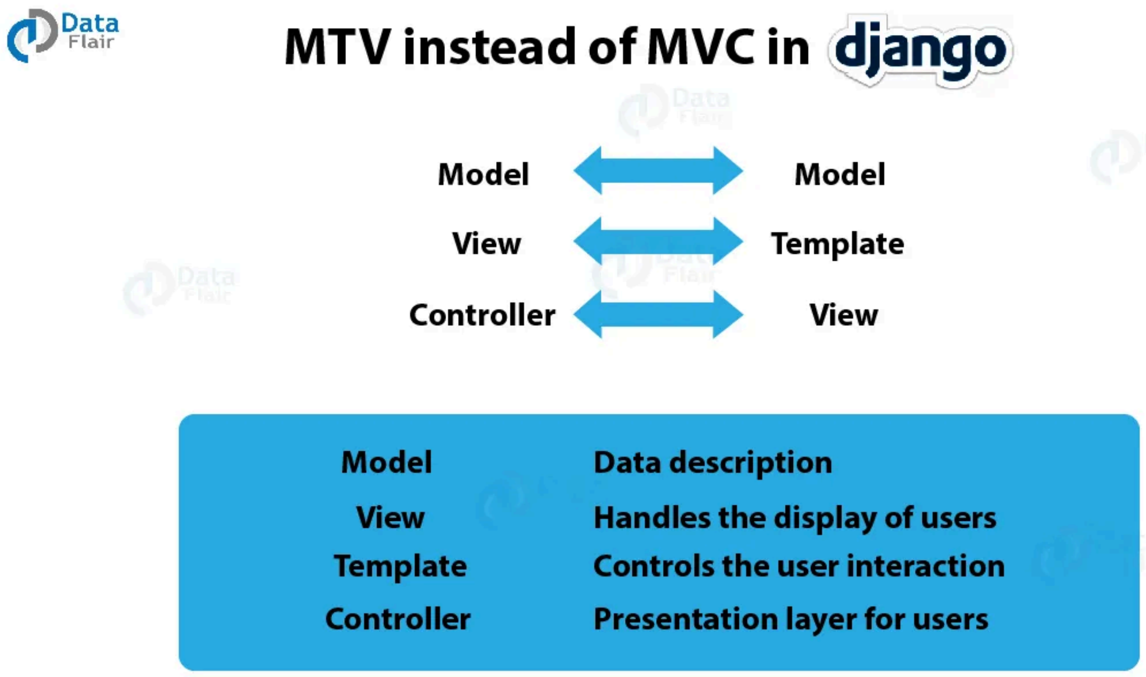
Architectural Pattern for Django

Architectural Pattern for Django

Django follows MTV (Model-Template-View) architecture which is similar to MVC (Model View Controller) architecture. MTV (Model-Template-View) framework uses the terminology Templates for Views and Views for Controller.

Template relates to the View in the MVC pattern as it refers to the presentation layer that manages the presentation logic in the framework and essentially controls the content to display and how to display it for the user.

Thus our Python code will be in views and models and HTML code will be in templates.


MTV instead of MVC in django
Model
Model
View
Template
Controller
View
Model
Data description
View
Handles the display of users
Template
Controls the user interaction
Controller
Presentation layer for users

Description of Model, View, Template

Model

Purpose: Manages the data and business logic.

Responsibilities:

- Defines the structure of the database using Django’s ORM.
- Handles data validation and business rules.
- Represents database tables as Python classes.

```
from django.db import models

# Create your models here.
class GeeksModel(models.Model):
    title = models.CharField(max_length = 200)
    description = models.TextField()
```

View

Purpose: Acts as a bridge between the model and the template.

Responsibilities:

- Processes user requests (e.g., HTTP requests).
- Retrieves data from the model and prepares it for the template.
- Determines which template to render based on the request.

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

def post_list(request):
    posts = Post.objects.all()
    return render(request, 'blog/post_list.html', {'posts': posts})
```

Template

Purpose: Defines the presentation layer.

Responsibilities:

- Renders the HTML output that users see.
- Displays data passed from the view in a user-friendly format.
- Supports template inheritance and includes to promote reusability.

```
<!DOCTYPE html>
<html>
<head>
    <title>Blog Posts</title>
</head>
<body>
    <h1>Blog Posts</h1>
    {% for post in posts %}
        <h2>{{ post.title }}</h2>
        <p>{{ post.content }}</p>
        <p><em>Published on {{ post.published_date }}</em></p>
    {% endfor %}
```

```
</body>
</html>
```

Benefits of MVT in Django

1. Separation of Concerns

- **Modularity**: Components can be developed and maintained independently.
- **Clarity**: Developers can focus on specific areas without needing to understand the entire codebase.

2. Reusability

- **Code Reuse**: Models and templates can be reused across different parts of the application.
- **Template Inheritance**: Allows for a base template, enabling consistent layouts across multiple pages.

3. Ease of Maintenance

- **Simplified Updates**: Changes to the user interface or data logic can be made independently.
- **Reduced Complexity**: Clear organization of code leads to easier debugging and testing.

4. Enhanced Collaboration

- **Parallel Development**: Team members can work on models, views, and templates simultaneously.
- **Clear Roles**: Designers focus on templates, while backend developers work on models and views.

5. Better Scalability

- **Flexible Architecture**: Allows for easier scalability as applications grow.
- **Efficient Database Management**: ORM streamlines database interactions.

6. Improved Testing

- **Unit Testing**: Each component can be tested in isolation.
- **Mocking and Stubbing**: Easier to mock dependencies in unit tests.

7. User-Centric Development

- **Responsive Design**: Templates can be designed for better user experiences.
- **Dynamic Content**: Easily serve dynamic content based on user interactions.

Author:

- Suraiya Mahmuda
- Mahfuz Anam
- Mohammed Tamjid Islam

- Kamrul Hasan Nahid

Reference:

- https://www.geeksforgeeks.org/django-project-mvt-structure/
- https://medium.com/@rinu.gour123/3-major-architectural-pattern-django-follows-8c7aff0a5af6