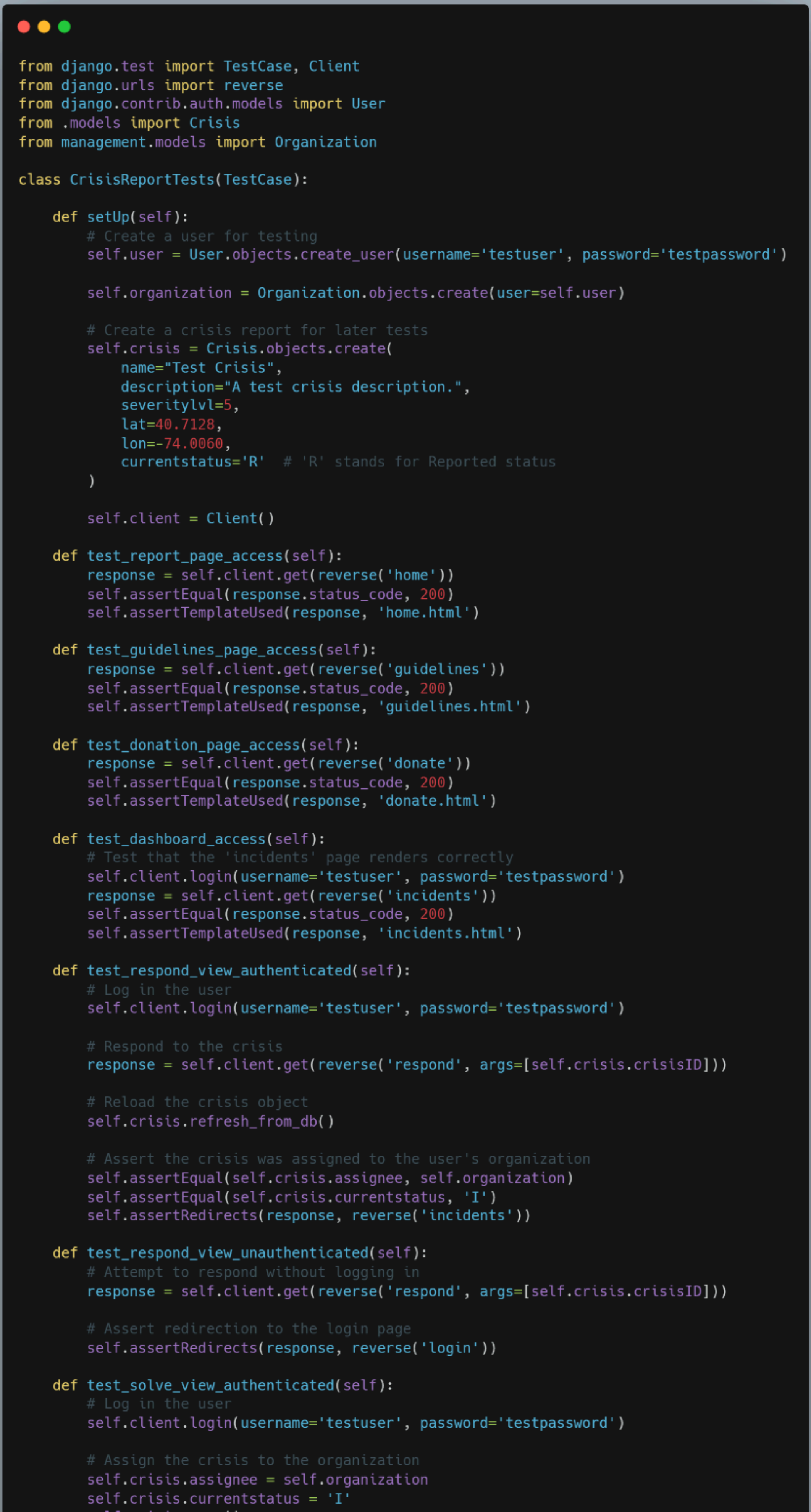
UNIT TESTING

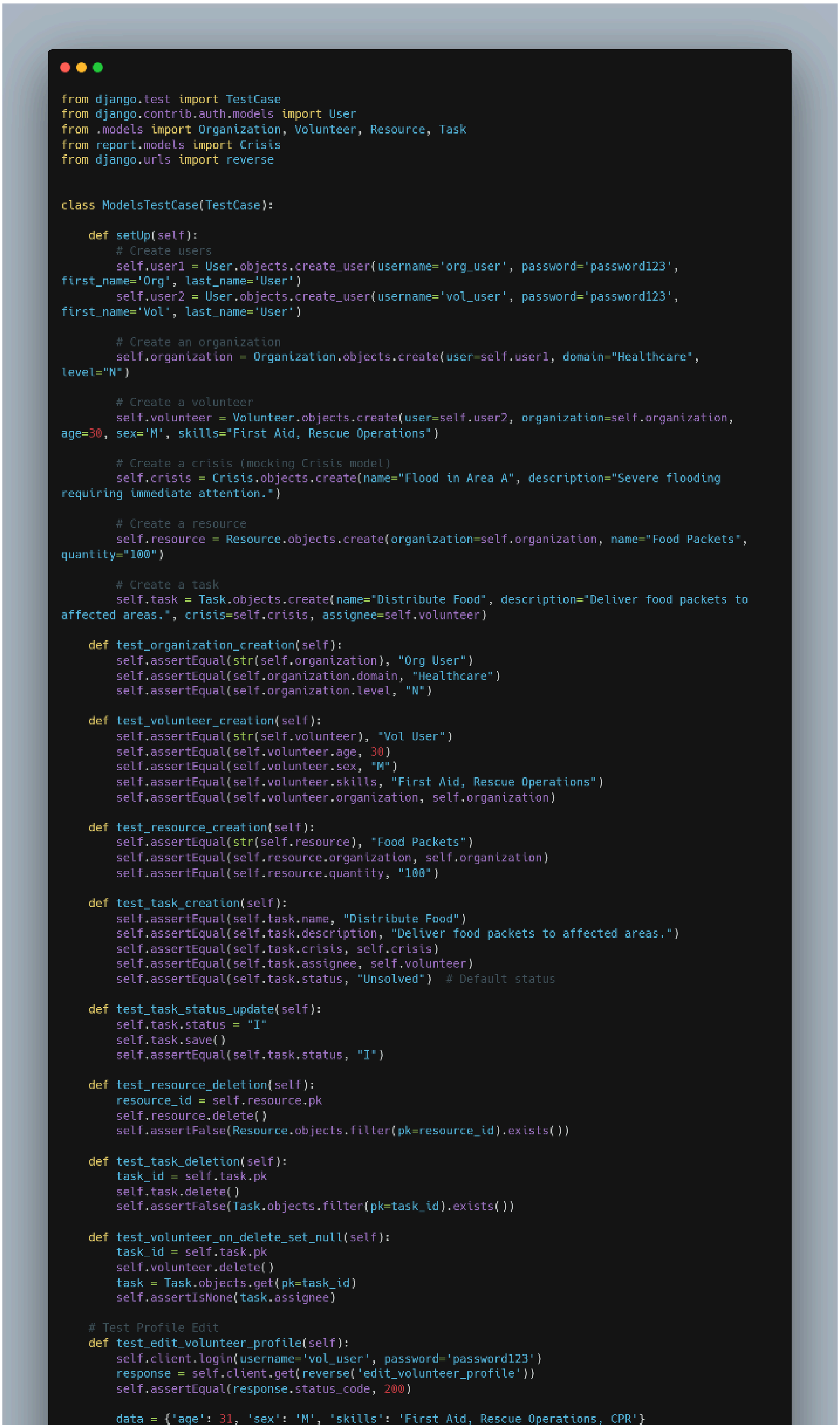
Done using pytest which is inbuilt given in django

1. **Crisis Reporting Tests:**
   - Ensure that views for various pages (home, guidelines, donate, incidents) load correctly.
   - Test authenticated and unauthenticated access for key functionalities like responding to and solving crises.
2. **Model Tests:**
   - Validate the creation and properties of `Organization`, `Volunteer`, `Resource`, and `Task` objects.
   - Ensure the proper handling of status updates, deletion, and cascading behavior (e.g., `on_delete` behavior for volunteers).
3. **Profile Editing Tests:**
   - Confirm that organization and volunteer profiles can be accessed and updated correctly.
4. **Task Management:**
   - Test actions such as accepting and marking tasks as solved.
   - Ensure that the application handles these transitions and permissions correctly.
5. **Logout Test:**
   - Verify that the logout functionality redirects as expected.

For Management & Report App:



```python
from django.test import TestCase, Client
from django.urls import reverse
from django.contrib.auth.models import User
from .models import Crisis
from management.models import Organization

class CrisisReportTests(TestCase):

    def setUp(self):
        # Create a user for testing
        self.user = User.objects.create_user(username='testuser', password='testpassword')

        self.organization = Organization.objects.create(user=self.user)

        # Create a crisis report for later tests
        self.crisis = Crisis.objects.create(
            name="Test Crisis",
            description="A test crisis description.",
            severitylvl=5,
            lat=40.7128,
            lon=-74.0060,
            currentstatus='R'  # 'R' stands for Reported status
        )

        self.client = Client()

    def test_report_page_access(self):
        response = self.client.get(reverse('home'))
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'home.html')

    def test_guidelines_page_access(self):
        response = self.client.get(reverse('guidelines'))
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'guidelines.html')

    def test_donation_page_access(self):
        response = self.client.get(reverse('donate'))
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'donate.html')

    def test_dashboard_access(self):
        # Test that the 'incidents' page renders correctly
        self.client.login(username='testuser', password='testpassword')
        response = self.client.get(reverse('incidents'))
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'incidents.html')

    def test_respond_view_authenticated(self):
        # Log in the user
        self.client.login(username='testuser', password='testpassword')

        # Respond to the crisis
        response = self.client.get(reverse('respond', args=[self.crisis.crisisID]))

        # Reload the crisis object
        self.crisis.refresh_from_db()

        # Assert the crisis was assigned to the user's organization
        self.assertEqual(self.crisis.assignee, self.organization)
        self.assertEqual(self.crisis.currentstatus, 'I')
        self.assertRedirects(response, reverse('incidents'))

    def test_respond_view_unauthenticated(self):
        # Attempt to respond without logging in
        response = self.client.get(reverse('respond', args=[self.crisis.crisisID]))

        # Assert redirection to the login page
        self.assertRedirects(response, reverse('login'))

    def test_solve_view_authenticated(self):
        # Log in the user
        self.client.login(username='testuser', password='testpassword')

        # Assign the crisis to the organization
        self.crisis.assignee = self.organization
        self.crisis.currentstatus = 'I'
```



```
from django.test import TestCase
from django.contrib.auth.models import User
from .models import Organization, Volunteer, Resource, Task
from report.models import Crisis
from django.urls import reverse


class ModelsTestCase(TestCase):

    def setUp(self):
        # Create users
        self.user1 = User.objects.create_user(username='org_user', password='password123',
first_name='Org', last_name='User')
        self.user2 = User.objects.create_user(username='vol_user', password='password123',
first_name='Vol', last_name='User')

        # Create an organization
        self.organization = Organization.objects.create(user=self.user1, domain="Healthcare",
level="N")

        # Create a volunteer
        self.volunteer = Volunteer.objects.create(user=self.user2, organization=self.organization,
age=30, sex='M', skills="First Aid, Rescue Operations")

        # Create a crisis (mocking Crisis model)
        self.crisis = Crisis.objects.create(name="Flood in Area A", description="Severe flooding
requiring immediate attention.")

        # Create a resource
        self.resource = Resource.objects.create(organization=self.organization, name="Food Packets",
quantity="100")

        # Create a task
        self.task = Task.objects.create(name="Distribute Food", description="Deliver food packets to
affected areas.", crisis=self.crisis, assignee=self.volunteer)

    def test_organization_creation(self):
        self.assertEqual(str(self.organization), "Org User")
        self.assertEqual(self.organization.domain, "Healthcare")
        self.assertEqual(self.organization.level, "N")

    def test_volunteer_creation(self):
        self.assertEqual(str(self.volunteer), "Vol User")
        self.assertEqual(self.volunteer.age, 30)
        self.assertEqual(self.volunteer.sex, "M")
        self.assertEqual(self.volunteer.skills, "First Aid, Rescue Operations")
        self.assertEqual(self.volunteer.organization, self.organization)

    def test_resource_creation(self):
        self.assertEqual(str(self.resource), "Food Packets")
        self.assertEqual(self.resource.organization, self.organization)
        self.assertEqual(self.resource.quantity, "100")

    def test_task_creation(self):
        self.assertEqual(self.task.name, "Distribute Food")
        self.assertEqual(self.task.description, "Deliver food packets to affected areas.")
        self.assertEqual(self.task.crisis, self.crisis)
        self.assertEqual(self.task.assignee, self.volunteer)
        self.assertEqual(self.task.status, "Unsolved")  # Default status

    def test_task_status_update(self):
        self.task.status = "I"
        self.task.save()
        self.assertEqual(self.task.status, "I")

    def test_resource_deletion(self):
        resource_id = self.resource.pk
        self.resource.delete()
        self.assertFalse(Resource.objects.filter(pk=resource_id).exists())

    def test_task_deletion(self):
        task_id = self.task.pk
        self.task.delete()
        self.assertFalse(Task.objects.filter(pk=task_id).exists())

    def test_volunteer_on_delete_set_null(self):
        task_id = self.task.pk
        self.volunteer.delete()
        task = Task.objects.get(pk=task_id)
        self.assertIsNone(task.assignee)

    # Test Profile Edit
    def test_edit_volunteer_profile(self):
        self.client.login(username='vol_user', password='password123')
        response = self.client.get(reverse('edit_volunteer_profile'))
        self.assertEqual(response.status_code, 200)

        data = {'age': 31, 'sex': 'M', 'skills': 'First Aid, Rescue Operations, CPR'}
```