The University of Azad Jammu and Kashmir

Lab Task # 07

Course Instructor: Engr. Sidra Rafique

Semester: Fall-2024

Session: 2022-2026

Submission Date: Nov 11,2024

Submitted By: Syeda Urwa Ajmal

Roll No: 2022-SE-16

Course Name: SC&D

Code: SE-3102

Implementation of code in MVC architecture

Table of Contents

Implementation of a Task Management System following the Model-View-Controller (MVC) architecture

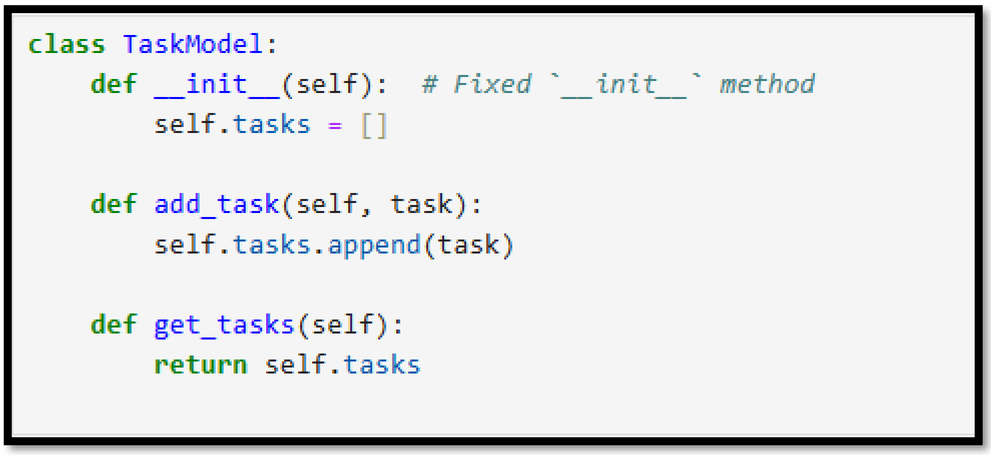
Here's a description of each component and how it fits into the MVC paradigm:

---

1. Model (TaskModel)

The Model represents the data and logic of the application. It handles the storage, modification, and retrieval of tasks.

- Responsibilities:
  - Store the list of tasks in self. tasks.
  - Provide methods for adding tasks (add_task) and retrieving the task list (get_tasks).
- Code Component:
- class TaskModel:

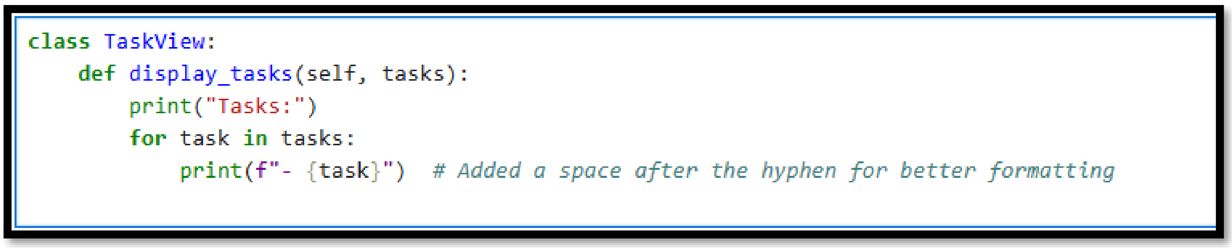


```
class TaskModel:
    def __init__(self):  # Fixed `__init__` method
        self.tasks = []

    def add_task(self, task):
        self.tasks.append(task)

    def get_tasks(self):
        return self.tasks
```

2. View (TaskView)

The View is responsible for presenting the data to the user in a readable format. It does not contain any application logic.

- Responsibilities:
  - Display the list of tasks passed to it by the controller.
  - Handle formatting and output to the console.
- Code Component:
- class TaskView:

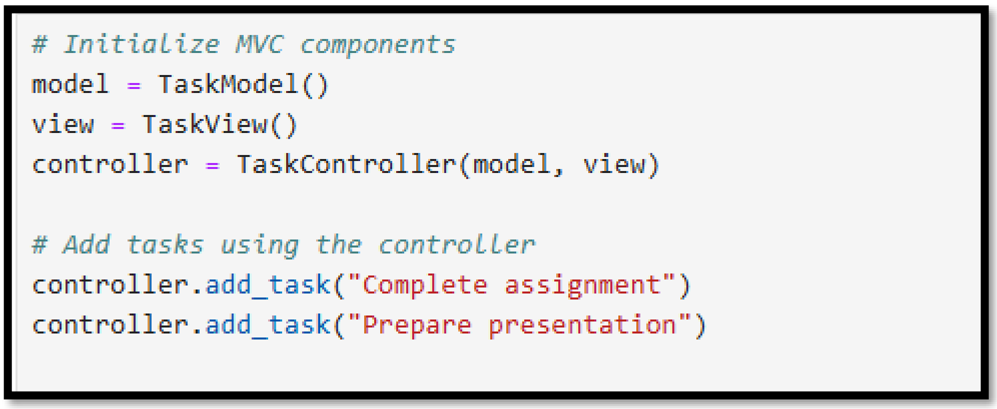


```python
class TaskView:
    def display_tasks(self, tasks):
        print("Tasks:")
        for task in tasks:
            print(f"- {task}")  # Added a space after the hyphen for better formatting
```

3. Controller (Task Controller)

The Controller acts as the intermediary between the Model and the View. It processes user inputs (in this case, method calls in main.py) and updates the Model and View accordingly.

- Responsibilities:
  - Add tasks to the model using add_task.
  - Update the view with the current list of tasks using update_view.
- Code Component:
- class TaskController:

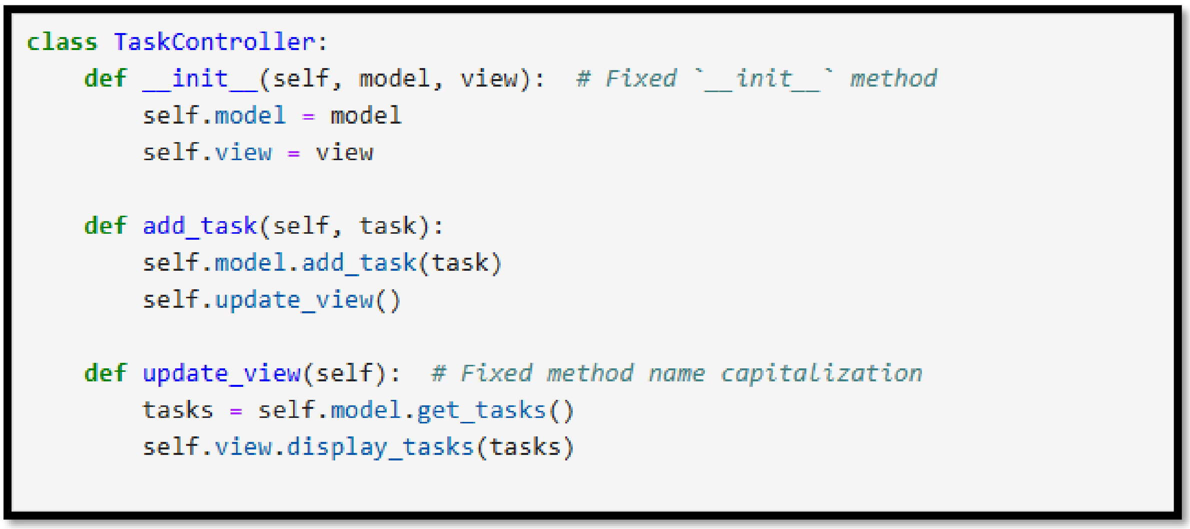


```python
# Initialize MVC components
model = TaskModel()
view = TaskView()
controller = TaskController(model, view)

# Add tasks using the controller
controller.add_task("Complete assignment")
controller.add_task("Prepare presentation")
```

4. Main (Application Entry Point)

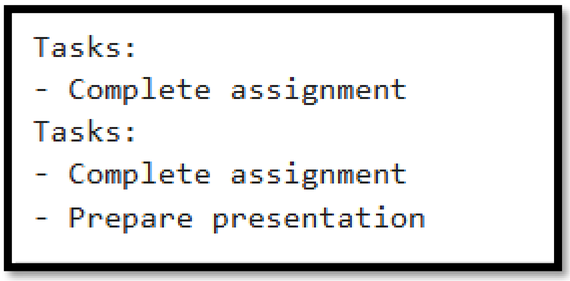
The main.py script initializes the Model, View, and Controller and simulates interaction with the system by adding tasks.

- Responsibilities:
  - Set up the MVC components (TaskModel, TaskView, and TaskController).
  - Pass tasks to the controller for processing.
- Code Component:
- from model import TaskModel
- from view import TaskView
- from controller import TaskController



```
class TaskController:
    def __init__(self, model, view):  # Fixed `__init__` method
        self.model = model
        self.view = view

    def add_task(self, task):
        self.model.add_task(task)
        self.update_view()

    def update_view(self):  # Fixed method name capitalization
        tasks = self.model.get_tasks()
        self.view.display_tasks(tasks)
```

5. Output of the Code:



```
Tasks:
- Complete assignment
Tasks:
- Complete assignment
- Prepare presentation
```

1. Workflow of the Code:

2. Task Addition:
    - The main.py script calls add_task on the controller.
    - The controller updates the model by adding the new task to the task list.
3. Task Display:
    - The controller retrieves the updated list of tasks from the model and passes it to the view.
    - The view displays the tasks in a formatted manner.

---

6. Advantages of Using MVC:

1. Separation of Concerns: Each component has a specific role, making the system easier to maintain and expand.
2. Modularity: The view and model can be modified independently as long as the controller's interface is maintained.
3. Testability: Individual components can be tested in isolation.

This code adheres to the MVC structure, ensuring a clean separation between data, user interface, and business logic.

7. Design Principle:

The **Model (TaskModel)** adheres to principles like **encapsulation, and low coupling**, ensuring focused, maintainable, and scalable task management logic.