**Report on the Operational Model**

Basic understanding of the model:

We contain 5 sections here,  
**Class Hierarchy:**

* Admin
* Managerial
* Sub-Managerial
* Task And Resource Manager
* Employee

**Responsibilities:**

* Admin: Manages overall tasks, resources, stipends.
* Managerial: Assigns tasks to Sub-Managerial.
* Sub-Managerial: Organizes and divides tasks, manages task and resource manager.
* Task And Resource Manager: Handles task sets, resource sets, and prioritizes tasks.
* Employee: Works on assigned tasks.

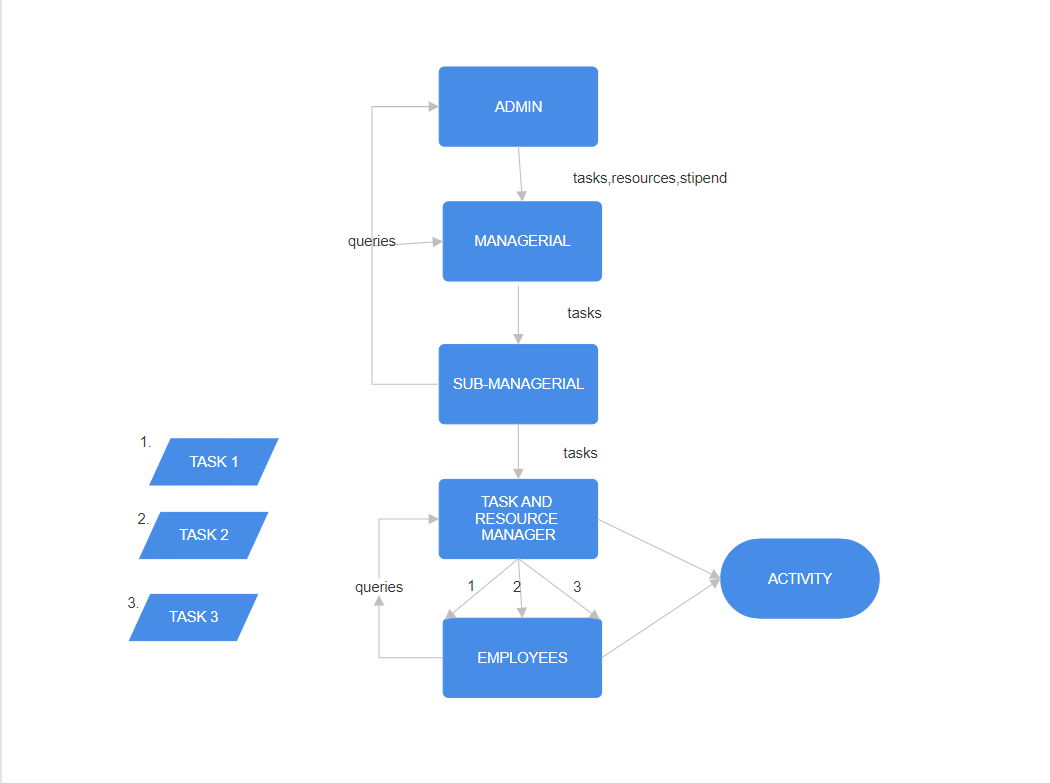
**Task Management:**

* Task set divided into priority set and remaining set.
* Constraint satisfaction to meet deadlines and minimize missed deadlines.
* Manage dependencies, requirements, and mutex for critical sections.

**Activity Page:**

* Track time taken, tasks done, tasks pending, time management.
* Separate sections for each project and employee activities.

**Flowchart:**

****

**Pseudocode:**

#Operational Model

#Task and Resource Class

class Task:

    def \_\_init\_\_(self, id, name, priority, deadline, dependencies=None):

        self.id = id

        self.name = name

        self.priority = priority

        self.deadline = deadline

        self.dependencies = dependencies if dependencies else []

        self.completed = False

class Resource:

    def \_\_init\_\_(self, id, name, quantity):

        self.id = id

        self.name = name

        self.quantity = quantity

#Employee class

class Employee:

    def \_\_init\_\_(self, id, name):

        self.id = id

        self.name = name

        self.tasks = []

    def add\_task(self, task):

        self.tasks.append(task)

#Task and Resource Manager

class TaskAndResourceManager:

    def \_\_init\_\_(self):

        self.tasks = []

        self.resources = []

        self.priority\_set = []

        self.remaining\_set = []

    def add\_task(self, task):

        self.tasks.append(task)

        if task.priority > 5:

            self.priority\_set.append(task)

        else:

            self.remaining\_set.append(task)

    def add\_resource(self, resource):

        self.resources.append(resource)

    def assign\_task\_to\_employee(self, task, employee):

        if task in self.tasks:

            employee.add\_task(task)

            self.tasks.remove(task)

            if task in self.priority\_set:

                self.priority\_set.remove(task)

            elif task in self.remaining\_set:

                self.remaining\_set.remove(task)

        else:

            print("Task not found.")

#Submanagerial

class SubManagerial(Employee):

    def \_\_init\_\_(self, id, name, task\_resource\_manager):

        super().\_\_init\_\_(id, name)

        self.task\_resource\_manager = task\_resource\_manager

    def divide\_and\_assign\_tasks(self, tasks, employees):

        for task in tasks:

            employee = employees.pop(0)

            self.task\_resource\_manager.assign\_task\_to\_employee(task, employee)

            employees.append(employee)

#Managerial

class Managerial(Employee):

    def \_\_init\_\_(self, id, name):

        super().\_\_init\_\_(id, name)

        self.sub\_managers = []

    def add\_sub\_manager(self, sub\_manager):

        self.sub\_managers.append(sub\_manager)

    def assign\_tasks\_to\_sub\_managers(self, tasks):

        for sub\_manager in self.sub\_managers:

            sub\_manager.divide\_and\_assign\_tasks(tasks, sub\_manager.employees)

#Admin

class Admin(Employee):

    def \_\_init\_\_(self, id, name):

        super().\_\_init\_\_(id, name)

        self.managers = []

    def add\_manager(self, manager):

        self.managers.append(manager)

    def assign\_tasks\_to\_managers(self, tasks):

        for manager in self.managers:

            manager.assign\_tasks\_to\_sub\_managers(tasks)

    def distribute\_resources(self, resources):

        for manager in self.managers:

            for sub\_manager in manager.sub\_managers:

                for resource in resources:

                    sub\_manager.task\_resource\_manager.add\_resource(resource)

#Activity Tracking

class ActivityPage:

    def \_\_init\_\_(self):

        self.activities = []

    def log\_activity(self, activity):

        self.activities.append(activity)

    def view\_activities(self):

        for activity in self.activities:

            print(activity)

#Defining all classes

# Create instances of each class

admin = Admin(1, "Admin 1")

manager1 = Managerial(2, "Manager 1")

sub\_manager1 = SubManagerial(3, "SubManager 1", TaskAndResourceManager())

# Add manager and sub-manager to admin and manager

admin.add\_manager(manager1)

manager1.add\_sub\_manager(sub\_manager1)

# Create tasks & resources

task1 = Task(1, "Task 1", 8, "2023-12-01")

task2 = Task(2, "Task 2", 3, "2023-12-05")

resource1 = Resource(1, "Resource 1", 10)

# Assign tasks & resources

admin.assign\_tasks\_to\_managers([task1, task2])

admin.distribute\_resources([resource1])

# Activity login

activity\_page = ActivityPage()

activity\_page.log\_activity("Admin assigned tasks to managers")

activity\_page.log\_activity("Resources distributed")

# View activities

activity\_page.view\_activities()