By Renugopal Sivaprakasam

MVC

ARCHITECTURE

Table of contents

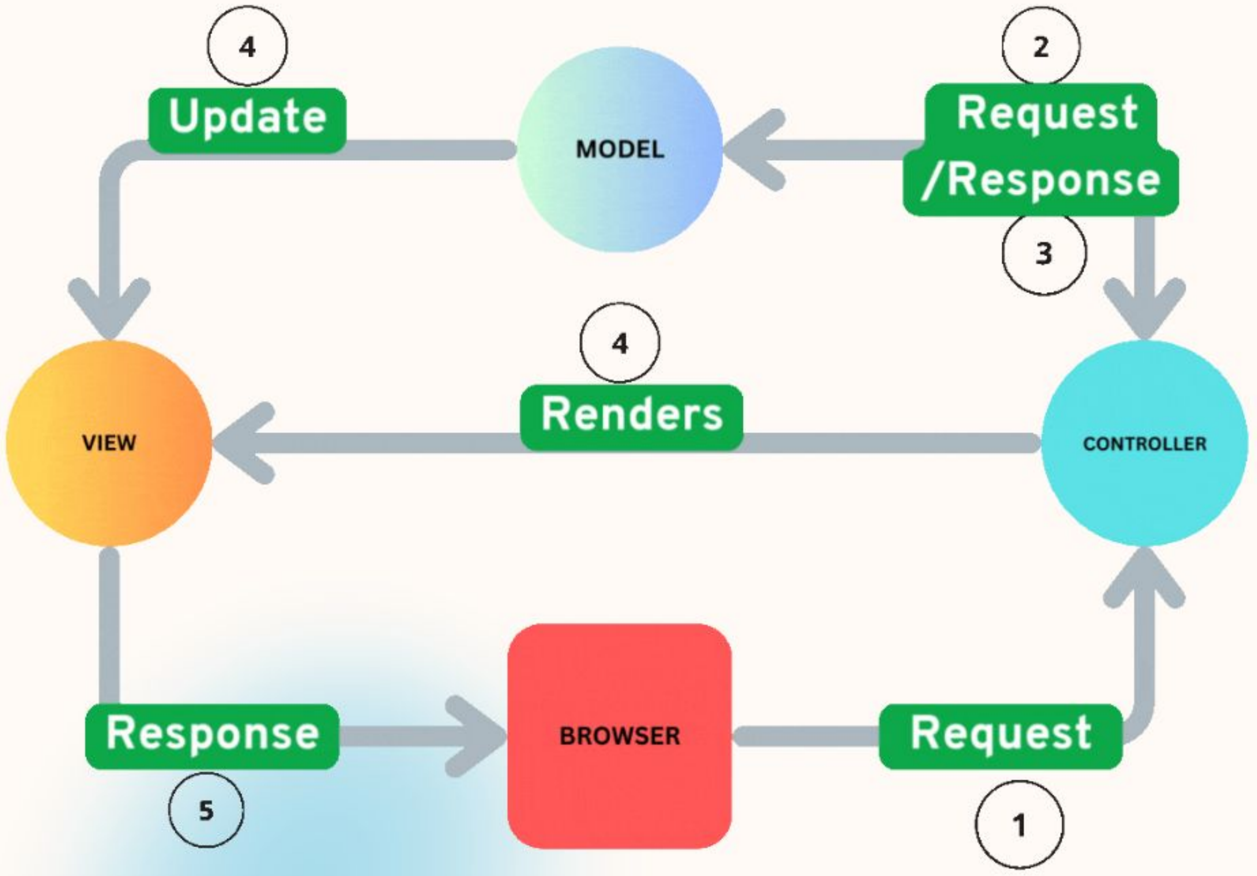
Overview

Like any other curious person, you must have wondered that what is the basic structure of your Web Applications and how they work?

MVC (Model-View-Controller pattern)

Models	Classes representing database
View	How we present it
Controllers	Business Logic and Interaction with the Model

MVC Flow


4
Update
MODEL
2
Request
/Response
3
4
Renders
VIEW
CONTROLLER
Response
5
BROWSER
Request
1

MVC (Model-View-Controller pattern)

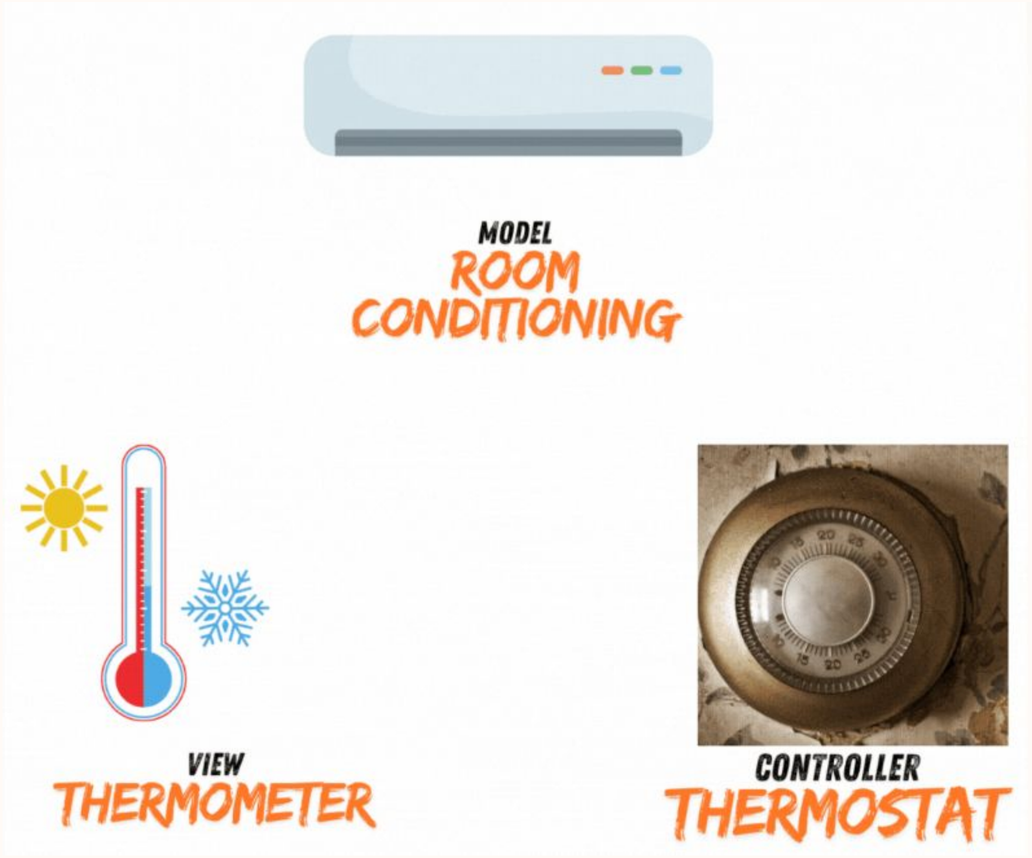
In Model–View–Controller pattern,

Model handles the data and logic. Model interacts with Database, retrieves data as per the requirement.

View, as the name suggests, takes care of the presentation of data. How should the data be presented? How should it look? All those things. Rendering and Data Binding

Controller handles the flow of requests. It can be seen as a middle–man who accepts requests and sends back responses while coordinating with Model and View.

Real world Example



MVC Tutorial using Flask and React

Running the backend application – https://github.com/regostar/mvc_flask/

Step 1: Create a Virtual Environment
Code Snippet:

```
python -m venv venv
source venv/bin/activate
# On Windows:
venv\Scripts\activate
```

Notes:

- Keeps dependencies isolated
- Use the correct activation command for your OS

Step 2: Install Backend Dependencies

Code Snippet:

```
pip install -r requirements.txt
```

Notes:

- Installs all Python packages listed in requirements.txt
- Run this inside the virtual environment

Step 3: Run the Flask Application

Code Snippet:

```
python app.py
```

Notes:

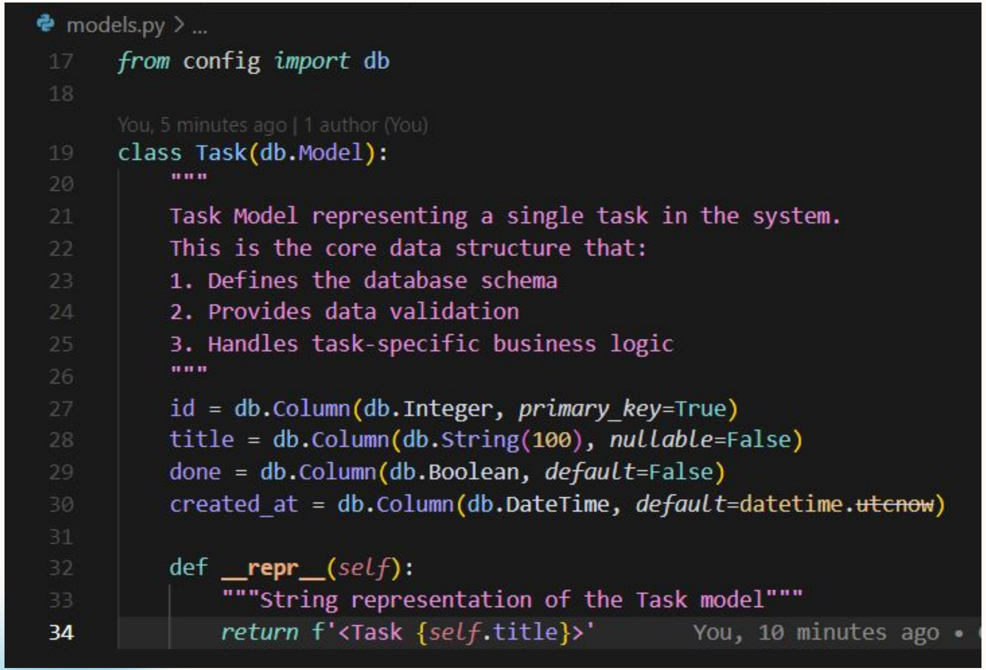
- Starts the Flask backend server
- Make sure you're in the backend directory

MVC Deepdive

-Backend (Flask) MVC Structure:

MODEL

- Located in models.py
- Represents the data structure and business rules
- Contains the database schema and relationships
- Handles data validation and business logic related to data



models.py > ...

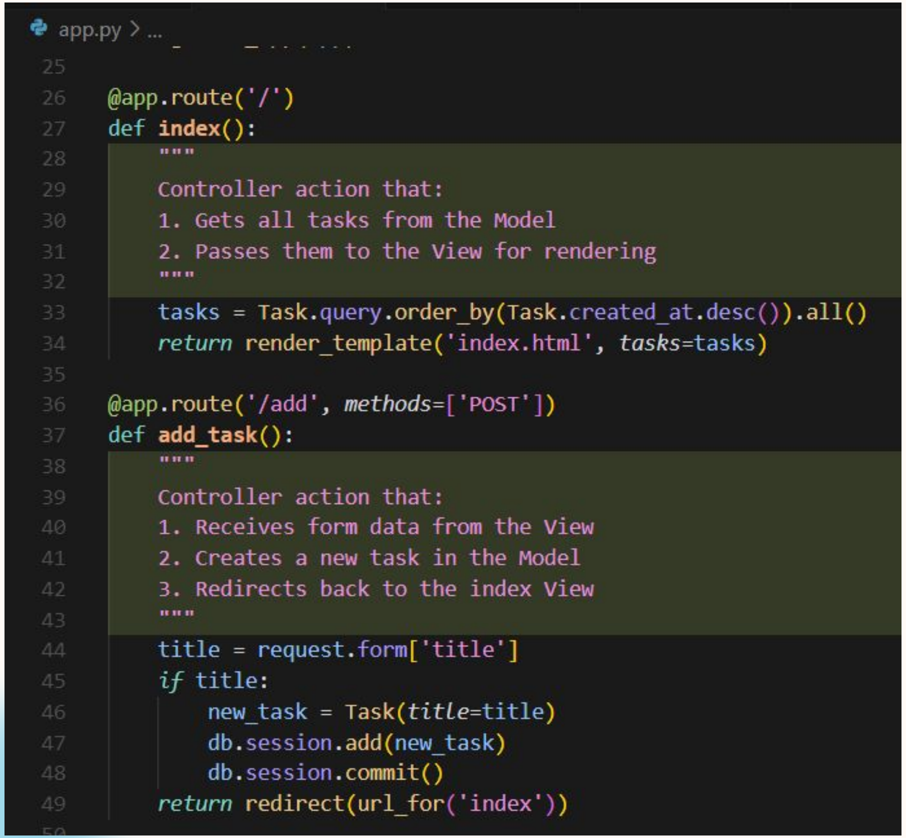
```python
from config import db

You, 5 minutes ago | 1 author (You)
class Task(db.Model):
    """
    Task Model representing a single task in the system.
    This is the core data structure that:
    1. Defines the database schema
    2. Provides data validation
    3. Handles task-specific business logic
    """
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(100), nullable=False)
    done = db.Column(db.Boolean, default=False)
    created_at = db.Column(db.DateTime, default=datetime.~~utcnow~~)

    def __repr__(self):
        """String representation of the Task model"""
        return f'<Task {self.title}>'        You, 10 minutes ago •
```

MVC Deepdive

-Backend (Flask) MVC Structure:

CONTROLLER

- Located in app.py
- Interacts with the model, and sends data to views.
- Contains business logic
- Runs the queries using the models based on the given business logic for a view.



app.py > ...

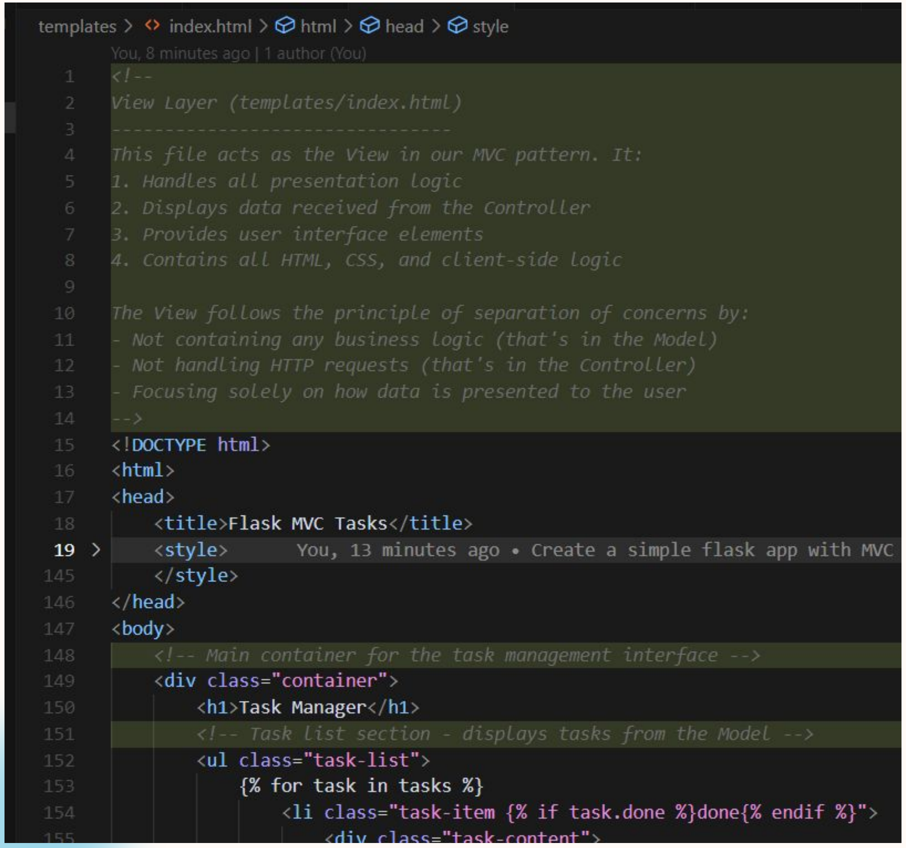
```python

@app.route('/')
def index():
    """
    Controller action that:
    1. Gets all tasks from the Model
    2. Passes them to the View for rendering
    """
    tasks = Task.query.order_by(Task.created_at.desc()).all()
    return render_template('index.html', tasks=tasks)

@app.route('/add', methods=['POST'])
def add_task():
    """
    Controller action that:
    1. Receives form data from the View
    2. Creates a new task in the Model
    3. Redirects back to the index View
    """
    title = request.form['title']
    if title:
        new_task = Task(title=title)
        db.session.add(new_task)
        db.session.commit()
    return redirect(url_for('index'))
```

MVC Deepdive

-Backend (Flask) MVC Structure:

VIEW

- Located in templates/index.html
- Handles the presentation layer
- Views are the components that display the application's user interface (UI).
- Typically, this UI is created from the model data. It is rendered by the Controller


templates > index.html > html > head > style

You, 8 minutes ago | 1 author (You)

```
<!--
View Layer (templates/index.html)
--------------------------------
This file acts as the View in our MVC pattern. It:
1. Handles all presentation logic
2. Displays data received from the Controller
3. Provides user interface elements
4. Contains all HTML, CSS, and client-side logic

The View follows the principle of separation of concerns by:
- Not containing any business logic (that's in the Model)
- Not handling HTTP requests (that's in the Controller)
- Focusing solely on how data is presented to the user
-->
<!DOCTYPE html>
<html>
<head>
    <title>Flask MVC Tasks</title>
    <style>      You, 13 minutes ago • Create a simple flask app with MVC
    </style>
</head>
<body>
    <!-- Main container for the task management interface -->
    <div class="container">
        <h1>Task Manager</h1>
        <!-- Task list section - displays tasks from the Model -->
        <ul class="task-list">
            {% for task in tasks %}
                <li class="task-item {% if task.done %}done{% endif %}">
```

Question 1

The Role of the Model

Question: Which of the following statements best describes the Model in an MVC architecture?

- A. It controls user interactions like button clicks and form submissions
- B. It holds data, manages state, and defines the logic for how data should be stored or retrieved
- C. It renders the user interface and provides styling to the application
- D. It validates user input from forms

Answer 1

The Role of the Model

Question: Which of the following statements best describes the Model in an MVC architecture?

- A. It controls user interactions like button clicks and form submissions
- **B. It holds data, manages state, and defines the logic for how data should be stored or retrieved**
  (The Model is responsible for managing and storing data (including database interactions or in-memory data structures). It also enforces the rules and constraints for how data can be accessed or modified.)
- C. It renders the user interface and provides styling to the application
- D. It validates user input from forms

Question 2

The Communication Flow

Which statement best describes communication flow in a classic (non-framework-specific) MVC?

- A. The View retrieves data directly from the Model and notifies the Controller of any state changes.
- B. The Controller updates the View directly, bypassing the Model for performance reasons.
- C. The View never interacts with the Model directly; the Controller acts as an intermediary.
- D. The Model updates the View only after the database transaction commits.

Question 2

Which statement best describes communication flow in a classic (non-framework-specific) MVC?

- A. The View retrieves data directly from the Model and notifies the Controller of any state changes.
- B. The Controller updates the View directly, bypassing the Model for performance reasons.

- **C. The View never interacts with the Model directly; the Controller acts as an intermediary.**
  In a traditional, strict MVC approach, the View typically does not interact with the Model directly. The Controller is responsible for getting data from the Model and pushing it to the View (or vice versa).

- D. The Model updates the View only after the database transaction commits.

Question 3

Reusability

If you need to expose core business logic to a different interface (e.g., a CLI tool in addition to your web app), which layer of MVC can often be reused with minimal changes?

- A. The Controller, because it has all the request-handling code
- B. The View, because it easily adapts to different output formats
- C. The Model, because it encapsulates the domain logic and can be reused irrespective of the interface
- D. None, you'd have to rebuild everything for the new interface

Answer 3

If you need to expose core business logic to a different interface (e.g., a CLI tool in addition to your web app), which layer of MVC can often be reused with minimal changes?

- A. The Controller, because it has all the request-handling code
  B. The View, because it easily adapts to different output formats
- **C. The Model, because it encapsulates the domain logic and can be reused irrespective of the interface**

  The Model holds the business logic and domain rules, which typically remain consistent across different interfaces (e.g., web, CLI, API). Controllers and Views are usually specific to a particular interface or presentation layer.
- D. None, you'd have to rebuild everything for the new interface

Why we need MVC ?

Separation of Concerns

- Each component has a specific responsibility
- Changes in one component don't affect others

Code Organization

- Clear structure for new features
- Easy to locate specific functionality
- Consistent pattern across the application

Scalability & Maintainability

- Easy to add new features
- Simple to modify existing functionality

MVC in Popular Frameworks

Django: Model-Template- View (MTV)

Flask: Simple MVC-inspired architecture

ASP.NET MVC: Direct MVC implementation

Do you have any questions?

renugopal.sp@gmail.com

Thanks!

Resources

- https://learn.microsoft.com/en-us/aspnet/mvc/overview/older-versions-1/overview/asp-net-mvc-overview
- https://elvinbaghele.medium.com/deep-dive-into-model-view-controller-mvc-best-practices-and-case-studies-c758e13ec4cf
- https://developer.mozilla.org/en-US/docs/Glossary/MVC
- https://plainenglish.io/blog/mvt-architecture-in-django-introduction-and-comparison-with-mvc
- https://www.geeksforgeeks.org/difference-between-mvc-and-mvt-design-patterns/
- https://unt.instructure.com/files/31435645/download?download_frd=1
- https://www.youtube.com/watch?v=r3id0xN8gqo