Class Notes: Django MVT Architecture

Learn:

- **Model View Template (MVT) Architecture**:
- Django follows the MVT architectural pattern, which is a variation of the popular MVC (Model-View-Controller) pattern.
- It separates the application logic into three interconnected components: Model, View, and Template.
- MVT helps to maintain clean and organized code by separating concerns and promoting reusability.

- **Model (M)**:
- Models represent the data structure of the application.
- In Django, models are Python classes that define the database schema.
- They encapsulate data access logic and provide an abstraction layer to interact

with the database.
- Django's ORM (Object-Relational Mapping) translates these model classes into database tables.

- **View (V)**:
- Views are responsible for processing user requests and returning appropriate responses.
- In Django, views are Python functions or classes that handle HTTP requests.
- They interact with models to retrieve or manipulate data and render templates to generate HTML responses.
- Views encapsulate business logic and control the flow of information between the model and the template.

- **Template (T)**:
- Templates are used to generate dynamic HTML content based on data provided by views.
- In Django, templates are written in HTML with embedded template tags and filters.

- They allow developers to separate the presentation layer from the application logic.
- Templates support inheritance, inclusion, and other features to promote code reuse and maintainability.

Practice:

1. **Define Models**:
- Create Python classes in the `models.py` file, representing the data entities of the application.
- Define attributes and methods to represent the structure and behavior of each model.
- Use Django's ORM to specify relationships between models and map them to database tables.

2. **Write Views**:
- Define view functions or classes in the `views.py` file to handle different HTTP requests.
- Implement business logic to process

data and generate responses.
- Utilize Django's class-based views or function-based views depending on the complexity of the application.

3. **Create Templates**:
- Design HTML templates in the `templates` directory to define the presentation layer.
- Use template tags and filters to dynamically render data from views.
- Organize templates into reusable components and extend base templates to maintain consistency across the application.

4. **Wire URLs**:
- Define URL patterns in the `urls.py` file to map URLs to corresponding view functions or classes.
- Use Django's URL dispatcher to route requests to the appropriate views based on URL patterns.

5. **Testing and Debugging**:

- Test views and templates using Django's testing framework to ensure functionality and correctness.
- Debug issues by inspecting error messages, logging, and using Django's debugging tools.

Know More (FAQs):

- **Q: Can we use other template engines instead of Django's default template engine?**
- Yes, Django allows using alternative template engines like Jinja2 by configuring the TEMPLATES setting in the settings.py file.

- **Q: How does Django handle authentication and authorization within the MVT architecture?**
- Django provides built-in authentication and authorization mechanisms that can be integrated into views to enforce access control and user authentication.

- **Q: Is it possible to implement AJAX functionality within Django views?**
- Yes, Django supports AJAX functionality by returning JSON responses from views and handling asynchronous requests using JavaScript on the client-side.