Module 9) Python DB and Framework

Introduction to embedding HTML within Python using web frameworks like Django or Flask.

Embedding HTML in Python is done using web frameworks like Flask and Django that follow a template-based approach. Python handles backend logic, while HTML is written in separate template files and rendered dynamically using template engines such Django Templates

Generating dynamic HTML content using Django templates.

Django generates dynamic HTML using its template system, where HTML files contain special template tags and variables. Data is passed from views to templates, and the template engine replaces variables and executes tags to render content dynamically at runtime.

Integrating CSS with Django templates.

CSS is integrated into Django templates by placing CSS files in the static directory and linking them using the {% load static %} tag.

How to serve static files (like CSS, JavaScript) in Django.

Django serves static files by configuring STATIC_URL and STATICFILES_DIRS in settings.py. Static files are stored in a static folder and accessed in templates using {% load static %} and {% static %}.

Using JavaScript for client-side interactivity in Django templates.

JavaScript is used in Django templates by linking JS files as static assets or embedding scripts inside HTML templates. It runs in the browser to handle events, update the UI, and interact with the server, enabling dynamic client-side interactivity without reloading pages.

Linking external or internal JavaScript files in Django.

JavaScript files are linked in Django by placing them in the static directory and loading them in templates using {% load static %}. Eg <script src="{% static 'js/script.js' %}"></script>

Overview of Django: Web development framework.

Django is a high-level Python web framework used to build secure and scalable web applications. It follows the Model-View-Template (MVT) pattern and includes built-in features for routing, databases, authentication, and administration.

Advantages of Django (e.g., scalability, security).

Django offers high scalability through its modular architecture and efficient request handling. It provides strong security with built-in protection against common threats like SQL injection, XSS, and CSRF, along with rapid development using reusable components.

Django vs. Flask comparison: Which to choose and why.

Django is a full-featured framework with built-in tools for authentication, database handling, and admin interfaces, making it suitable for large, complex projects. Flask is lightweight and flexible, giving more control and minimal structure, which works well for small apps or projects where you want to choose your components.

Understanding the importance of a virtual environment in Python projects.

A virtual environment isolates a Python project's dependencies from the system Python, preventing version conflicts between projects. It ensures that each project has its own packages, making development more reliable, portable, and easier to manage.

Using venv or virtualenv to create isolated environments.

You can create an isolated Python environment using venv or virtualenv.
With venv, run python -m venv env_name to create the environment and source env_name\Scripts\activate (Windows) to activate it.

Steps to create a Django project and individual apps within the project.

Create a virtual environment and activate it. Install Django: pip install django. Create a Django project: django-admin startproject project_name. Navigate into the project folder: cd project_name. Create an app: python manage.py startapp app_name. Register the app in INSTALLED_APPS in settings.py. Run the server: python manage.py runserver to verify setup

Understanding the role of manage.py, urls.py, and views.py.

manage.py is used to manage the project via commands. urls.py routes URLs to the right views. views.py handles requests and returns responses or renders templates.

Django's MVT (Model-View-Template) architecture and how it handles request-response cycles.

Django's MVT architecture separates an application into Models, Views, and Templates. Models handle data and database interactions, Views process requests and return responses, and Templates define the HTML presentation.

Introduction to Django's built-in admin panel.

Django’s admin panel is an automatic interface for managing models and database data. It allows adding, editing, and deleting records without extra coding. It’s customizable and secured with authentication.

Customizing the Django admin interface to manage database records.

The Django admin can be customized by registering models with admin.py and using ModelAdmin options. You can change list displays, filters, search fields, and form layouts for easier data management.

Setting up URL patterns in urls.py for routing requests to views.

In urls.py, you define URL patterns that map paths to view functions or classes. Django uses these patterns to route incoming requests to the correct view for processing.

Integrating templates with views to render dynamic HTML content.

Views pass data to templates, which use Django’s template language to render dynamic HTML. The view calls render() to combine data with the template and return the response.

Using JavaScript for front-end form validation.

JavaScript validates form inputs in the browser before submission, checking formats, required fields, or value ranges. This improves user experience and reduces server load.

Connecting Django to a database (SQLite or MySQL).

Django connects to a database via settings.py by configuring the DATABASES setting. It supports SQLite by default and can connect to MySQL by installing the appropriate driver and updating settings.

Using the Django ORM for database queries.

Django ORM allows interacting with the database using Python objects instead of SQL. Models represent tables, and queries like .filter(), .get(), or .create() handle data operations.

Understanding Django’s ORM and how QuerySets are used to interact with the database.

Django’s ORM maps models to database tables, letting you work with data using Python. QuerySets retrieve, filter, and manipulate records efficiently without writing SQL.

Using Django’s built-in form handling.

Django forms handle input validation, rendering, and processing automatically. You define forms in forms.py, use them in views, and render them in templates for secure data handling.

Implementing Django’s authentication system (sign up, login, logout, password management).

Django’s authentication system manages users with built-in models and views. It provides signup, login, logout, and password management features through forms and authentication functions.

Using AJAX for making asynchronous requests to the server without reloading the page.

AJAX sends asynchronous requests from JavaScript to Django views, allowing data exchange without full page reloads. Responses can update the page dynamically using JSON or HTML.

Techniques for customizing the Django admin panel.

The Django admin can be customized by registering models with ModelAdmin, adjusting list displays, filters, search fields, and adding custom actions or inline models.

Introduction to integrating payment gateways (like Paytm) in Django projects.

Payment gateways like Paytm can be integrated in Django by using their SDK or APIs. Views handle payment requests and responses, while templates manage the payment forms and redirects.

Steps to push a Django project to GitHub.

Initialize a Git repository in the project folder, add files with git add ., commit with git commit -m "message", add the GitHub remote, and push using git push origin main.

Introduction to deploying Django projects to live servers like PythonAnywhere.

Django projects can be deployed on servers like PythonAnywhere by uploading code, configuring the virtual environment, setting database and static files, and pointing the web app to the project’s WSGI file.

Setting up social login options (Google, Facebook, GitHub) in Django using OAuth2.

Social logins in Django use OAuth2 via packages like django-allauth. You configure provider credentials, add URLs and views, and let users authenticate through Google, Facebook, or GitHub.

Integrating Google Maps API into Django projects.

Google Maps API is integrated by including the API script in templates and using JavaScript to render maps. Django views provide dynamic location data to display on the map.