IT313 - Software Engineering

Warehouse Management System

Group No. 5

Unit Testing

Group Members

202101081 Janardan Pandya
202101054 Bhavya Dudhagara
202101075 Smit Rupapara
202101070 Jayesh Pandya
202101061 Manav Fitter
202101027 Akhil Kanteti
202101037 Kanishk Jain
202101074 Jenil Patel
202101055 Viren Goswami
202101046 Adit Parekh
202101065 Krish Vaghela

Under the guidance of

Professor: Dr. Saurabh Tiwari
Mentor: Shrut Shah

Pattern Used in Writing Test Cases (AAA Structure)

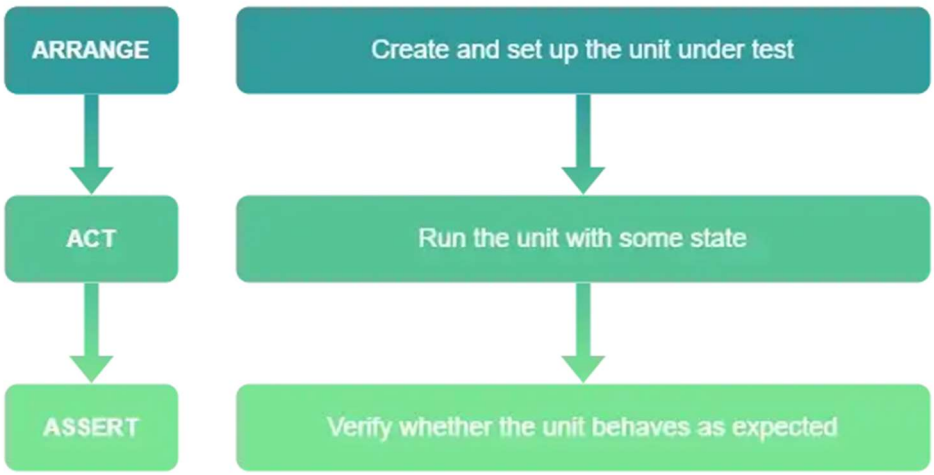
The utilization of the AAA pattern in writing test cases can aid in creating tests that are both easily comprehensible and maintainable, while simultaneously improving the ability to identify and detect errors and bugs.

Arrange: prepare and initialize

Act: invoke method testing

Assert: validate result expected

The **AAA (Arrange-Act-Assert) pattern** is a widely adopted method for composing test cases that facilitates the creation of tests with clear and logical structure. It involves a set of steps that include: arranging the necessary preconditions, performing the action to be tested, and verifying the expected result.



AAA pattern

1. **Arrange:** Prepare for the test by creating objects, configuring mock data, or executing other necessary setup tasks that establish the required preconditions.

2. **Act:** Execute the action that is being tested, which may involve calling a function, interacting with a widget, or performing another relevant action.

3. **Assert:** In the Assert step, it is necessary to validate that the action produced the intended outcome. This validation process may include verifying the return value of a function, assessing the state of an object, or confirming the truth of another relevant condition.

1. All Farmer Side Testing:

In this Module all tests for farmer like searching for Warehouse and edit profile for farmers is tested.

```
from django.test import TestCase, Client
from django.urls import reverse, resolve
from django.contrib.auth.models import User
from mainapp.models import Farmer,Warehouse_owner, Warehouse, Unit
from django.contrib.messages import get_messages

print("=> Jenil Testing: Edit Profile")
class TestFarmerEditProfile(TestCase):
    def setUp(self):
        self.client = Client()
        self.testemail = "test@gmail.com"
        self.testpassword = "testpassword"
        self.user = User.objects.create_user(self.testemail,self.testemail,
self.testpassword)
        self.farmer = Farmer.objects.create(email = self.testemail)
        self.client.login(username=self.testemail,password=self.testpassword)
        self.url = reverse('farmer_editprofile')

    def test_validcase(self):
        response = self.client.post(self.url, {
            'first_name': 'Jenil',
            'last_name':'Patel',
            'phone_no': 1234567890,
            'city':'Test city',
            'state': 'Test state',
        })
        print(response)
        # self.assertRedirects(response, reverse('farmer_profile'),
status_code=302)
        self.assertTrue(response.url == reverse('farmer_profile') and
response.status_code == 302)

    def test_invalid_empty_first_name(self):
        response = self.client.post(self.url, {
            'first_name': '',
            'last_name':'Patel',
            'phone_no': 1234567890,
            'city':'Test city',
            'state': 'Test state',
        })
        self.assertTrue(response.status_code == 200) # Because Form will
become invalid and if clause will pass

    def test_invalid_empty_last_name(self):
```

```
        response = self.client.post(self.url, {
            'first_name': 'Jenil',
            'last_name':'',
            'phone_no': 1234567890,
            'city':'Test city',
            'state': 'Test state',
        })
        self.assertTrue(response.status_code == 200) # Because Form will
become invalid and if clause will pass

    def test_invalid_shorter_phone_no(self):
        response = self.client.post(self.url, {
            'first_name': 'Jenil',
            'last_name': 'Patel',
            'phone_no': 123456,
            'city' : 'Test city',
            'state': 'Test state',
        })
        self.assertTrue(response.url == reverse('farmer_editprofile') and
response.status_code == 302)

    def test_invalid_longer_phone_no(self):
        response = self.client.post(self.url, {
            'first_name': 'Jenil',
            'last_name': 'Patel',
            'phone_no': 12345678901,
            'city':'Test city',
            'state': 'Test state',
        })
        self.assertTrue(response.status_code == 200) # Because Form will
become invalid and if clause will pass

    def test_invalid_empty_city(self):
        response = self.client.post(self.url, {
            'first_name': 'Jenil',
            'last_name':'Patel',
            'phone_no': 1234567890,
            'city':'',
            'state': 'Test state',
        })
        self.assertTrue(response.status_code == 200) # Because Form will
become invalid and if clause will pass

    def test_invalid_empty_state(self):
        response = self.client.post(self.url, {
            'first_name': 'Jenil',
            'last_name':'Patel',
            'phone_no': 1234567890,
```

```
            'city':'Test city',
            'state': '',
        })
        self.assertTrue(response.status_code == 200) # Because Form will
become invalid and if clause will pass

print("=> Jenil Testing: Farmer Searching Warehouse")
class TestWarehouseSearch(TestCase):
    def setUp(self):
        self.client = Client()
        self.testemail = "test@gmail.com"
        self.testpassword = "testpassword"
        self.user = User.objects.create_user(self.testemail,self.testemail,
self.testpassword)
        self.farmer = Farmer.objects.create(email = self.testemail)
        self.owner = Warehouse_owner.objects.create(email = "2"+
self.testemail)
        self.warehouse =
Warehouse.objects.create(owner=self.owner,poc_phone_no=6543219870)
        self.unit =
Unit.objects.create(warehouse=self.warehouse,capacity=50,price=6000.25)
        self.client.login(username=self.testemail,password=self.testpassword)
        self.url = reverse('search')

    def test_validcase(self):
        response = self.client.get(self.url, {
            'start_date':'2023-12-02',
            'end_date':'2023-12-05',
            'latitude': 23.02,
            'longitude': 72.57,
        })
        # print(response.content)
        self.assertContains(response, 'Jenil Testing: All Good')
        # self.assertRedirects(response, reverse('farmer_profile'),
status_code=302)
        # self.assertTrue(response.url == reverse('search') and
response.status_code == 302)

    def test_end_date_greater_than_start_date(self):
        response = self.client.get(self.url, {
            'startdate':'2023-12-05',
            'enddate':'2023-12-01',
            'latitude': 23.022505,
            'longitude': 72.5713621,
        })
        # print(response.content)
        self.assertContains(response,'Jenil Testing: Invalid Dates')
```

```
    def test_wrong_latitude(self):
        response = self.client.get(self.url, {
            'startdate':'2023-12-01',
            'enddate':'2023-12-05',
            'latitude': -100,
            'longitude': 72.5713621,
        })
        self.assertContains(response, 'Jenil Testing: Invalid Latitude')

    def test_wrong_latitude2(self):
        response = self.client.get(self.url, {
            'startdate':'2023-12-01',
            'enddate':'2023-12-05',
            'latitude': 100,
            'longitude': 72.5713621,
        })
        self.assertContains(response, 'Jenil Testing: Invalid Latitude')

    def test_wrong_longitude(self):
        response = self.client.get(self.url, {
            'startdate':'2023-12-01',
            'enddate':'2023-12-05',
            'latitude': -10.50,
            'longitude': 192.5,
        })
        self.assertContains(response, 'Jenil Testing: Invalid Longitude')

    def test_wrong_longitude2(self):
        response = self.client.get(self.url, {
            'startdate':'2023-12-01',
            'enddate':'2023-12-05',
            'latitude': -50.50,
            'longitude': -192.5,
        })
        self.assertContains(response, 'Jenil Testing: Invalid Longitude')

    def test_wrong_latitude_and_longitude(self):
        response = self.client.get(self.url, {
            'startdate':'2023-12-01',
            'enddate':'2023-12-05',
            'latitude': -100,
            'longitude': 272.5713621,
        })
        self.assertContains(response, 'Jenil Testing: Invalid Latitude and
Longitude')

    def test_wrong_latitude_and_longitude2(self):
        response = self.client.get(self.url, {
```

```
            'startdate':'2023-12-01',
            'enddate':'2023-12-05',
            'latitude': 165.64,
            'longitude': -201.79,
        })
        self.assertContains(response, 'Jenil Testing: Invalid
Latitude and Longitude')
```

2. Main App Testing

Here testing for sign in and sign up are done which works for both farmer and warehouse Owner.

```
from django.test import TestCase, Client
from django.urls import reverse, resolve
from django.contrib.auth.models import User


print("=> Jenil Testing: Sign in")
class TestSignIn(TestCase):
    def setUp(self):
        self.client = Client()
        self.testemail = "Jenil_test@gmail.com"
        self.testpassword = "testpassword"
        self.user = User.objects.create_user(self.testemail,self.testemail,
self.testpassword)
        self.url = reverse('login')

    def test_validcase(self):
        response = self.client.post(self.url, {
            'username': self.testemail,
            'password': self.testpassword
        })
        print(response)
        # self.assertRedirects(response, reverse('Warehouse_profile'),
status_code=302)
        self.assertTrue(response.url == reverse('Warehouse_profile') and
response.status_code == 302)

    def test_invalid_email(self):
        response = self.client.post(self.url, {
            'username': '@gmail.com',
            'password': self.testpassword
        })
        self.assertTrue(response.url == self.url and response.status_code ==
302)

    def test_invalid_email2(self):
```

```
        response = self.client.post(self.url, {
            'username': 'test@.com',
            'password': self.testpassword
        })
        self.assertTrue(response.url == self.url and response.status_code ==
302)

    def test_invalid_email3(self):
        response = self.client.post(self.url, {
            'username': 'test@',
            'password': self.testpassword
        })
        self.assertTrue(response.url == self.url and response.status_code ==
302)

    def test_invalid_password(self):
        response = self.client.post(self.url, {
            'username': self.testemail,
            'password': ''
        })
        self.assertTrue(response.url == self.url and response.status_code ==
302)

    def test_invalid_password2(self):
        response = self.client.post(self.url, {
            'username': self.testemail,
            'password': 'abra_k4_dabra!'
        })
        self.assertTrue(response.url == self.url and response.status_code ==
302)


print("=> Jenil Testing: Sign Up")
class TestSignUp(TestCase):

    def setUp(self):
        self.client = Client()
        self.testemail = "Jeniltest@gmail.com"
        self.testpassword = "Jenil_Test1!"
        self.url = reverse('register')

    def test_validcase(self):
        response = self.client.post(self.url, {
            'email': self.testemail,
            'password1': self.testpassword,
            'password2': self.testpassword,
            'user': 0,
        })
        print(response)
```

```
        # self.assertRedirects(response, reverse('Warehouse_profile'),
status_code=302)
        # self.assertTrue(response.url == reverse('farmer_editprofile') and
response.status_code == 302)
        self.assertTrue(response.status_code == 200) # Because User will be
created and sent to activation code email page

    def test_existing_email(self):
        self.user = User.objects.create_user(self.testemail,self.testemail,
self.testpassword)
        response = self.client.post(self.url, {
            'email': self.testemail,
            'password1': self.testpassword,
            'password2': self.testpassword,
            'user': 0,
        })
        self.assertTrue(response.url == reverse('register') and
response.status_code == 302)


    def test_invalid_chars_in_password(self):
        response = self.client.post(self.url, {
            'email': self.testemail,
            'password1': '1234(da;]ff',
            'password2': '1234(da;]ff',
            'user': 0,
        })
        self.assertTrue(response.url == reverse('register') and
response.status_code == 302)

    def test_invalid_spaces_in_password(self):
        response = self.client.post(self.url, {
            'email': self.testemail,
            'password1': "Jen il_Test1! ",
            'password2': "Jen il_Test1! ",
            'user': 0,
        })
        self.assertTrue(response.url == reverse('register') and
response.status_code == 302)

    def test_invalid_longer_password(self):
        response = self.client.post(self.url, {
            'email': self.testemail,
            'password1': "Jenil_Test1-Longer!#012",
            'password2': "Jenil_Test1-Longer!#012",
            'user': 0,
        })
```

```
        self.assertTrue(response.url == reverse('register') and
response.status_code == 302)

    def test_invalid_shorter_password(self):
        response = self.client.post(self.url, {
            'email': self.testemail,
            'password1': "Je_Te1!",
            'password2': "Je_Te1!",
            'user': 0,
        })
        self.assertTrue(response.url == reverse('register') and
response.status_code == 302)

    def test_invalid_no_numbers_in_password(self):
        response = self.client.post(self.url, {
            'email': self.testemail,
            'password1': "Jenil_Test-!#",
            'password2': "Jenil_Test-!#",
            'user': 0,
        })
        self.assertTrue(response.url == reverse('register') and
response.status_code == 302)

    def test_invalid_no_lowercase_in_password(self):
        response = self.client.post(self.url, {
            'email': self.testemail,
            'password1': "JENIL_TEST-12!#",
            'password2': "JENIL_TEST-12!#",
            'user': 0,
        })
        self.assertTrue(response.url == reverse('register') and
response.status_code == 302)

    def test_invalid_no_special_chars_in_password(self):
        response = self.client.post(self.url, {
            'email': self.testemail,
            'password1': "JenilTest012",
            'password2': "JenilTest012",
            'user': 0,
        })
        self.assertTrue(response.url == reverse('register') and
response.status_code == 302)

    def test_invalid_no_uppercase_in_password(self):
        response = self.client.post(self.url, {
            'email': self.testemail,
            'password1': "jenil_test-12!#",
            'password2': "jenil_test-12!#",
```

```
            'user': 0,
        })
        self.assertTrue(response.url == reverse('register') and
response.status_code == 302)

    def test_invalid_both_blank_password(self):
        response = self.client.post(self.url, {
            'email': self.testemail,
            'password1': "",
            'password2': "",
            'user': 0,
        })
        self.assertTrue(response.url == reverse('register') and
response.status_code == 302)

    def test_invalid_one_blank_password(self):
        response = self.client.post(self.url, {
            'email': self.testemail,
            'password1': "JENIL_TEST-12!#",
            'password2': "JENIL_TEST-12!#",
            'user': 0,
        })
        self.assertTrue(response.url == reverse('register') and
response.status_code == 302)

    def test_invalid_mismatch_password(self):
        response = self.client.post(self.url, {
            'email': self.testemail,
            'password1': "JENIL_TEST-12!#",
            'password2': "JENIL1_TEST-!#2",
            'user': 0,
        })
        self.assertTrue(response.url == reverse('register') and
response.status_code == 302)
```

3. Warehouse side Tests

In this module all the tests for Warehouse owner like adding a warehouse, adding a warehouse unit and editing profile for warehouse owner is done.

```
from django.test import TestCase, Client
from django.urls import reverse, resolve
from Warehouse.views import editprofile
from django.contrib.auth.models import User
from mainapp.models import Warehouse_owner, Warehouse
from django.contrib.messages import get_messages
```

```
print("=> Jenil Testing: Warehouse owner Edit Profile")
class TestWarehouseOwnerEditProfile(TestCase):
    def setUp(self):
        self.client = Client()
        self.testemail = "test@gmail.com"
        self.testpassword = "testpassword"
        self.user = User.objects.create_user(self.testemail,self.testemail,
self.testpassword)
        self.owner = Warehouse_owner.objects.create(email = self.testemail)
        self.client.login(username=self.testemail,password=self.testpassword)
        self.edit_profile_url = reverse('warehouse_editprofile')

    def test_validcase(self):
        response = self.client.post(self.edit_profile_url, {
            'first_name': 'Jenil',
            'last_name':'Patel',
            'phone_no': 1234567890
        })
        print(response)
        # self.assertRedirects(response, reverse('Warehouse_profile'),
status_code=302)
        self.assertTrue(response.url == reverse('Warehouse_profile') and
response.status_code == 302)

    def test_empty_first_name(self):
        response = self.client.post(self.edit_profile_url, {
            'first_name': "",
            'last_name':'Patel',
            'phone_no': 1234567890
            })
        self.assertTrue(response.status_code == 200) # Because Form will
become invalid and if clause will pass

    def test_last_name(self):
        response = self.client.post(self.edit_profile_url, {
            'first_name': "Jenil",
            'last_name':'',
            'phone_no': 1234567890
            })
        self.assertTrue(response.status_code == 200) # Because Form will
become invalid and if clause will pass

    def test_shorter_phone_no(self):
        response = self.client.post(self.edit_profile_url, {
            'first_name': "Jenil",
            'last_name':'Patel',
            'phone_no': 1234567
            })
```

```
        self.assertTrue(response.url == reverse('warehouse_editprofile') and
response.status_code == 302)

    def test_longer_phone_no(self):
        response = self.client.post(self.edit_profile_url, {
            'first_name': "Jenil",
            'last_name':'Patel',
            'phone_no': 1234567890123
            })
        self.assertTrue(response.status_code == 200) # Because Form will
become invalid and if clause will pass


print("=> Jenil Testing: Add Warehouse")
class TestAddWarehouse(TestCase):
    def setUp(self):
        self.client = Client()
        self.testemail = "test@gmail.com"
        self.testpassword = "testpassword"
        self.user = User.objects.create_user(self.testemail,self.testemail,
self.testpassword)
        self.owner = Warehouse_owner.objects.create(email = self.testemail)
        self.client.login(username=self.testemail,password=self.testpassword)
        self.url = reverse('add_warehouse')

    def test_validcase(self):
        response = self.client.post(self.url, {
            'name':"Test Warehouse",
            'address':"Test Address",
            'city':"Test city",
            'state':'Test state',
            'poc_name':'Test POC',
            'phone_no':9876543210,
            'latitude': 60,
            'longitude':120,
        })
        self.assertTrue(response.url == reverse('warehouses') and
response.status_code == 302)

    def test_empty_name(self):
        response = self.client.post(self.url, {
            'name':"",
            'address':"Test Address",
            'city':"Test city",
            'state':'Test state',
            'poc_name':'Test POC',
            'phone_no':9876543210,
            'latitude': 60,
            'longitude':120,
```

```
        })
        self.assertTrue(response.url == reverse('add_warehouse') and
response.status_code == 302)

    def test_empty_address(self):
        response = self.client.post(self.url, {
            'name':"Test Warehouse",
            'address':"",
            'city':"Test city",
            'state':'Test state',
            'poc_name':'Test POC',
            'phone_no':9876543210,
            'latitude': 60,
            'longitude':120,
        })
        self.assertTrue(response.url == reverse('add_warehouse') and
response.status_code == 302)

    def test_empty_city(self):
        response = self.client.post(self.url, {
            'name':"Test Warehouse",
            'address':"Test Address",
            'city':"",
            'state':'Test state',
            'poc_name':'Test POC',
            'phone_no':9876543210,
            'latitude': 60,
            'longitude':120,
        })
        self.assertTrue(response.url == reverse('add_warehouse') and
response.status_code == 302)

    def test_empty_state(self):
        response = self.client.post(self.url, {
            'name':"Test Warehouse",
            'address':"Test Address",
            'city':"Test city",
            'state':'',
            'poc_name':'Test POC',
            'phone_no':9876543210,
            'latitude': 60,
            'longitude':120,
        })
        self.assertTrue(response.url == reverse('add_warehouse') and
response.status_code == 302)

    def test_empty_poc_name(self):
        response = self.client.post(self.url, {
```

```
            'name':"Test Warehouse",
            'address':"Test Address",
            'city':"Test city",
            'state':'Test state',
            'poc_name':'',
            'phone_no':9876543210,
            'latitude': 60,
            'longitude':120,
        })
        self.assertTrue(response.url == reverse('add_warehouse') and
response.status_code == 302)

    def test_empty_shorter_phone_no(self):
        response = self.client.post(self.url, {
            'name':"Test Warehouse",
            'address':"Test Address",
            'city':"Test city",
            'state':'Test state',
            'poc_name':'Test POC',
            'phone_no':987654,
            'latitude': 60,
            'longitude':120,
        })
        self.assertTrue(response.url == reverse('add_warehouse') and
response.status_code == 302)

    def test_empty_longer_phone_no(self):
        response = self.client.post(self.url, {
            'name':"Test Warehouse",
            'address':"Test Address",
            'city':"Test city",
            'state':'Test state',
            'poc_name':'Test POC',
            'phone_no':98765432101234,
            'latitude': 60,
            'longitude':120,
        })
        self.assertTrue(response.url == reverse('add_warehouse') and
response.status_code == 302)

    def test_empty_invalid_latitude(self):
        response = self.client.post(self.url, {
            'name':"Test Warehouse",
            'address':"Test Address",
            'city':"Test city",
            'state':'Test state',
            'poc_name':'Test POC',
            'phone_no':9876543210,
```

```
            'latitude': -100, # should be in range (-90,90)
            'longitude':120,
        })
        self.assertTrue(response.url == reverse('add_warehouse') and
response.status_code == 302)

    def test_empty_invalid_latitude2(self):
        response = self.client.post(self.url, {
            'name':"Test Warehouse",
            'address':"Test Address",
            'city':"Test city",
            'state':'Test state',
            'poc_name':'Test POC',
            'phone_no':9876543210,
            'latitude': 100, # should be in range (-90,90)
            'longitude':120,
        })
        self.assertTrue(response.url == reverse('add_warehouse') and
response.status_code == 302)

    def test_empty_invalid_longitude(self):
        response = self.client.post(self.url, {
            'name':"Test Warehouse",
            'address':"Test Address",
            'city':"Test city",
            'state':'Test state',
            'poc_name':'Test POC',
            'phone_no':9876543210,
            'latitude': 60,
            'longitude': -200, # should be in range (-180,180)
        })
        self.assertTrue(response.url == reverse('add_warehouse') and
response.status_code == 302)

    def test_empty_invalid_longitude2(self):
        response = self.client.post(self.url, {
            'name':"Test Warehouse",
            'address':"Test Address",
            'city':"Test city",
            'state':'Test state',
            'poc_name':'Test POC',
            'phone_no':9876543210,
            'latitude': 60,
            'longitude':200, # should be in range (-180,180)
        })
        self.assertTrue(response.url == reverse('add_warehouse') and
response.status_code == 302)
```

```
print("=> Jenil Testing: Add Warehouse Unit")
class TestAddUnit(TestCase):
    def setUp(self):
        self.client = Client()
        self.testemail = "test@gmail.com"
        self.testpassword = "testpassword"
        self.user = User.objects.create_user(self.testemail,self.testemail,
self.testpassword)
        self.owner = Warehouse_owner.objects.create(email = self.testemail)
        self.warehouse = Warehouse.objects.create(
            name = "Test Warehouse",
            address="Test Address",
            city="Test city",
            state='Test state',
            poc_name='Test POC',
            poc_phone_no = 9876543210,
            latitude = 60,
            longitude =200,
            owner = self.owner)
        self.client.login(username=self.testemail,password=self.testpassword)
        self.url = reverse('addunit', args=(self.warehouse.id,))

    def test_validcase(self):
        response = self.client.post(self.url, {
            'type': "Hot",
            'capacity': 50,
            'price':5000.50,
        })
        self.assertTrue(response.url ==
reverse('all_units',args=(self.warehouse.id,)) and response.status_code ==
302)

    def test_invalid_type(self):
        response = self.client.post(self.url, {
            'type': "Rainy",
            'capacity': 50,
            'price':5000,
        })
        self.assertTrue(response.url == self.url and response.status_code ==
302)

    def test_invalid_capacity(self):
        response = self.client.post(self.url, {
            'type': "Cold",
            'capacity': 0,
            'price':5000,
        })
```

```
        self.assertTrue(response.url == self.url and response.status_code ==
302)

    def test_invalid_capacity2(self):
        response = self.client.post(self.url, {
            'type': "Cold",
            'capacity': -50,
            'price':5000.5,
        })
        self.assertTrue(response.url == self.url and response.status_code ==
302)

    def test_invalid_price(self):
        response = self.client.post(self.url, {
            'type': "Hot",
            'capacity': 50,
            'price': 0,
        })
        self.assertTrue(response.url == self.url and response.status_code ==
302)

    def test_invalid_price2(self):
        response = self.client.post(self.url, {
            'type': "Hot",
            'capacity': 50,
            'price': -600,
        })
        self.assertTrue(response.url == self.url and
response.status_code == 302)
```

Results:

Here after running all these modules, we get result as below which are running completely perfect which displays success in Unit testing.