1. Questions About Django's MVT Architecture

Django uses the Model-View-Template (MVT) architecture, which is slightly different from classic MVC.

1.1. Basic MVT Structure

Question
> *Can you explain the MVT architecture in Django?*

Answer Outline
1. Model:
   - Represents the data layer (database tables) in your application.
   - Defined in `models.py`.
2. View:
   - Handles business logic and HTTP requests/responses.
   - Defined in `views.py`.
3. Template:
   - The presentation layer (HTML, CSS, JavaScript).
   - Stored in `templates/`.
4. How It Flows:
   - The user makes a request → Django URL dispatcher routes it → A View processes data (possibly querying a Model) → The View hands data to a Template for rendering → HTML is returned to the user.

Key Points
- Stress why MVT is important: separation of concerns.
- Emphasize that Django's "View" in MVT is somewhat analogous to a Controller in MVC.

1.2. Role of URLs in MVT

Question
> *How do URLs fit into Django's MVT architecture?*

Answer Outline
1. URL Dispatcher:
   - Django uses `urls.py` to map URL patterns to specific View functions/classes.
2. Why:
   - This decouples the request routing from the logic in views and helps maintain a clean structure.

---

 2. Questions About Django ORM (Object-Relational Mapping)

Django’s ORM abstracts database operations so you can work in Python instead of writing raw SQL.

# 2.1. Basic Model Definition

Question
> *How do you define a model in Django?*

Answer Outline
1. Subclass `models.Model`:
   ```python
   from django.db import models

   class Book(models.Model):
       title = models.CharField(max_length=100)
       author = models.CharField(max_length=100)
       published_date = models.DateField()
   ```
2. Fields:
   - Use specific field classes (`CharField`, `DateField`, etc.).
   - `max_length` is required for text-based fields like `CharField`.
3. Migration:
   - After defining, you run `python manage.py makemigrations` and `python manage.py migrate`.

2.2. CRUD Operations with Django ORM

Question
> *How would you perform basic CRUD (Create, Read, Update, Delete) with Django ORM?*

Answer Outline
1. Create:
   ```python
   book = Book.objects.create(title="Django Basics", author="Alice", published_date="2025-01-01")
   ```
2. Read (Query):
   ```python
   books = Book.objects.all()
   single_book = Book.objects.get(id=1)
   filtered_books = Book.objects.filter(author="Alice")
   ```
3. Update:
   ```python
   book = Book.objects.get(id=1)
   book.title = "Updated Title"
   book.save()
   ```
4. Delete:
   ```python
   book = Book.objects.get(id=1)
   book.delete()
   ```

Key Points
- Mention difference between `get()` (returns single object or raises `DoesNotExist`) and `filter()` (returns a `QuerySet`, can be empty).
- Stress that each model has `save()` and `delete()` methods.

# 2.3. Relationships

Question
> *How do you handle relationships between models in Django?*

Answer Outline
1. One-to-Many:
   - `models.ForeignKey(OtherModel, on_delete=models.CASCADE)`
2. Many-to-Many:
   - `models.ManyToManyField(OtherModel)`
3. One-to-One:
   - `models.OneToOneField(OtherModel, on_delete=models.CASCADE)`

Explanation
- Emphasize that these relationships make it easy to navigate related objects using dot notation (e.g., `book.author_set.all()` if you set up foreign keys properly).

---

 3. Questions About Middleware

Middleware in Django is a framework of hooks into the request/response processing.

# 3.1. Basic Middleware Concept

Question
> *What is Django Middleware, and why is it useful?*

Answer Outline
1. Definition:
   - Components that process requests and/or responses globally before/after the View is called.
2. Examples:
   - AuthenticationMiddleware sets `request.user`.
   - SessionMiddleware manages user sessions.

3. Use Cases:
   - Security (e.g., CSRF protection), logging, modifying response headers, etc.

## 3.2. Writing Custom Middleware

Question
> *How do you create and use custom middleware in Django?*

Answer Outline
1. Create a Class with `__call__` or old-style `process_request`, `process_response`:
   ```python
   class MyCustomMiddleware:
       def __init__(self, get_response):
           self.get_response = get_response

       def __call__(self, request):
           # Code to process request before view
           response = self.get_response(request)
           # Code to process response after view
           return response
   ```
2. Add to `MIDDLEWARE` in `settings.py`:
   ```python
   MIDDLEWARE = [
       # ...
       'path.to.MyCustomMiddleware',
   ]
   ```
3. Why:
   - Could track performance, authenticate users differently, etc.

---

4. Entry-Level Django Questions

These questions often appear for junior or entry-level roles to ensure basic familiarity with Django.

4.1. Project and App Structure

Question
> *What’s the difference between a Django project and a Django app?*

Answer Outline
- Project:
  - The overall site configuration, containing settings, URLs, WSGI, etc.
  - Created with `django-admin startproject`.
- App:
  - A self-contained module or component (like “blog”, “shop”).
  - Created with `python manage.py startapp myapp`.
- Django projects typically contain multiple apps.

4.2. Managing Migrations

Question
> *How do you handle database changes in Django?*

Answer Outline
1. Migrations:
   - Use `makemigrations` to create migration files.
   - Use `migrate` to apply them to the database.
2. Rollbacks:
   - Typically done by reversing a migration or manually adjusting if needed.

4.3. Admin Site Basics

Question
> *How do you enable and use the Django admin site?*

Answer Outline
1. Enable Admin:
   - Add `'django.contrib.admin'` to `INSTALLED_APPS`.
   - Include `path('admin/', admin.site.urls)` in `urls.py`.
2. Create Superuser:
   ```bash
   python manage.py createsuperuser
   ```
3. Register Models in `admin.py`:
   ```python
   from django.contrib import admin
   from .models import Book

   admin.site.register(Book)
   ```
4. Access:
   - Go to `http://127.0.0.1:8000/admin/`.

# 4.4. Virtual Environments and Requirements

Question
> *Why use a virtual environment, and how do you share project dependencies?*

Answer Outline
1. Virtual Environment:
   - Isolates your project’s Python packages from system-wide packages.
2. Pip and `requirements.txt`:
   ```bash
   pip freeze > requirements.txt
   ```

```
# Later or on another machine
pip install -r requirements.txt
```

## 4.5. URL Routing

Question
> *How do you configure URL routing in Django for a specific app?*

Answer Outline
1. Project `urls.py`:
   ```python
   from django.urls import path, include

   urlpatterns = [
       path('admin/', admin.site.urls),
       path('blog/', include('blog.urls')),
   ]
   ```
2. App-level `urls.py` (e.g., `blog/urls.py`):
   ```python
   from django.urls import path
   from . import views

   urlpatterns = [
       path('', views.home, name='blog-home'),
       path('post/<int:id>/', views.post_detail, name='post-detail'),
   ]
   ```

---

5. General Tips for Answering Django Questions

1. Use Examples:
   - Demonstrate with short code snippets—interviewers often want to see how you do something in addition to an explanation why.
2. Explain the “Why”:
   - If asked about MVT, mention it helps separate data (Model), logic (View), and presentation (Template).
   - If asked about ORM, note that it allows you to avoid raw SQL and reduces boilerplate.
3. Highlight Best Practices:
   - Using virtual environments (`venv`), implementing tests, applying migrations properly, etc.
4. Mention Common Pitfalls:
   - For example, forgetting to add your app to `INSTALLED_APPS`, misplacing your middleware in the wrong order, etc.
5. Know Core Django Commands:
   - `startproject`, `startapp`, `runserver`, `makemigrations`, `migrate`, `createsuperuser`, etc.

# Example Combined Question

Question
> *Walk me through creating a new Django project called “mysite”, an app called “blog”, defining a simple model, setting up a URL route, and registering it in the admin site?*

How to Answer
1. Create Project:
   ```bash
   django-admin startproject mysite
   ```
2. Create App:
   ```bash
   cd mysite
   python manage.py startapp blog
   ```
3. Define Model (`blog/models.py`):
   ```python
   from django.db import models
   ```

```python
   class Post(models.Model):
       title = models.CharField(max_length=200)
       content = models.TextField()
       created_at = models.DateTimeField(auto_now_add=True)

       def __str__(self):
           return self.title
   ```
4. Register the App in `mysite/settings.py`:
   ```python
   INSTALLED_APPS = [
       'django.contrib.admin',
       'django.contrib.auth',
       ...
       'blog',
   ]
   ```
5. Migrations:
   ```bash
   python manage.py makemigrations
   python manage.py migrate
   ```
6. Register in Admin (`blog/admin.py`):
   ```python
   from django.contrib import admin
   from .models import Post

   admin.site.register(Post)
   ```
7. Set Up URL (`mysite/urls.py`):
   ```python
   from django.urls import path, include

   urlpatterns = [
   ```

```python
       path('admin/', admin.site.urls),
       path('blog/', include('blog.urls')),
   ]
```

8. App-level URLs (`blog/urls.py`):
   ```python
   from django.urls import path
   from . import views

   urlpatterns = [
       path('', views.post_list, name='post-list'),
   ]
   ```
9. View (`blog/views.py`):
   ```python
   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})
   ```
10. Create Template (`blog/templates/blog/post_list.html`):
    ```html
    <h1>Blog Posts</h1>
    {% for post in posts %}
      <h2>{{ post.title }}</h2>
      <p>{{ post.content }}</p>
    {% endfor %}
    ```

Key Takeaway
- This workflow addresses models, views, templates, URLs, and the admin site—covering core Django concepts at an entry level.

```
 Final Thoughts

- MVT: Understand the separation of concerns between models, views, and templates.
- ORM: Practice CRUD and relationship queries.
- Middleware: Know how to create custom middleware and understand built-in ones (like session, CSRF).
- Entry-Level: Grasp project/app structure, URL routing, admin setup, migrations, and best practices for using virtual
environments.
```