
12          'email': 'sumit123@example.com',
13          'first_name': 'Sumit',
14          'last_name': 'Vish',
15          'password1': 'madhav123',
16          'password2': 'madhav123',
17      }
18      form = UserSignUpForm(data)
19      print(form.errors)
20      assert form.is_valid() # Check if form is valid
21
```

**Verdict:** Test Passed (User is successfully logged in)

```
PS C:\Users\Work\Desktop\IT314_Project_G22\DJANGO_BACKEND\weather_wise> pytest --cov
===== test session starts =====
platform win32 -- Python 3.13.0b4, pytest-8.3.3, pluggy-1.5.0
django: version: 5.1.1, settings: weather_wise.settings (from ini)
rootdir: C:\Users\Work\Desktop\IT314_Project_G22\DJANGO_BACKEND\weather_wise
configfile: pytest.ini
plugins: cov-6.0.0, django-4.9.0
collected 25 items

weather_wise\tests\appphome\test_forms.py .....
weather_wise\tests\appphome\test_model.py .....X.....XX..

===== 22 passed, 3 xfailed in 9.10s =====
PS C:\Users\Work\Desktop\IT314_Project_G22\DJANGO_BACKEND\weather_wise>
```

File	Lines	Errors	Coverage
weather_wise\tests\appphome\test_forms.py	62	0	100%
weather_wise\tests\appphome\test_model.py	108	4	96%
weather_wise\tests\appphome\test_views.py	0	0	100%
<b>TOTAL</b>	<b>332</b>	<b>6</b>	<b>98%</b>

### Case 3: Duplicate Username

```
test_views.py U test_forms.py U X test_model.py urls.py M
DJANGO_BACKEND > weather_wise > weather_wise > tests > apphome > test_forms.py > test_user_signup_form_invalid_username
21
22 @pytest.mark.django_db
23 def test_user_signup_form_invalid_username():
24     # to test a user for duplicate username validation
25     User.objects.create_user(username='Sumitg', email='sumit2@example.com', password='shravan12345')
26
27     data = {
28         'username': 'Sumitg', # This is the username already in use
29         'email': 'sumit2@example.com',
30         'first_name': 'New',
31         'last_name': 'Sumit',
32         'password1': 'shravan12345',
33         'password2': 'shravan12345',
34     }
35     form = UserSignUpForm(data)
36     assert form.is_valid() # Form should not be valid
37     assert 'This username is already taken. Please choose another one.' in form.errors['username']
38
```

**Verdict:** Test failed Assertion error (This username is already taken. Please choose another one)

```
weather_wise\tests\appphome\test_forms.py:37: AssertionError
----- Captured stdout call -----
<ul class="errorlist"><li>username<ul class="errorlist"><li>This username is already taken. Please choose another one.</li></ul></li><li>email<ul class="errorlist"><li>This email address is already registered. Please log in instead.</li></ul></li></ul>
```

### Case 4: Duplicate Email id

```

@pytest.mark.django_db
def test_user_signup_form_invalid_email():
    # Create a user for email validation
    User.objects.create_user(username='testuser2', email='testuser@example.com', password='password123')

    data = {
        'username': 'newuser',
        'email': 'testuser@example.com', # Duplicate email
        'first_name': 'New',
        'last_name': 'User',
        'password1': 'newpassword123',
        'password2': 'newpassword123',
    }
    form = UserSignUpForm(data)
    assert form.is_valid() # Form should not be valid
    assert 'This email address is already registered. Please log in instead.' in form.errors['email']

```

**Verdict:** Test Failed Assertion Error ('This email address is already registered. Please log in instead.')

```

===== short test summary info =====
FAILED weather_wise/tests/apphome/test_forms.py::test_user_signup_form_invalid_email - assert False
===== 1 failed, 22 passed, 3 xfailed in 9.03s =====

```

## 2) Update Profile feature Testing

### Case 1: changing email address

```

test_views.py U  test_forms.py X  test_model.py  urls.py M
DJANGO_BACKEND > weather_wise > weather_wise > tests > apphome > test_forms.py > test_user_profile_edit_form_valid
59 #to test update user profile
60 @pytest.mark.django_db
61 def test_user_profile_edit_form_valid():
62     user = User.objects.create_user(username='Sumit3', email='sumit345@example.com', password='shravan333')
63     data = {
64         'username': 'Sumit3',
65         'email': 'sumit345example.com', # intentionally entered invalid email format
66         'first_name': 'Test',
67         'last_name': 'ing',
68     }
69     form = UserProfileEditForm(data, instance=user)
70     print(form.errors)
71     assert form.is_valid()

```

**Verdict:** Assertion Error (Enter Valid email address)

```

weather_wise\tests\apphome\test_forms.py:71: AssertionError
----- Captured stdout call -----
<ul class="errorlist"><li>email<ul class="errorlist"><li>Enter a valid email address.</li></ul></li></ul>

```

### Case2: Valid Email address update

```
test_views.py U test_forms.py U X test_model.py urls.py M
DJANGO_BACKEND > weather_wise > tests > apphome > test_forms.py > ...
73 #to test update user profile
74 @pytest.mark.django_db
75 def test_user_profile_edit_form_valid():
76     user = User.objects.create_user(username='Sumit3', email='sumit345@example.com', password='shravan333')
77     data = {
78         'username': 'Sumit3',
79         'email': 'sumit345@example.com', # entered valid email format
80         'first_name': 'Test',
81         'last_name': 'ing',
82     }
83     form = UserProfileEditForm(data, instance=user)
84     print(form.errors)
85     assert form.is_valid()
86
```

**Verdict: Test Passed**

```
PROBLEMS OUTPUT DEBUG CONSOLE PORTS TERMINAL COMMENTS powershell - weather_wise + v [] ... ^ x
weather_wise\tests\apphome\__init__.py 0 0 100%
weather_wise\tests\apphome\test_forms.py 71 0 100%
weather_wise\tests\apphome\test_model.py 108 4 96%
weather_wise\tests\apphome\test_views.py 0 0 100%
-----
TOTAL 341 6 98%

===== 23 passed, 3 xfailed in 11.87s =====
PS C:\Users\Work\Desktop\IT314_Project_G22\DJANGO_BACKEND\weather_wise>
master* 0 0 0 Watch Ln 85, Col 29 Spaces: 4 UTF-8 CRLF {} Python 3.13.0b4 64-bit Go Live
```

### 3) Notification Testing

**Case 1:** Notification field Trsue and preferred location not empty

```
86
87 # to test notify form
88 @pytest.mark.django_db
89 def test_notify_form_valid():
90     data = {
91         'get_notifications': True,
92         'preferred_location': 'New York',
93     }
94     form = NotifyForm(data)
95     assert form.is_valid()
96
```

**Verdict: Passed**

```
weather_wise\tests\apphome\__init__.py 0 0 100%
weather_wise\tests\apphome\test_forms.py 71 0 100%
weather_wise\tests\apphome\test_model.py 108 4 96%
weather_wise\tests\apphome\test_views.py 0 0 100%
-----
TOTAL 341 6 98%

===== 23 passed, 3 xfailed in 11.87s =====
PS C:\Users\Work\Desktop\IT314_Project_G22\DJANGO_BACKEND\weather_wise>
master* 0 0 0 Watch Ln 104, Col 59 Spaces: 4 UTF-8 CRLF {} Python 3.13.0b4 64-bit Go Live
```

## Case2: Notification field True and preferred location not empty

```
96
97 @pytest.mark.django_db
98 def test_notify_form_invalid_notifications():
99     data = {
100         'get_notifications': False,      # but receive notification field is False
101         'preferred_location': 'Ahmedabad', #non empty field
102     }
103     form = NotifyForm(data)
104     assert not form.is_valid() # Form should not be valid
105     assert 'You need to enable notifications to have a preferred location.' in form.errors
106
107
108 @pytest.mark.django_db
```

**Verdict:** assert not form.is\_valid() is True (Test Passed).

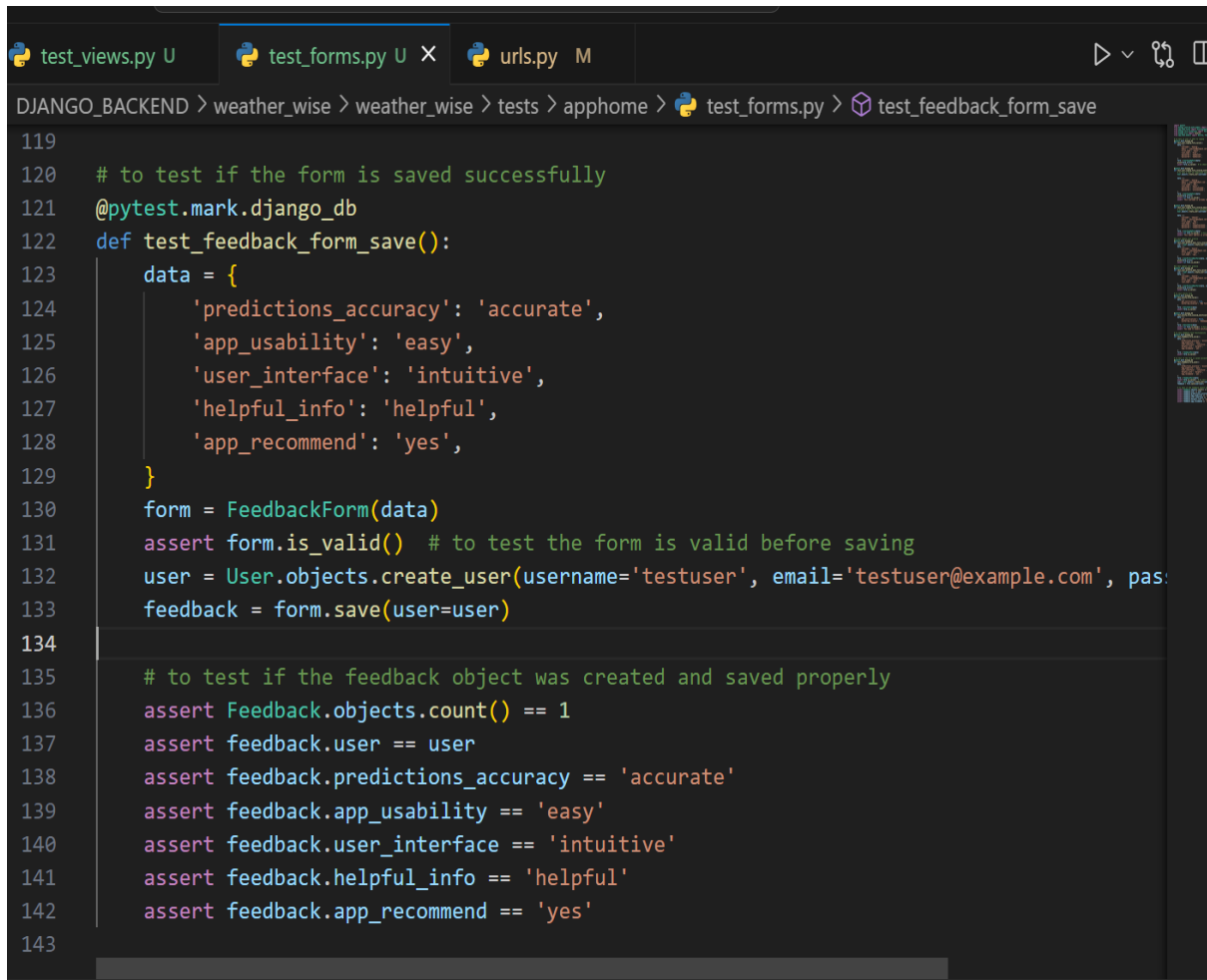
Means the testcase works fine.

weather_wise\tests\apphome\test_forms.py	71	0	100%
weather_wise\tests\apphome\test_model.py	108	4	96%
weather_wise\tests\apphome\test_views.py	0	0	100%
-----			
TOTAL	341	6	98%



## 4) Feedback form testing

**Case1:** Checking basic functionality by entering valid checkbox entries.



```
119
120 # to test if the form is saved successfully
121 @pytest.mark.django_db
122 def test_feedback_form_save():
123     data = {
124         'predictions_accuracy': 'accurate',
125         'app_usability': 'easy',
126         'user_interface': 'intuitive',
127         'helpful_info': 'helpful',
128         'app_recommend': 'yes',
129     }
130     form = FeedbackForm(data)
131     assert form.is_valid() # to test the form is valid before saving
132     user = User.objects.create_user(username='testuser', email='testuser@example.com', pas
133     feedback = form.save(user=user)
134
135     # to test if the feedback object was created and saved properly
136     assert Feedback.objects.count() == 1
137     assert feedback.user == user
138     assert feedback.predictions_accuracy == 'accurate'
139     assert feedback.app_usability == 'easy'
140     assert feedback.user_interface == 'intuitive'
141     assert feedback.helpful_info == 'helpful'
142     assert feedback.app_recommend == 'yes'
143
```

**Verdict:** Passed

weather_wise\tests\apphome\test_forms.py	71	0	100%
weather_wise\tests\apphome\test_model.py	108	4	96%
weather_wise\tests\apphome\test_views.py	0	0	100%
-----			
TOTAL	341	6	98%

```
===== 23 passed, 3 xfailed in 11.06s =====
PS C:\Users\Work\Desktop\IT314_Project_G22\DJANGO_BACKEND\weather_wise>
```

## 5) Recent Location model testing

After completing the unit testing of all models, in the coverage report, one function's coverage was detected to be missing:

```
Coverage for app_home\models.py: 97%
31 statements  30 run  1 missing  0 excluded
« prev  ^ index  » next  coverage.py v7.6.7, created at 2024-11-18 13:51 +0530

1 from django.db import models
2 from django.contrib.auth.models import User
3
4 class Notify(models.Model):
5     user = models.OneToOneField(User, on_delete=models.CASCADE)
6     preferred_location = models.CharField(max_length=100)
7
8     def __str__(self):
9         return f"Notify preferences for {self.user.username}"
10
11 class Fav_loc(models.Model):
12     user = models.ForeignKey(User, on_delete=models.CASCADE) # One user can have many favorite Locations
13     favourite_location = models.CharField(max_length=100)
14
15     def __str__(self):
16         return f"{self.user.username} saves {self.favourite_location}"
17
18 class Recent_loc(models.Model):
19     user = models.ForeignKey(User, on_delete=models.CASCADE) # One user can have many favorite Locations
20     recent_location = models.CharField(max_length=100)
21
22     def __str__(self):
23         return f"{self.user.username} saves {self.recent_location}"
24
25 class Feedback(models.Model):
26     user = models.ForeignKey(User, on_delete=models.CASCADE, null=True, blank=True) # Optional user registration
```

Thus, created two tests for it to **ensure 100% code coverage**

**Case1:** To test the return value

```
183 #Below is the testcase to test the recent location model return string
184 @pytest.mark.django_db
185 def test_recent_loc_str(user):
186     # to test the __str__ method of Fav_loc model
187     rec_loc = Recent_loc.objects.create(user=user, recent_location="Jamnagar")
188     assert str(rec_loc) == f"{rec_loc.user.username} saves {rec_loc.favourite_location}"
189
190 @pytest.mark.xfail("This should fail if the recent location string >100 characters")
191 @pytest.mark.django_db
192 def test_recent_loc_string_length(user):
193     # to test the __str__ method of Fav_loc model
194     rec_loc = Recent_loc.objects.create(user=user, recent_location="Varanasi"*101)
195     assert str(rec_loc) == f"{rec_loc.user.username} saves {rec_loc.recent_location}"
196     ⚡ if len(rec_loc.recent_location) > 100:
197         raise ValidationError("Favourite location string cant exceed 100 chars!")
198
```

**Verdict:** Passed

**Case 2:** Recent location string should not exceed 100chars

```
@pytest.mark.xfail("This should fail if the recent location string >100 characters")
@pytest.mark.django_db
def test_recent_loc_string_length(user):
    # to test the __str__ method of Fav_loc model
    rec_loc = Recent_loc.objects.create(user=user, recent_location="Varanasi"*101)
    assert str(rec_loc) == f"{rec_loc.user.username} saves {rec_loc.recent_location}"
    if len(rec_loc.recent_location) > 100:
        raise ValidationError("Favourite location string cant exceed 100 chars!")
```

**Verdict:** xFail (expected failure) means the test is passed and It doesn't allows system to enter string of more than 100 characters.

```
weather_wise\tests\appphome\test_model.py .....X.....XX...X [100%]

----- coverage: platform win32, python 3.13.0-beta-4 -----
Coverage HTML written to dir htmlcov

===== 24 passed, 4 xfailed in 10.51s =====
PS C:\Users\Work\Desktop\IT314_Project_G22\DJANGO_BACKEND\weather_wise> 
```

## 6) Testing URLs

client: The client is a Django test client that allows to simulate HTTP requests (like GET and POST) to our Django application during unit testing. It helps us test views and their responses without running a server.

```
test_model.py M test_urls.py U X models.py
DJANGO_BACKEND > weather_wise > weather_wise > tests > apphome > test_urls.py > test_urls

1 import pytest
2 from django.contrib.auth.models import User
3
4 @pytest.mark.django_db
5 @pytest.mark.parametrize("path,view_name,requires_auth,expected_status", [
6     ('/', 'home_view', False, 200),
7     ('/about/', 'about_view', False, 200),
8     ('/login/', 'login_view', False, 200),
9     ('/dashboard/', 'dashboard_view', True, 200), # we need user to be logged in for this page
10    ('/logout/', 'logout_view', True, 302), # here expected_status=302 -> because of redirect to home page after
11    ('/signup/', 'signup_view', False, 200),
12    ('/predict/', 'predict_view', True, 200),
13    ('/profile/', 'profile_view', True, 200),
14    ('/profile/edit/', 'profile_edit_view', True, 200),
15    ('/feedback/', 'feedback_view', True, 200), # similar here, we need user to be logged in for this page
16    ('/switch-theme/', 'change-theme', False, 302),
```

Parametrize helps to run the same test function multiple times with different sets of input data (here, we test different URLs with the same function instead of creating multiple testcases which would be a hefty task).

**Verdict:** All urls work fine. (All tests passed)

```
plugins: cov-6.0.0, django-4.9.0
collected 39 items

weather_wise\tests\apphome\test_forms.py ..... [ 23%]
weather_wise\tests\apphome\test_model.py .....X.....XX...X [ 71%]
weather_wise\tests\apphome\test_urls.py .....

----- coverage: platform win32, python 3.13.0-beta-4 -----
Coverage HTML written to dir htmlcov

===== 35 passed, 4 xfailed in 25.46s =====
PS C:\Users\Work\Desktop\IT314_Project_G22\DJANGO_BACKEND\weather_wise> 
```

### Coverage for `app_home\urls.py`: 100%

4 statements

4 run

0 missing

0 excluded

[« prev](#)   [^ index](#)   [» next](#)   *coverage.py v7.6.7, created at 2024-11-18 14:36 +0530*

```
1 from django.urls import path
2 from app_home import views
3 from theme.views import change_theme
4
5 urlpatterns = [
6     path('', views.home_view, name="home_view"),
7     path('about/', views.about_view, name="about_view"),
8     path('login/', views.login_view, name="login_view"),
9     path('dashboard/', views.dashboard_view, name="dashboard_view"),
10    path('logout/', views.logout_view, name="logout_view"),
11    path('signup/', views.signup_view, name="signup_view"),
12    path('predict/', views.predict_view, name="predict_view"),
13    path('profile/', views.profile_view, name="profile_view"),
14    path('profile/edit/', views.profile_edit_view, name='profile_edit_view'),
15    path('feedback/', views.feedback_view, name='feedback_view'),
16    path('switch-theme/', change_theme, name='change-theme'),
17 ]
```

[« prev](#)   [^ index](#)   [» next](#)

*coverage.py v7.6.7, created at 2024-11-18 14:36 +0530*

## 7) Views Testing

**Case1 :** testing HTTP GET request to the home\_view view function.

```
6
7 # this is to test the get request of home view
8 @pytest.mark.django_db
9 def test_home_view_get(client):
10
11     #this test checks the return of default weather data for predefined cities (as metioned in the views.py file)
12
13     url = reverse('home_view')
14     response = client.get(url)
15     assert response.status_code == 200
16     assert 'data_Delhi' in response.context #this verifies that the home_view view passes the weather data to
17     assert 'data_Mumbai' in response.context
18     assert 'data_Hyderabad' in response.context
19
```

**Case 2:** testing that the homeview correctly handels a POST request for city

```
20
21 @pytest.mark.django_db
22 def test_post_city(client):
23
24     # this is to test that the homeview correctly handels a POST request for city
25     url = reverse('home_view')
26     response = client.post(url, {'location': 'Gandhinagar'}) # simulating a post request
27     assert response.status_code == 200
28     assert 'data' in response.context # checks if the data is passed form the template successfully
29
```

**Case 3:** to test and ensures that the dashboard\_view works correctly when an **authenticated user** accesses it

```
@pytest.mark.django_db
def test_dashboard_view_authenticated_user(client, django_user_model):

    #this is to check the dashboard view for the logged in user
    user = django_user_model.objects.create_user(username='Madhav', password='Kakadiya')
    client.login(username='Madhav', password='Kakadiya')
    url = reverse('dashboard_view')
    response = client.get(url)
    assert response.status_code == 200
    assert 'fav_locs_data' in response.context # checks if the fav_locs data is passed successfully
```

**Case 4:** to test that logged in user adding a favorite location in the dashboard view

```

42
43 @pytest.mark.django_db
44 def test_dashboard_add_favorite(client, django_user_model):
45
46     # this is to check logged in user adding a favorite location in the dashboard view
47     user = django_user_model.objects.create_user(username='Shravan', password='Vishwakarma')
48     client.login(username='Shravan', password='Vishwakarma')
49
50     url = reverse('dashboard_view')
51     response = client.post(url, {'fav_location_save': 'Ahmedabad'}) # creating an instance of fav_location
52     assert response.status_code == 200 # checks if there is no redirect to other page
53     assert Fav_loc.objects.filter(user=user, favourite_location='Ahmedabad').exists() # to check that the fav loc
54

```

## Case 5: Test the login view responds correctly to GET request

```

55
56 @pytest.mark.django_db
57 def test_login_view_get(client):
58     # to test the login view with get request
59     url = reverse('login_view')
60     response = client.get(url)
61     assert response.status_code == 200
62
63

```

## Case 6: to test the login view POST request for authenticated user

```

64
65 @pytest.mark.django_db
66 def test_login_view_post(client, django_user_model):
67     # to test the login view POST request for authenticated user
68     user = django_user_model.objects.create_user(username='Sharvil', password='Oza')
69     url = reverse('login_view') # generates url for login
70     response = client.post(url, {'username_or_email': 'Sharvil', 'password': 'Oza'}) # simulating a POST request
71     assert response.status_code == 302 # checks redirect to dashboard

```

## Case 7: to test that signup view responds correctly to a GET request.

```

@pytest.mark.django_db
def test_signup_view_get(client):
    # to check that signup view responds correctly to a GET request.

    url = reverse('signup_view')
    response = client.get(url) # GET request from signup view
    assert response.status_code == 200
    assert 'form' in response.context # to confirm that the view passes a form object

```

**Case 8:** to test that the signup\_view correctly handles a POST request.

```

82
83 @pytest.mark.django_db
84 def test_signup_view_post_invalid(client):
85     # to verify that the signup_view correctly handles a POST request with valid data.
86     url = reverse('signup_view') # to fetch the url for signup_view
87     response = client.post(url, {
88         'username': 'SaurabhSir',
89         'password1': 'IT314',
90         'password2': 'wrongpass'
91     })
92     # due to wrong password it will stay on the same page (signup)
93     assert response.status_code==200
94     assert not User.objects.filter(username='SaurabhSir').exists() # to check if the new account is created or n
95

```

**Case 9:** to test that predict view responds correctly to a GET request.

```

@pytest.mark.django_db
def test_predict_view_user(client, django_user_model):

    # to test predict view for a logged in user
    user = django_user_model.objects.create_user(username='Nisarg', password='Modi')
    client.login(username='Nisarg', password='Modi')
    url = reverse('predict_view')
    response = client.get(url)
    assert response.status_code == 200
    assert 'recentLocs' in response.context # test if recent_Locs is passed to the template

```

**Case 10:** to test profile view responds correctly to GET request for logged in user

```

@pytest.mark.django_db
def test_profile_view_user(client, django_user_model):
    # to test profile view for a logged in user
    user = django_user_model.objects.create_user(username='Bhavya', password='Shah')
    client.login(username='Bhavya', password='Shah')
    url = reverse('profile_view')
    response = client.get(url) # simulate get request
    assert response.status_code == 200

```

**Case 11:** to test profile edit view responds correctly to GET request for a logged in user

```

118 @pytest.mark.django_db
119 def test_profile_edit_view_user(client, django_user_model):
120     # to test profileedit view for a logged in user
121     user = django_user_model.objects.create_user(username='Vraj', password='K')
122     client.login(username='Vraj', password='K')
123     url = reverse('profile_edit_view')
124     response = client.get(url)
125     assert response.status_code == 200
126     assert 'profile_form' in response.context
127     assert 'notify_form' in response.context
128

```

**Case 12:** to test feedback view responds correctly to GET request for a logged in user

```

129
130 @pytest.mark.django_db
131 def test_feedback_view_user(client, django_user_model):
132     # to test the feedback view for a logged in user
133     user = django_user_model.objects.create_user(username='noMoreUserename', password='nopassword')
134     client.login(username='noMoreUsername', password='nopassword')
135     url = reverse('feedback_view')
136     response = client.get(url) #simulate a GET request
137     assert response.status_code == 200
138     assert 'form' in response.context
139

```



# Report:

# Terminal:

```
plugins: cov-6.0.0, django-4.9.0
PS C:\Users\Work\Desktop\IT314_Project_G22\DJANGO_BACKEND\weather_wise> pytest --cov=app_home --cov-report=html
===== test session starts =====
platform win32 -- Python 3.13.0b4, pytest-8.3.3, pluggy-1.5.0
django: version: 5.1.1, settings: weather_wise.settings (from ini)
rootdir: C:\Users\Work\Desktop\IT314_Project_G22\DJANGO_BACKEND\weather_wise
configfile: pytest.ini
plugins: cov-6.0.0, django-4.9.0
collected 62 items

weather_wise\tests\apphome\test_forms.py ..... [ 14%]
weather_wise\tests\apphome\test_model.py .....x.....xx...x [ 45%]
weather_wise\tests\apphome\test_urls.py ..... [ 62%]
weather_wise\tests\apphome\test_views.py ..... [100%]

----- coverage: platform win32, python 3.13.0-beta-4 -----
Coverage HTML written to dir htmlcov

===== 58 passed, 4 xfailed in 56.24s =====
PS C:\Users\Work\Desktop\IT314_Project_G22\DJANGO_BACKEND\weather_wise> █
```

# Code Coverage: 97%

Coverage report: 97%

filter...hide coverage

FilesFunctionsClasses

coverage.py v7.6.7, created at 2024-11-21 13:12 +0530

File	function	statements	missing	excluded	coverage
weather_wise\__init__.py	(no function)	0	0	0	100%
weather_wise\asgi.py	(no function)	4	4	0	0%
weather_wise\settings.py	(no function)	34	0	0	100%
weather_wise\tests\apphome\__init__.py	(no function)	0	0	0	100%
weather_wise\tests\apphome\test_forms.py	test_user_signup_form_valid	4	0	0	100%
weather_wise\tests\apphome\test_forms.py	test_user_signup_form_invalid_username	6	0	0	100%
weather_wise\tests\apphome\test_forms.py	test_user_signup_form_invalid_email	5	0	0	100%
weather_wise\tests\apphome\test_forms.py	test_user_profile_edit_form_invalid	5	0	0	100%
weather_wise\tests\apphome\test_forms.py	test_user_profile_edit_form_valid	5	0	0	100%
weather_wise\tests\apphome\test_forms.py	test_notify_form_valid	3	0	0	100%
weather_wise\tests\apphome\test_forms.py	test_notify_form_invalid_notifications	4	0	0	100%
weather_wise\tests\apphome\test_forms.py	test_feedback_form_valid	3	0	0	100%
weather_wise\tests\apphome\test_forms.py	test_feedback_form_save	12	0	0	100%
weather_wise\tests\apphome\test_forms.py	(no function)	23	0	0	100%
weather_wise\tests\apphome\test_model.py	user	1	1	0	0%
weather_wise\tests\apphome\test_model.py	feedback	1	0	0	100%

weather_wise\tests\apphome\test_views.py	test_home_view_get	6	0	0	100%
weather_wise\tests\apphome\test_views.py	test_home_view_post	5	0	0	100%
weather_wise\tests\apphome\test_views.py	test_home_view_post_with_error	5	0	0	100%
weather_wise\tests\apphome\test_views.py	test_post_city	4	0	0	100%
weather_wise\tests\apphome\test_views.py	test_dashboard_view_authenticated_user	6	0	0	100%
weather_wise\tests\apphome\test_views.py	test_dashboard_add_favorite	6	0	0	100%
weather_wise\tests\apphome\test_views.py	test_dashboard_no_favorites	7	0	0	100%
weather_wise\tests\apphome\test_views.py	test_dashboard_delete_favorite	9	0	0	100%
weather_wise\tests\apphome\test_views.py	test_dashboard_search_location	7	0	0	100%
weather_wise\tests\apphome\test_views.py	test_dashboard_search_invalid_location	8	0	0	100%
weather_wise\tests\apphome\test_views.py	test_dashboard_multiple_favorites	9	0	0	100%
weather_wise\tests\apphome\test_views.py	test_dashboard_view_with_fav_locations	11	0	0	100%
weather_wise\tests\apphome\test_views.py	test_login_view_get	3	0	0	100%
weather_wise\tests\apphome\test_views.py	test_login_view_logout_redirect	7	0	0	100%
weather_wise\tests\apphome\test_views.py	test_login_view_post	4	0	0	100%
weather_wise\tests\apphome\test_views.py	test_login_view_post_invalid_user	5	0	0	100%
weather_wise\tests\apphome\test_views.py	test_signup_view_get	4	0	0	100%
weather_wise\tests\apphome\test_views.py	test_signup_view_post_invalid	7	0	0	100%
weather_wise\tests\apphome\test_views.py	test_predict_view_post	9	4	0	56%
weather_wise\tests\apphome\test_views.py	test_profile_view_user	5	0	0	100%
weather_wise\tests\apphome\test_views.py	test_profile_edit_view_post_update_notifications	10	0	0	100%

Coverage report: 97%

Files Functions Classes

coverage.py v7.6.7, created at 2024-11-21 13:15 +0530

File ▲	statements	missing	excluded	coverage
weather_wise\__init__.py	0	0	0	100%
weather_wise\asgi.py	4	4	0	0%
weather_wise\settings.py	34	0	0	100%
weather_wise\tests\apphome\__init__.py	0	0	0	100%
weather_wise\tests\apphome\test_forms.py	70	0	0	100%
weather_wise\tests\apphome\test_model.py	119	4	0	97%
weather_wise\tests\apphome\test_urls.py	10	0	0	100%
weather_wise\tests\apphome\test_views.py	207	4	0	98%
weather_wise\tests\apphome\test_wsgi.py	4	0	0	100%
weather_wise\urls.py	3	0	0	100%
weather_wise\wsgi.py	4	0	0	100%
<b>Total</b>	<b>455</b>	<b>12</b>	<b>0</b>	<b>97%</b>

coverage.py v7.6.7, created at 2024-11-21 13:15 +0530