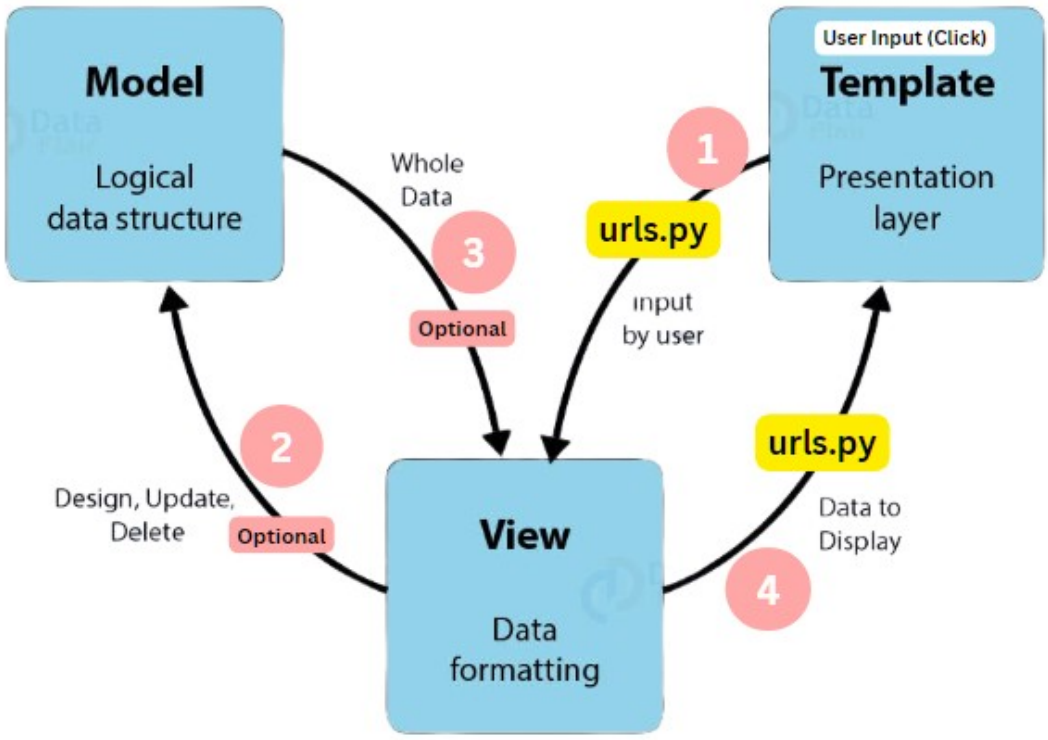
Lecture 16: Django Architecture

`urls, views, templates`

Prepared by: Durjoy Mistry Lecturer, Department of CSE University of Asia Pacific

MVT Overview



Basic Setup

In this tutorial, I'll guide you through setting up a basic Django project, creating multiple HTML pages, and linking them using URLs and views.

Step 1: Install Django

First, make sure you have Python installed on your system. Then, install Django using pip:

```
pip install django
```

Step 2: Create a Django Project

Now, let's create a new Django project. Open your terminal and run:

```
django-admin startproject myproject
```

This will create a directory called `myproject` with the basic structure for a Django project.

Step 3: Create an App

In Django, functionality is organized into apps. Let's create an app within our project:

```
cd myproject
python manage.py startapp myapp
```

This will create a directory called `myapp` with the necessary files for our app.

Step 4: Define URL patterns

Open `myproject/myapp/urls.py` and define your URL patterns. Here's a simple example:

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

urlpatterns = [
    path('', views.home, name='home'),
    path('about/', views.about, name='about'),
    path('contact/', views.contact, name='contact'),
]
```

Step 5: Create Views

In Django, views are Python functions that take a web request and return a web response. Let's create some views. Open `myproject/myapp/views.py`:

```
from django.shortcuts import render
from django.http import HttpResponse

def home(request):
    return render(request, 'home.html')

def about(request):
    return render(request, 'about.html')

def contact(request):
    return render(request, 'contact.html')
```

Step 6: Create HTML Templates

Create HTML templates for each view in the `myproject/myapp/templates` directory:

- `home.html`
- `about.html`
- `contact.html`

These templates will contain the HTML code for your pages.

Step 7: Configure URLs

In your project's main `urls.py` file (located at `myproject/urls.py`), include your app's URLs:

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('myapp.urls')),
]
```

Step 8: Run the Server

Finally, start the Django development server:

```
python manage.py runserver
```

Now you can visit `http://127.0.0.1:8000/` in your browser to see your home page, `http://127.0.0.1:8000/about/` for the about page, and `http://127.0.0.1:8000/contact/` for the contact page. Django will route these URLs to the appropriate views and templates.

Navigate from one page to another

Step 1: Create HTML Templates

Create HTML templates for each of your pages (`home.html`, `about.html`, `contact.html`) in the `myapp/templates` directory.

Here's an example of `home.html`:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
```

```
scale=1.0">
    <title>Home</title>
</head>
<body>
    <h1>Welcome to the Home Page!</h1>
    <a href="{% url 'about' %}"><button>About</button></a>
    <a href="{% url 'contact' %}"><button>Contact</button></a>
</body>
</html>
```

Similarly, create `about.html` and `contact.html` with appropriate content and navigation buttons.

Step 2: Define URL Patterns

In your app's `urls.py` file (`myapp/urls.py`), define the URL patterns:

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

urlpatterns = [
    path('', views.home, name='home'),
    path('about/', views.about, name='about'),
    path('contact/', views.contact, name='contact'),
]
```

Step 3: Create Views

In your app's `views.py` file (`myapp/views.py`), define the views:

```
from django.shortcuts import render

def home(request):
    return render(request, 'home.html')

def about(request):
    return render(request, 'about.html')

def contact(request):
    return render(request, 'contact.html')
```

Step 4: Configure URLs

In the main `urls.py` file of your Django project (`myproject/urls.py`), include your app's URLs:

```
from django.contrib import admin
from django.urls import path, include
```

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('myapp.urls')),
]
```

Step 5: Run the Server

Finally, start the Django development server:

```
python manage.py runserver
```

You can now navigate to `http://127.0.0.1:8000/` in your browser to see the home page with buttons to navigate to the about and contact pages.

This setup will allow you to create HTML pages with navigation buttons that link to other pages within your Django project. Feel free to expand upon this by adding more views, templates, and functionality as needed.

Video Tutorial

Play Video

Lecture 17: Superuser, models, admin

```
Superuser, models, admin
```

Prepared by: Durjoy Mistry Lecturer, Department of CSE University of Asia Pacific

Superuser

Creating a superuser in Django is a straightforward process. Superusers have access to the Django admin interface and can perform administrative tasks such as managing users, groups, and other site content. Here's how you can create a superuser in Django:

if you are creating a super user for the first ime in a projet just run:

```
python manage.py migrate
```

Or just follow:

Step 1: Navigate to Your Django Project Directory

Open your terminal or command prompt and navigate to the root directory of your Django project.

Step 2: Run the `createsuperuser` Management Command

Use the `createsuperuser` management command provided by Django. Enter the following command:

```
python manage.py createsuperuser
```

Step 3: Enter Superuser Details

After running the command, you will be prompted to enter details for the superuser:

- Username: This is the username for the superuser. Choose a unique username.
- Email address: Provide an email address for the superuser. This is optional but recommended.
- Password: Set a password for the superuser. The password will be hidden as you type.
- Password (again): Re-enter the password for confirmation.

Step 4: Confirm Superuser Creation

After entering the details, press Enter. Django will validate the information provided. If everything is valid, you will see a message confirming the creation of the superuser:

```
Superuser created successfully.
```

Step 5: Access Django Admin Interface

Now that you've created the superuser, you can access the Django admin interface by running the Django development server and navigating to the admin URL in your web browser.

Start the development server:

```
python manage.py runserver
```

Then, open your web browser and navigate to `http://127.0.0.1:8000/admin`.

Step 6: Log in as Superuser

At the login page, enter the username and password you specified when creating the superuser. Once logged in, you'll have access to the Django admin interface and can start managing your Django project.

That's it! You've successfully created a superuser in Django. You can now use this superuser account to perform administrative tasks and manage your Django project through the admin interface.

Create a new app

Creating a new app in Django involves several steps, including defining its structure, adding it to the project's settings, and connecting its URLs to the project's main URLs. Let's walk through the process step by step:

Step 1: Create a New App

Open your terminal or command prompt and navigate to the root directory of your Django project.

Run the following command to create a new Django app:

```
python manage.py startapp myapp
```

Replace `myapp` with the desired name of your app.

Step 2: Add the App to Installed Apps

Open the `settings.py` file located in your Django project directory (`myproject/settings.py`).

Find the `INSTALLED_APPS` list and add your app's name to it:

```
INSTALLED_APPS = [
    ...
    'myapp',
]
```

Step 3: Define URL Patterns

Create a new `urls.py` file in your app's directory (`myapp/urls.py`).

Define the URL patterns for your app in this file. Here's an example:

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

urlpatterns = [
    path('', views.index, name='index'),
    # Add more URL patterns as needed
]
```

Step 4: Connect App URLs to Project URLs

In the project's main `urls.py` file (usually located in `myproject/urls.py`), include the URLs of your app using the `include` function:

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('myapp/', include('myapp.urls')),
    # Add more URL patterns as needed
]
```

Step 5: Define Views

Create a `views.py` file in your app's directory (`myapp/views.py`).

Define view functions to handle requests. For example:

```
from django.shortcuts import render
from django.http import HttpResponse

def index(request):
    return HttpResponse("Hello, world. This is my app.")
```

Step 6: Run the Server

Start the Django development server:

```
python manage.py runserver
```

Now you can access your app by navigating to the appropriate URL in your browser. For example, if your app's index view is mapped to `myapp/`, you can access it at `http://127.0.0.1:8000/myapp/`.

That's it! You've successfully created a new app in Django, connected it to the project's settings, and defined URL patterns to access its views. You can now expand your app by adding more views, templates, and functionality as needed.

Define a model

To define a new model in your Django app and make it accessible via the Django admin interface, follow these steps:

Step 1: Define the Model

Open the `models.py` file in your app's directory (`myapp/models.py`). Define your new model class.

For example, let's create a simple model representing a product:

```
from django.db import models

class Product(models.Model):
    name = models.CharField(max_length=100)
    price = models.DecimalField(max_digits=10, decimal_places=2)
    description = models.TextField()

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

Step 2: Make Migrations

Run the following command to generate a migration file based on your new model:

```
python manage.py makemigrations myapp
```

Replace `myapp` with the name of your app.

Step 3: Apply Migrations

Apply the migration to your database:

```
python manage.py migrate
```

Step 4: Register the Model in `admin.py`

Open the `admin.py` file in your app's directory (`myapp/admin.py`). Register your model with the Django admin interface.

```
from django.contrib import admin
from .models import Product

admin.site.register(Product)
```

Step 5: Run the Server

Start the Django development server:

```
python manage.py runserver
```

Step 6: Access Django Admin Interface

Now, navigate to `http://127.0.0.1:8000/admin` in your web browser. Log in using your superuser credentials.

Step 7: Manage Your Model in Admin Interface

You should now see your `Product` model listed in the admin interface. You can add, view, edit, and delete products directly from the browser.

That's it! You've successfully defined a new model in your Django app and made it accessible via the Django admin interface. You can continue to expand your model by adding more fields and functionality as needed.

Admin CRUD

Performing CRUD (Create, Read, Update, Delete) operations on a product model in Django admin is straightforward. Once you've registered your model with the admin interface, you can manage your products directly from the admin dashboard. Let's go through each CRUD operation:

Create (Add) a Product

1. Log in to the Django admin interface at `http://127.0.0.1:8000/admin`.
2. Navigate to the "Products" section or whichever section you've registered your product model under.
3. Click on the "Add Product" button.
4. Fill in the fields for the new product (e.g., name, price, description).
5. Click the "Save" button to create the product.

Read (View) Products

1. Once logged in to the admin interface, you'll see a list of all existing products.
2. You can click on any product to view its details.
3. You can also use search and filter options to find specific products.

Update (Edit) a Product

1. From the list of products, click on the product you want to edit.
2. This will open the product's details page.
3. Click on the "Edit" button to make changes to the product's fields.
4. Update the desired fields.
5. Click the "Save" button to save your changes.

Delete a Product

1. From the list of products, locate the product you want to delete.
2. Next to the product entry, there should be a checkbox or a delete button.
3. Select the checkbox or click the delete button.
4. Confirm the deletion when prompted.

Video

https://drive.google.com/file/d/1lGkidp6cuXxtlvwqlOx5YggzrCXlbvMr/view?usp=sharing

```
# CT-3 Solutions for Object Oriented Programming II and Web Programming
# Course code: CSE 301

# Question 1: Rectangle Class
print("=== Question 1: Rectangle Class ===")

class Rectangle:
    def __init__(self, length, width):
        self.length = length  # integer attribute
        self.width = width    # integer attribute

    def area(self):
        """Returns the area of the rectangle"""
        return self.length * self.width

    def perimeter(self):
        """Returns the perimeter of the rectangle"""
        return 2 * (self.length + self.width)

    def is_square(self):
        """Returns True if it's a square (length == width), otherwise False"""
        return self.length == self.width

# Create an object of Rectangle with length = 5 and width = 5
rect = Rectangle(5, 5)
```

```
# Check if it's a square and print its area and perimeter
print(f"Is square: {rect.is_square()}")
print(f"Area: {rect.area()}")
print(f"Perimeter: {rect.perimeter()}")

print("\n" + "="*50 + "\n")

# Question 2: Book Class
print("=== Question 2: Book Class ===")

class Book:
    def __init__(self, title, author, pages):
        self.title = title      # string attribute
        self.author = author    # string attribute
        self.pages = pages      # integer attribute

    def is_thick(self):
        """Returns True if the book has more than 300 pages, otherwise False"""
        return self.pages > 300

    def summary(self):
        """Returns a string in the format: Title: [title], Author: [author], Pages: [pages]"""
        return f"Title: {self.title}, Author: {self.author}, Pages: {self.pages}"

# Create an object of Book with specified details
book = Book("The Great Gatsby", "F. Scott Fitzgerald", 180)
```

```
# Check if it's a thick book and print its summary
print(f"Is thick book: {book.is_thick()}")
print(f"Summary: {book.summary()}")

print("\n" + "="*50 + "\n")

# Question 3: Triangle Class
print("=== Question 3: Triangle Class ===")

class Triangle:
    def __init__(self, side1, side2, side3):
        self.side1 = side1    # integer attribute
        self.side2 = side2    # integer attribute
        self.side3 = side3    # integer attribute

    def perimeter(self):
        """Returns the perimeter of the triangle"""
        return self.side1 + self.side2 + self.side3

    def is_equilateral(self):
        """Returns True if all sides are equal, otherwise False"""
        return self.side1 == self.side2 == self.side3

    def is_valid(self):
```

```
        """Returns True if the given sides can form a valid triangle (Triangle Inequality Theorem), otherwise False"""
        # Triangle Inequality: sum of any two sides must be greater than the third side
        return (self.side1 + self.side2 > self.side3 and
                self.side2 + self.side3 > self.side1 and
                self.side1 + self.side3 > self.side2)

# Create an object of Triangle with side1 = 3, side2 = 4, and side3 = 5
triangle = Triangle(3, 4, 5)

# Check if it's an equilateral triangle and a valid triangle, then print its perimeter
print(f"Is equilateral triangle: {triangle.is_equilateral()}")
print(f"Is valid triangle: {triangle.is_valid()}")
print(f"Perimeter: {triangle.perimeter()}")

print("\n" + "="*50)
print("All solutions completed successfully!")
```

Extra Demo Questions from Lec 16,17:

Django Questions & Answers - Systematic Coverage of PDF

TOPIC 1: Django Installation & Basic Setup

Questions:

Q1.1: How do you install Django on your system?

Q1.2: What command is used to create a new Django project?

Q1.3: What is the directory structure created when you run django-admin startproject myproject?

Answers:

A1.1: Install Django using pip:

pip install django

A1.2: To create a new Django project:

django-admin startproject myproject

A1.3: The directory structure created is:

myproject/

manage.py

myproject/

__init__.py

settings.py

urls.py

wsgi.py

TOPIC 2: Creating Django Apps

Questions:

Q2.1: How do you create a new app in Django?

Q2.2: What files are created when you create a new Django app?

Q2.3: After creating an app, what configuration step is required in settings.py?

Answers:

A2.1: Create a new app using:

```
cd myproject
python manage.py startapp myapp
```

A2.2: Files created in the app directory:

```
myapp/
    __init__.py
    admin.py
    apps.py
    migrations/
        __init__.py
    models.py
    tests.py
    views.py
```

A2.3: Add the app to INSTALLED_APPS in settings.py:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
```

```
    'myapp',  # Add your app here
]
```

TOPIC 3: URL Patterns and Routing

Questions:

Q3.1: How do you define URL patterns in a Django app?

Q3.2: What is the purpose of the name parameter in URL patterns?

Q3.3: How do you connect app URLs to the main project URLs?

Answers:

A3.1: Define URL patterns in myapp/urls.py:

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

urlpatterns = [
    path('', views.home, name='home'),
    path('about/', views.about, name='about'),
    path('contact/', views.contact, name='contact'),
]
```

A3.2: The name parameter allows you to reference URLs by name in templates and views, making it easier to change URLs later without breaking links.

A3.3: Include app URLs in the main myproject/urls.py:

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
```

```
    path('', include('myapp.urls')),
]
```

TOPIC 4: Django Views

Questions:

Q4.1: What is a view in Django?

Q4.2: How do you create views for different pages?

Q4.3: What is the difference between HttpResponse and render functions?

Answers:

A4.1: Views in Django are Python functions that take a web request and return a web response. They contain the logic for processing requests and returning appropriate responses.

A4.2: Create views in myapp/views.py:

```
from django.shortcuts import render
from django.http import HttpResponse

def home(request):
    return render(request, 'home.html')

def about(request):
    return render(request, 'about.html')

def contact(request):
    return render(request, 'contact.html')
```

A4.3:

- HttpResponse: Returns a simple HTTP response with plain text or HTML
- render: Returns an HTTP response using a template file, allowing you to use Django's template engine

---

TOPIC 5: HTML Templates

Questions:

Q5.1: Where should HTML templates be stored in a Django app?

Q5.2: How do you create navigation links between pages using Django template tags?

Q5.3: Write a complete HTML template for a home page with navigation buttons.

Answers:

A5.1: HTML templates should be stored in the myapp/templates/ directory.

A5.2: Use Django URL template tags:

```
<a href="{% url 'about' %}"><button>About</button></a>
<a href="{% url 'contact' %}"><button>Contact</button></a>
```

A5.3: Complete home.html template:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Home</title>
</head>
<body>
    <h1>Welcome to the Home Page!</h1>
    <a href="{% url 'about' %}"><button>About</button></a>
    <a href="{% url 'contact' %}"><button>Contact</button></a>
</body>
</html>
```

TOPIC 6: Running Django Development Server

Questions:

Q6.1: How do you start the Django development server?

Q6.2: What is the default URL and port for the Django development server?

Q6.3: How can you access different pages in your Django application?

Answers:

A6.1: Start the development server using:

python manage.py runserver

A6.2: The default URL and port is: http://127.0.0.1:8000/

A6.3: Access different pages using:

- Home page: http://127.0.0.1:8000/
- About page: http://127.0.0.1:8000/about/
- Contact page: http://127.0.0.1:8000/contact/

TOPIC 7: Django Superuser Creation

Questions:

Q7.1: What is a superuser in Django?

Q7.2: What command do you run before creating a superuser for the first time?

Q7.3: What are the steps to create a superuser?

Q7.4: How do you access the Django admin interface?

Answers:

A7.1: A superuser in Django is an administrative user account that has access to the Django admin interface and can perform all administrative tasks such as managing users, groups, and site content.

A7.2: Run migrations first:

python manage.py migrate

A7.3: Steps to create a superuser:

1. Navigate to your Django project directory
2. Run the command:

python manage.py createsuperuser

3. Enter the required details:
    - Username
    - Email address (optional but recommended)
    - Password
    - Password confirmation

A7.4: Access the admin interface by:

1. Starting the development server: python manage.py runserver
2. Navigate to: http://127.0.0.1:8000/admin
3. Login with superuser credentials

TOPIC 8: Django Models

Questions:

Q8.1: What is a model in Django?

Q8.2: Create a Product model with the specified fields from the PDF.

Q8.3: What is the purpose of the __str__ method in a model?

Q8.4: What commands are needed after creating a model?

Answers:

A8.1: A model in Django is a Python class that represents a database table. It defines the structure and behavior of the data you want to store.

A8.2: Product model in myapp/models.py:

```
from django.db import models


class Product(models.Model):
```

```
    name = models.CharField(max_length=100)
    price = models.DecimalField(max_digits=10, decimal_places=2)
    description = models.TextField()

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

A8.3: The __str__ method defines how the model instance is displayed as a string, making it more readable in the admin interface and when printing objects.

A8.4: Commands needed after creating a model:

```
python manage.py makemigrations myapp
python manage.py migrate
```

TOPIC 9: Django Admin Registration

Questions:

Q9.1: How do you register a model with the Django admin interface?

Q9.2: What file is used to register models for admin access?

Q9.3: After registering a model, how can you verify it's accessible in admin?

Answers:

A9.1: Register a model in myapp/admin.py:

```
from django.contrib import admin
from .models import Product

admin.site.register(Product)
```

A9.2: The admin.py file in your app directory is used to register models for admin access.

A9.3: To verify model registration:

1. Start the development server

2. Navigate to http://127.0.0.1:8000/admin
3. Login with superuser credentials
4. The Product model should appear in the admin interface

TOPIC 10: Admin CRUD Operations

Questions:

Q10.1: How do you create (add) a new product in Django admin?

Q10.2: How do you view (read) existing products?

Q10.3: How do you update (edit) a product?

Q10.4: How do you delete a product?

Answers:

A10.1: Create a new product:

1. Log in to Django admin interface
2. Navigate to the "Products" section
3. Click "Add Product" button
4. Fill in the required fields (name, price, description)
5. Click "Save" button

A10.2: View existing products:

1. In the admin interface, click on "Products"
2. You'll see a list of all existing products
3. Click on any product to view its details
4. Use search and filter options to find specific products

A10.3: Update a product:

1. From the products list, click on the product you want to edit
2. This opens the product's details page
3. Click the "Change" or edit button

4. Update the desired fields
5. Click "Save" to save changes

A10.4: Delete a product:

1. From the products list, locate the product to delete
2. Select the checkbox next to the product
3. Choose "Delete selected products" from the action dropdown
4. Click "Go" and confirm the deletion
5. Alternatively, click on the product and use the delete button on the detail page

---

TOPIC 11: Database Migrations

Questions:

Q11.1: What are migrations in Django?

Q11.2: What's the difference between makemigrations and migrate commands?

Q11.3: When do you need to run migrations?

Answers:

A11.1: Migrations in Django are Python files that contain instructions for modifying the database schema. They track changes to your models and apply them to the database.

A11.2:

- makemigrations: Creates migration files based on changes to your models
- migrate: Applies the migration files to the database, actually creating/modifying tables

A11.3: Run migrations when:

- You create a new model
- You modify existing model fields
- You're setting up the project for the first time
- Before creating a superuser

TOPIC 12: MVT Architecture

Questions:

Q12.1: What does MVT stand for in Django?

Q12.2: Explain each component of MVT architecture.

Q12.3: How does MVT differ from MVC architecture?

Answers:

A12.1: MVT stands for Model-View-Template architecture.

A12.2: MVT Components:

- **Model**: Defines data structure and database schema
- **View**: Contains business logic and processes requests
- **Template**: Handles presentation layer (HTML rendering)

A12.3: MVT vs MVC:

- **MVT (Django)**: Model-View-Template
- **MVC (Traditional)**: Model-View-Controller
- In Django, the "Controller" functionality is handled by the framework itself, and "View" contains the logic

---

TOPIC 13: Template Directory Structure

Questions:

Q13.1: Where exactly should you create the templates directory?

Q13.2: What happens if Django can't find your templates?

Q13.3: How do you create multiple HTML templates for navigation?

Answers:

A13.1: Create templates directory at: myproject/myapp/templates/

A13.2: Django will raise a TemplateDoesNotExist error if it can't find the specified template.

A13.3: Create multiple templates:

myapp/templates/

├── home.html

├── about.html

└── contact.html

Each template should include navigation using Django URL tags.

TOPIC 14: Error Troubleshooting

Questions:

Q14.1: What should you do if you get "ModuleNotFoundError: No module named 'django'"?

Q14.2: What does "INSTALLED_APPS" error mean and how to fix it?

Q14.3: How to fix "NoReverseMatch" error in templates?

Answers:

A14.1: Install Django: pip install django

A14.2: This means your app is not registered in settings.py. Add your app name to the INSTALLED_APPS list.

A14.3: "NoReverseMatch" occurs when Django can't find a URL pattern with the given name. Check:

- URL name is correct in template
- URL pattern exists in urls.py
- URLs are properly included in main urls.py

TOPIC 15: Complete Step-by-Step Implementation

Questions:

Q15.1: List all commands needed to create a complete Django project from scratch.

Q15.2: What are all the files you need to create manually?

Q15.3: What is the complete workflow for adding a new page?

Answers:

A15.1: Complete command sequence:

```
# 1. Install Django
pip install django

# 2. Create project
django-admin startproject myproject
cd myproject

# 3. Create app
python manage.py startapp myapp

# 4. Run initial migrations
python manage.py migrate

# 5. Create superuser
python manage.py createsuperuser

# 6. After model creation
python manage.py makemigrations myapp
python manage.py migrate

# 7. Run server
python manage.py runserver
```

A15.2: Files to create manually:

1. myapp/urls.py - App URL patterns

2. myapp/templates/ directory
3. Individual HTML templates (home.html, about.html, contact.html)
4. Update myapp/views.py - Add view functions
5. Update myapp/admin.py - Register models
6. Update myproject/settings.py - Add app to INSTALLED_APPS
7. Update myproject/urls.py - Include app URLs

A15.3: Workflow for adding a new page:

1. Create view function in views.py
2. Create HTML template in templates/ directory
3. Add URL pattern in app's urls.py
4. Test the new page by running server

TOPIC 16: Admin Interface Details

Questions:

Q16.1: What information do you need to provide when creating a superuser?

Q16.2: What can you do in the Django admin interface?

Q16.3: How do you access the admin interface and what's the URL?

Answers:

A16.1: Superuser creation requires:

- Username (required)
- Email address (optional but recommended)
- Password (required, will be hidden)
- Password confirmation

A16.2: In Django admin interface you can:

- Manage users and groups
- Perform CRUD operations on registered models

- View, add, edit, and delete model instances
- Use search and filtering features
- Manage site content

A16.3:

- URL: http://127.0.0.1:8000/admin/
- Access after creating superuser and running server
- Login with superuser credentials

TOPIC 17: Model Field Types

Questions:

Q17.1: What are the common Django model field types used in the PDF?

Q17.2: What are the parameters for DecimalField?

Q17.3: What's the difference between CharField and TextField?

Answers:

A17.1: Common field types from PDF:

- CharField: For short text fields
- DecimalField: For decimal numbers (prices)
- TextField: For longer text content

A17.2: DecimalField parameters:

- max_digits=10: Total number of digits
- decimal_places=2: Number of decimal places

A17.3:

- **CharField**: Limited length text, requires max_length parameter
- **TextField**: Unlimited length text, good for descriptions

---

TOPIC 18: Complete Project Structure

Questions:

Q18.1: What is the complete file structure for a Django project with one app?

Q18.2: What are the key configuration files and their purposes?

Q18.3: How do all the components (URLs, views, templates, models) work together?

Answers:

A18.1: Complete project structure:

```
myproject/
 ├── manage.py
 ├── myproject/
|   ├── __init__.py
|   ├── settings.py
|   ├── urls.py
|   └── wsgi.py
└── myapp/
    ├── __init__.py
    ├── admin.py
    ├── apps.py
    ├── migrations/
   |   └── __init__.py
    ├── models.py
    ├── tests.py
    ├── views.py
    ├── urls.py (create manually)
   └── templates/
        ├── home.html
```

├── about.html

└── contact.html

A18.2: Key configuration files:

- settings.py: Contains all project settings and configurations
- urls.py (main): Main URL routing configuration
- urls.py (app): App-specific URL patterns
- models.py: Database model definitions
- views.py: View functions and logic
- admin.py: Admin interface registration

A18.3: Component interaction:

1. **Request Flow**: URL → View → Template → Response
2. **URL Routing**: Main urls.py includes app urls.py
3. **Views**: Process requests and return responses using templates
4. **Models**: Define data structure and interact with database
5. **Templates**: Render HTML with dynamic content
6. **Admin**: Provides interface for model CRUD operations

ADDITIONAL PROBABLE EXAM QUESTIONS

TOPIC 19: Practical Implementation Questions

Q19.1: Write the complete code to create a Django project with navigation between 3 pages.

Q19.2: Create a Student model and register it in admin with fields: name, email, age, course.

Q19.3: What are the exact steps to see your model in Django admin interface?

Answers:

A19.1: Complete implementation:

Step 1: Create project and app

django-admin startproject school

```
cd school
python manage.py startapp students
```

Step 2: Add to settings.py

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'students',
]
```

Step 3: Create students/urls.py

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

urlpatterns = [
    path('', views.home, name='home'),
    path('about/', views.about, name='about'),
    path('contact/', views.contact, name='contact'),
]
```

Step 4: Update main urls.py

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
```

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

Step 5: Create views

```
from django.shortcuts import render

def home(request):
    return render(request, 'home.html')

def about(request):
    return render(request, 'about.html')

def contact(request):
    return render(request, 'contact.html')
```

A19.2: Student model:

```
# models.py
from django.db import models

class Student(models.Model):
    name = models.CharField(max_length=100)
    email = models.EmailField()
    age = models.IntegerField()
    course = models.CharField(max_length=50)

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

```
# admin.py

from django.contrib import admin

from .models import Student


admin.site.register(Student)
```

A19.3: Steps to see model in admin:

1. Create the model in models.py
2. Run python manage.py makemigrations
3. Run python manage.py migrate
4. Register model in admin.py
5. Create superuser: python manage.py createsuperuser
6. Run server: python manage.py runserver
7. Visit http://127.0.0.1:8000/admin/
8. Login with superuser credentials