IT314 - Software Engineering

WeatherWise – A Weather Prediction Site

Group No. 22

Unit Testing File

Unit Testing using **AAA Method**:

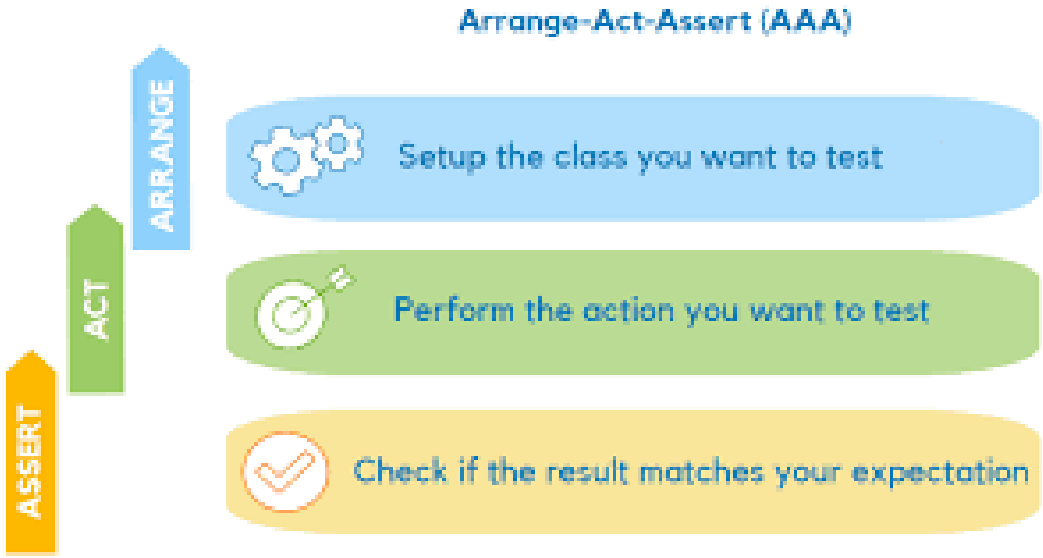

image source: medium.com

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 | test_model.py | urls.py M

DJANGO_BACKEND > weather_wise > weather_wise > tests > apphome > test_forms.py > test_user_signup_form_

```python
from django.core.exceptions import ValidationError
from django.urls import reverse
from app_home.models import Notify, Feedback

@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

```

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 ======
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   PORTS   TERMINAL   COMMENTS                    powershell - weather_wise

    @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>

master*  ⊗ 0 ⚠ 0   0   Watch   Ln 20, Col 29   Spaces: 4   UTF-8   CRLF   {} Python   3.13.0b4 64-bit   Go Live
```

Case 2: Valid credentials

```
test_views.py U   test_forms.py U ×   test_model.py   urls.py M
DJANGO_BACKEND > weather_wise > weather_wise > tests > apphome > test_forms.py >

from django.core.exceptions import ValidationError
from django.urls import reverse
from app_home.models import Notify, Feedback

@pytest.mark.django_db
def test_user_signup_form_valid():
    data = {
        'username': 'Sumitg',
        'email': 'sumit123@example.com',
        'first_name': 'Sumit',
        'last_name': 'Vish',
        'password1': 'madhav123',
        'password2': 'madhav123',
    }
    form = UserSignUpForm(data)
    print(form.errors)
    assert form.is_valid()  # Check if form is valid
```

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\apphome\test_forms.py ........
weather_wise\tests\apphome\test_model.py ......x......xx..
```

```
weather_wise\tests\apphome\test_forms.py                          62      0   100%
weather_wise\tests\apphome\test_model.py                         108      4    96%
weather_wise\tests\apphome\test_views.py                           0      0   100%
-------------------------------------------------------------------------------------
TOTAL                                                            332      6    98%


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

Case 3: Duplicate Username

test_views.py U | test_forms.py U | test_model.py | urls.py M

DJANGO_BACKEND > weather_wise > weather_wise > tests > apphome > test_forms.py > test_user_signup_form_invalid_username

```python

@pytest.mark.django_db
def test_user_signup_form_invalid_username():
    # to test a user for duplicate username validation
    User.objects.create_user(username='Sumitg', email='sumit2@example.com', password='shravan12345')

    data = {
        'username': 'Sumitg',  # This is the username already in use
        'email': 'sumit2@example.com',
        'first_name': 'New',
        'last_name': 'Sumit',
        'password1': 'shravan12345',
        'password2': 'shravan12345',
    }
    form = UserSignUpForm(data)
    assert form.is_valid()  # Form should not be valid
    assert 'This username is already taken. Please choose another one.' in form.errors['username']
```

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

```
weather_wise\tests\apphome\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

```python
@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 U | test_model.py | urls.py M

DJANGO_BACKEND > weather_wise > weather_wise > tests > apphome > test_forms.py > test_user_profile_edit_form_valid

```python
#to test update user profile
@pytest.mark.django_db
def test_user_profile_edit_form_valid():
    user = User.objects.create_user(username='Sumit3', email='sumit345@example.com', password='shravan333')
    data = {
        'username': 'Sumit3',
        'email': 'sumit345example.com',   # intentionally entered invalid email format
        'first_name': 'Test',
        'last_name': 'ing',
    }
    form = UserProfileEditForm(data, instance=user)
    print(form.errors)
    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 ×   test_model.py   urls.py M
DJANGO_BACKEND > weather_wise > weather_wise > tests > apphome > test_forms.py > ...
#to test update user profile
@pytest.mark.django_db
def test_user_profile_edit_form_valid():
    user = User.objects.create_user(username='Sumit3', email='sumit345@example.com', password='shravan333')
    data = {
        'username': 'Sumit3',
        'email': 'sumit345@example.com',   #  entered valid email format
        'first_name': 'Test',
        'last_name': 'ing',
    }
    form = UserProfileEditForm(data, instance=user)
    print(form.errors)
    assert form.is_valid()
```

Verdict: Test Passed

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   PORTS   TERMINAL   COMMENTS        powershell - weather_wise
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

```
# to test notify form
@pytest.mark.django_db
def test_notify_form_valid():
    data = {
        'get_notifications': True,
        'preferred_location': 'New York',
    }
    form = NotifyForm(data)
    assert form.is_valid()
```

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

```python

@pytest.mark.django_db
def test_notify_form_invalid_notifications():
    data = {
        'get_notifications': False,         # but receive notification field is False
        'preferred_location': 'Ahmedabad',  #non empty field
    }
    form = NotifyForm(data)
    assert not form.is_valid()  # Form should not be valid
    assert 'You need to enable notifications to have a preferred location.' in form.errors


@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.

test_views.py U | test_forms.py U | urls.py M

DJANGO_BACKEND > weather_wise > weather_wise > tests > apphome > test_forms.py > test_feedback_form_save

```python

# to test if the form is saved successfully
@pytest.mark.django_db
def test_feedback_form_save():
    data = {
        'predictions_accuracy': 'accurate',
        'app_usability': 'easy',
        'user_interface': 'intuitive',
        'helpful_info': 'helpful',
        'app_recommend': 'yes',
    }
    form = FeedbackForm(data)
    assert form.is_valid()  # to test the form is valid before saving
    user = User.objects.create_user(username='testuser', email='testuser@example.com', pas
    feedback = form.save(user=user)

    # to test if the feedback object was created and saved properly
    assert Feedback.objects.count() == 1
    assert feedback.user == user
    assert feedback.predictions_accuracy == 'accurate'
    assert feedback.app_usability == 'easy'
    assert feedback.user_interface == 'intuitive'
    assert feedback.helpful_info == 'helpful'
    assert feedback.app_recommend == 'yes'

```

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

```python
from django.db import models
from django.contrib.auth.models import User

class Notify(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    preferred_location = models.CharField(max_length=100)

    def __str__(self):
        return f"Notify preferences for {self.user.username}"

class Fav_loc(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)  # One user can have many favorite locations
    favourite_location = models.CharField(max_length=100)

    def __str__(self):
        return f"{self.user.username} saves {self.favourite_location}"

class Recent_loc(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)  # One user can have many favorite locations
    recent_location = models.CharField(max_length=100)

    def __str__(self):
        return f"{self.user.username} saves {self.recent_location}"

class Feedback(models.Model):
```

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

Case1: To test the return value

```python
#Below is the testcase to test the recent location model return string
@pytest.mark.django_db
def test_recent_loc_str(user):
    # to test the __str__ method of Fav_loc model
    rec_loc = Recent_loc.objects.create(user=user, recent_location="Jamnagar")
    assert str(rec_loc) == f"{rec_loc.user.username} saves {rec_loc.favourite_location}"

@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: Passed

Case 2: Recent location string should not exceed 100chars

```python
@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\apphome\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 | models.py

DJANGO_BACKEND > weather_wise > weather_wise > tests > apphome > test_urls.py > test_urls

```python
import pytest
from django.contrib.auth.models import User

@pytest.mark.django_db
@pytest.mark.parametrize("path,view_name,requires_auth,expected_status", [
    ('/', 'home_view', False, 200),
    ('/about/', 'about_view', False, 200),
    ('/login/', 'login_view', False, 200),
    ('/dashboard/', 'dashboard_view', True, 200), # we need user to be logged in for this page
    ('/logout/', 'logout_view', True, 302),  # here expected_status=302 -> because of redirect to home page after
    ('/signup/', 'signup_view', False, 200),
    ('/predict/', 'predict_view', True, 200),
    ('/profile/', 'profile_view', True, 200),
    ('/profile/edit/', 'profile_edit_view', True, 200),
    ('/feedback/', 'feedback_view', True, 200), # similar here, we need user to be logged in for this page
    ('/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.

```python
# this is to test the get request of home view
@pytest.mark.django_db
def test_home_view_get(client):

    #this test checks the return of default weather data for predefined cities (as metioned in the views.py file)

    url = reverse('home_view')
    response = client.get(url)
    assert response.status_code == 200
    assert 'data_Delhi' in response.context    #this verifies that the home_view view passes the weather data to
    assert 'data_Mumbai' in response.context
    assert 'data_Hyderabad' in response.context
```

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

```python
@pytest.mark.django_db
def test_post_city(client):

    # this is to test that the homeview correctly handels a POST request for city
    url = reverse('home_view')
    response = client.post(url, {'location': 'Gandhinagar'})  # simulating a post request
    assert response.status_code == 200
    assert 'data' in response.context  # checks if the data is passed form the template successfully
```

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

```python
@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

```python

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

    # this is to check logged in user adding a favorite location in the dashboard view
    user = django_user_model.objects.create_user(username='Shravan', password='Vishwakarma')
    client.login(username='Shravan', password='Vishwakarma')

    url = reverse('dashboard_view')
    response = client.post(url, {'fav_location_save': 'Ahmedabad'})  # creating an instance of fav_location
    assert response.status_code == 200  # checks if there is no redirect to other page
    assert Fav_loc.objects.filter(user=user, favourite_location='Ahmedabad').exists() # to check that the fav loc

```

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

```python

@pytest.mark.django_db
def test_login_view_get(client):
    # to test the login view with get request
    url = reverse('login_view')
    response = client.get(url)
    assert response.status_code == 200


```

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

```python

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

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

```python
@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.

```python

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

```

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

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

    # to test predict view for a loggied 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

```python
@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

```python
@pytest.mark.django_db
def test_profile_edit_view__user(client, django_user_model):
    # to test profileedit  view for a logged in user
    user = django_user_model.objects.create_user(username='Vraj', password='K')
    client.login(username='Vraj', password='K')
    url = reverse('profile_edit_view')
    response = client.get(url)
    assert response.status_code == 200
    assert 'profile_form' in response.context
    assert 'notify_form' in response.context
```

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

```python

@pytest.mark.django_db
def test_feedback_view__user(client, django_user_model):
    # to test the feedback view for a logged in user
    user = django_user_model.objects.create_user(username='noMoreUserename', password='nopassword')
    client.login(username='noMoreUsernamе', password='nopassword')
    url = reverse('feedback_view')
    response = client.get(url)  #simulate a GET request
    assert response.status_code == 200
    assert 'form' in response.context
```

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:

Coverage report: 92%

Files | Functions | Classes

coverage.py v7.6.7, created at 2024-11-18 18:56 +0530

File	statements	missing	excluded	coverage
app_home__init__.py	0	0	0	100%
app_home\admin.py	9	0	0	100%
app_home\apps.py	4	0	0	100%
app_home\forms.py	71	2	0	97%
app_home\migrations__init__.py	0	0	0	100%
app_home\migrations\0001_initial.py	4	0	0	100%
app_home\migrations\0002_initial.py	7	0	0	100%
app_home\migrations\0003_feedback.py	6	0	0	100%
app_home\migrations\0004_alter_feedback_app_recommend_and_more.py	6	0	0	100%
app_home\migrations\0005_alter_fav_loc_user.py	6	0	0	100%
app_home\migrations\0006_recent_loc.py	6	0	0	100%
app_home\migrations\0007_notify_get_notifications.py	4	0	0	100%
app_home\models.py	32	0	0	100%
app_home\urls.py	4	0	0	100%
app_home\views.py	195	26	0	87%
Total	**354**	**28**	**0**	**92%**

file:///C:/Users/Work/Desktop/IT314_Project_G22/DJANGO_BACKEND/weather_wise/htmlcov/z_3e8218a190445750_0005_alter_fav_loc_user_py.html