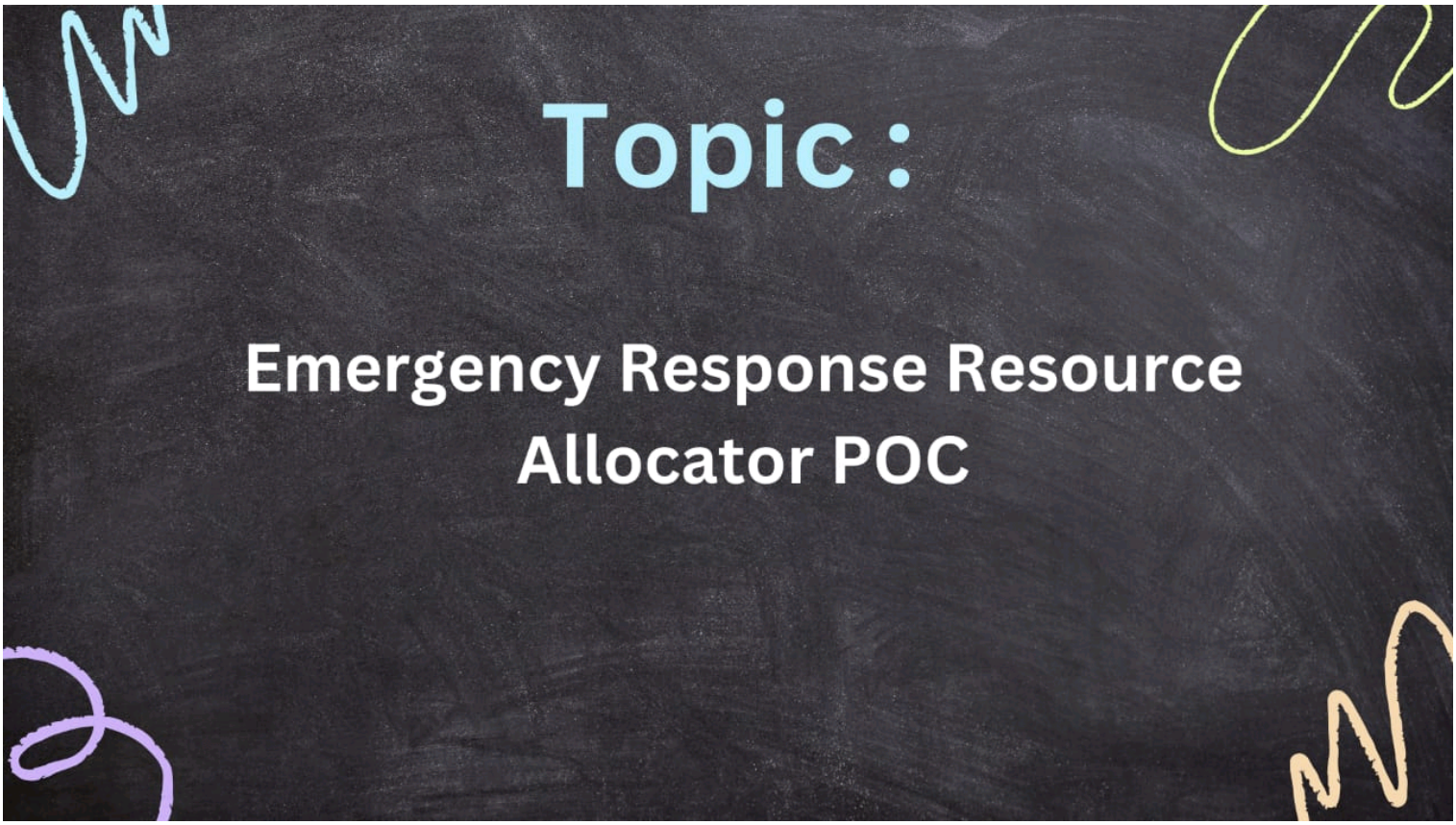
Topic :
Emergency Response Resource Allocator POC

Introduction:

Python Coding: It is an interpreter general purpose high level programming language with easy syntax and dynamic semantics.

Code: Below is a python based proof of concept (POC) following the requirements for an Emergency Response Resource Allocator. This solution includes classes modeling the resources and emergency request handling services, CRUD operations, and methods for resource allocation and monitoring. The application utilizes python's built-in data type and includes unit tests.

Class Resource:

Represents an individual resource
Ex:- ambulance, police unit

```
class Resource :
  def __init__(self, resource_id, type, available=True):
        self.resource_id = resource_id
        self.type = type
        self.available = available
```

Class Emergency Request:

Emergency Request : Handles details about emergency service request.

```
class EmergencyRequest:
    def __init__(self, request_id, required_resource):
        self.request_id = request_id
        self.required_resources = required_resources
        self.allocated_resources = []
        self.status = "Pending"
```

Class Resource Management:

ResourceManager: Manages CRUD operations for resources.

```
class ResourceManager:
    def __init__(self):
        self.resources = {}
```

```python
def add_resource(self, resource):
        self.resources[resource.resource_id] = resource

    def get_resource(self, resource_id):
        return self.resources.get(resource_id)

    def update_resource(self, resource_id, **updates):
        if resource_id in self.resources:
            for key, value in updates.items():
                setattr(self.resources[resource_id], key, value)

    def delete_resource(self, resource_id):
        if resource_id in self.resources:
            del self.resources[resource_id]
    def list_resources(self):
        return list(self.resources.values())
```

Class Emergency Service:

EmergencyService: Handles the allocation of resources and monitoring of resource usage.

```
def _init_(self, resource_manager):
    self.requests = {}
    self.resource_manager = resource_manager

def create_request(self, request_id, required_resources):
    request = EmergencyRequest(request_id, required_resources)
    self.requests[request_id] = request

def allocate_emergency_resources(self, request_id):
    request = self.requests.get(request_id)
    if request and request.status == "Pending":
        for res_type, quantity in request.required_resources.items():
            allocated = 0
            for resource in self.resource_manager.list_resources():
                if resource.type == res_type and resource.available and
                    allocated < quantity:
                    resource.available = False
                    self.resource_manager.update_resource(resource
                        .resource_id, available=False)
```

```
                request.allocated_resources.append(resource)
                allocated += 1
                if allocated == quantity:
                    break
        if allocated < quantity:
            return False  # Not enough resources available
        request.status = "Fulfilled"
        return True
    return False

def monitor_resource_usage(self, request_id):
    request = self.requests.get(request_id)
    if request:
        resource_usage = {}
        for resource in request.allocated_resources:
            resource_usage[resource.resource_id] = {"type": resource
                .type, "status": "in use"}
        return resource_usage
    return None
```

```### Unit Tests Using Python's `unittest` Framework ""
Import unittest

```python
class TestEmergencyResourceAllocator(unittest.TestCase):
    def setUp(self):
        self.resource_manager = ResourceManager()
        self.service = EmergencyService(self.resource_manager)
        self.resource_manager.add_resource(Resource("R1", "ambulance"))
        self.resource_manager.add_resource(Resource("R2", "police"))
        self.service.create_request("REQ1", {"ambulance": 1, "police": 1})

    def test_resource_allocation(self):
        allocation_result = self.service.allocate_emergency_resources("REQ1")
        self.assertTrue(allocation_result)
        self.assertEqual(self.service.requests["REQ1"].status, "Fulfilled")

    def test_resource_monitor(self):
        self.service.allocate_emergency_resources("REQ1")
        usage = self.service.monitor_resource_usage("REQ1")
        self.assertIn("R1", usage)
        self.assertIn("R2", usage)

    def test_insufficient_resources(self):
        self.service.create_request("REQ2", {"ambulance": 2})
        allocation_result = self.service.allocate_emergency_resources("REQ2")
        self.assertFalse(allocation_result)
if _name_ == "_main_":
    unittest.main()
```
```


THANK
YOU
Team Members:-
Harshita P
H Anusha
Priyanka
Sujal
Aaditya