In Django, **MVT (Model-View-Template)** is the architectural pattern used to structure web applications. It is similar to the **MVC (Model-View-Controller)** pattern but with some differences.

---

1. Model

- **What it is**:
  - The **Model** represents the data layer of the application.
  - It defines the structure of the database and handles all database-related operations (e.g., creating, reading, updating, and deleting records).
- **How it works**:
  - In Django, models are Python classes that subclass `django.db.models.Model`.
  - Each attribute of the model represents a database field.
  - Django’s ORM (Object-Relational Mapper) translates Python code into SQL queries to interact with the database.
- **Example**:

```
from django.db import models

class Task(models.Model):
    title = models.CharField(max_length=200)
    description = models.TextField()
    completed = models.BooleanField(default=False)
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return self.title
```

  - This `Task` model represents a table in the database with fields like `title`, `description`, `completed`, and `created_at`.

---

2. View

- **What it is**:
  - The **View** is the business logic layer of the application.
  - It handles user requests, processes data (interacts with the Model), and returns responses (renders a template or returns JSON).
- **How it works**:
  - In Django, views are Python functions or classes.
  - They receive an HTTP request, perform actions (e.g., querying the database), and return an HTTP response.
- **Example**:

```
from django.shortcuts import render
from .models import Task
```

```python
def task_list(request):
    tasks = Task.objects.all()  # Fetch all tasks from the database
    return render(request, 'tasks/task_list.html', {'tasks': tasks})
```

- This view fetches all tasks from the database and passes them to the `task_list.html` template.

---

3. Template

- **What it is**:
  - The **Template** is the presentation layer of the application.
  - It defines how data is displayed to the user (HTML, CSS, etc.).
  - Templates are dynamic and can include placeholders for data provided by the view.
- **How it works**:
  - Django uses its templating engine to render HTML files.
  - Templates can include variables, loops, conditionals, and other logic to dynamically generate content.
- **Example**:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Task List</title>
</head>
<body>
    <h1>Task List</h1>
    <ul>
        {% for task in tasks %}
            <li>{{ task.title }} - {{ task.completed|yesno:"Completed,Not
Completed" }}</li>
        {% endfor %}
    </ul>
</body>
</html>
```

  - This template loops through the `tasks` passed from the view and displays them in an HTML list.

How MVT Works Together

1. **User Request**:

   - A user sends a request to the Django application (e.g., by visiting a URL).

2. **URL Routing**:

- Django’s URL dispatcher maps the request to the appropriate view based on the URL pattern.

3. **View Processing**:
   - The view handles the request, interacts with the Model to fetch or manipulate data, and prepares the data to be displayed.
4. **Template Rendering**:
   - The view passes the data to a template, which renders the final HTML.
5. **Response**:
   - The rendered HTML is sent back to the user as an HTTP response.

Example Workflow

Let’s say a user visits the URL `/tasks/` to view a list of tasks:

1. **URL Routing**:
   - The URL pattern maps `/tasks/` to the `task_list` view.

     ```
     from django.urls import path
     from . import views
     ```

   urlpatterns = [

   ```
   path('tasks/', views.task_list, name='task_list'),
   ```

   ]

   ```
   ```

2. **View Processing**:
   - The `task_list` view fetches all tasks from the database.

     ```
     def task_list(request):
       tasks = Task.objects.all()
       return render(request, 'tasks/task_list.html', {'tasks': tasks})
     ```

3. **Template Rendering**:
   - The `task_list.html` template receives the `tasks` data and dynamically generates the HTML.

     ```
     <ul>
       {% for task in tasks %}
           <li>{{ task.title }} - {{ task.completed|yesno:"Completed,Not
     Completed" }}</li>
       {% endfor %}
     </ul>
     ```

4. **Response**:

    - The rendered HTML is sent back to the user’s browser.

Key Differences Between MVT and MVC

Aspect	MVC (Model-View-Controller)	MVT (Model-View-Template)
Controller	Handles user input and updates the model.	Django’s framework itself acts as the controller.
View	Displays data to the user.	Handles business logic and passes data to the template.
Template	N/A	Handles the presentation layer (HTML rendering).

Advantages of MVT in Django

1. **Separation of Concerns**:
    - Each component (Model, View, Template) has a specific responsibility, making the code modular and maintainable.
2. **Reusability**:
    - Templates and views can be reused across different parts of the application.
3. **Scalability**:
    - The MVT pattern makes it easier to scale the application as it grows.
4. **Built-in Features**:
    - Django provides built-in tools for handling models, views, and templates, reducing the need for boilerplate code.

Summary

- **Model**: Manages data and database interactions.
- **View**: Handles business logic and processes user requests.
- **Template**: Renders the HTML and presents data to the user.

The MVT pattern is at the core of Django’s design philosophy, making it a powerful and efficient framework for building web applications.