DJANGO FLOW

Python Revision:

- Python functions
- OOPs concepts
- Packages and modules

Web Technology:

- HTML (basics, p, head, div, a, table, form, nav)
- CSS (inline, internal, external)

Introduction - I:

- Start with systems (windows, linux, etc)
- Softwares ? different softwares (ex)
- Types of softwares (standalone, Client-Server (2-tier, 3-tier))
- Flow of Requests and Response as per 2-tier & 3-tier.
- Examples of all.

Introduction - II:

- Framework
- Web framework
- Framework Architectures
    - MVC (Model, View, Controller)
    - MVT (Model, View, Templates)

Starting with Django:

- System environment and Virtual environment.
- Creating Virtual Environment
- Installing Django
- Create a Project.
- Understanding Project directory
- Running server.

Apps Creation:

- Creating apps.
- Registering apps in settings.py
- Understanding App Directory.
- Creating multiple apps and registering.

Views & urls:

- What are views in detail and Writing our first view.
- Working on project urls.
- Creating views using HttpResponse.
- Explaining all the functions and variables present in urls.py
- Explaining the flow from urls view.
- Assignment (to create different apps and writing views and urls)

Back to views:

- Why view?
- Working on data using fstring (HttpResponse).
- Passing some html code ….
- Creating multiple apps and multiple views in each app
- Importing views using alias names in import statements.

App Urls:

- Understanding the need of app urls.
- Creating App urls.py
- Creating routes
- Connecting app url to project url [include()]

Templates (app level):

- Introduction to templates.
- Create an html file.
- How to connect to a view.
- Rendering template
- Replacing HttpResponse() class to render() function.
- Creating multiple templates (html files).
- Understanding need of Template inheritance

DTL (Django Template Language):

- Introduction to DTL
- Basic Tag in DTL: For Inheritance (block, extends)
- Understanding Context Dictionary and passing it to the template.
- Rendering the data from the context using DTL/Jinja Tags {{}}
- Rendering Collection (List) ⇒ **DTL for loop {% for var in collection %}**
- Rendering Dictionary (DTL syntax for retreiving dictionary value)
- Understanding {% include %} tag and implementing it.

Dynamic URL Routings:

- Making a clickable text and passing the static route in **href="" .**
- Understanding the argument "**name=**" in our **path()** function.
- Using the **name** argument of the url to call the route on clicking.
- Implementing {% url %} tag. **{% url 'name' %}**

Passing argument through url:

- Trying to fetch individual data from our Backend.
- Understanding to pass any data from url to backend function.
- Syntax to pass the data through url **/<datatype:var_name>**
- Passing argument using {% url %} tag. **{% url 'name' argument %}**

Static file(app level):

- App level static file.
- What are static files used for.
- How to create static files.
- Different folders to create.. (css, js, media)
- Connecting static files (load, static) {% load static %}, {% static 'path' %}

HTML Forms:

- Creating Forms in HTML
- Getting data from the frontend.
- GET / POST requests
- {% csrf_token %}
- Search Functionality

Template (Project level):

- Creating ‘templates’ folder at project level.
- Configuration in setting.py
- What is **BASE_DIR.** (Variable which stores the path of project Base directory)
- Creating the same files as app level.
- Rendering the templates.
- Inheritance from project level to app level templates.

Static files (project level):

- Configuration.
- Some explanation as above.

Project:

- 1 project using templates and static files. Using only a list of dictionaries.
- Ex: Library, Articles, Portfolio, Cloning Website (KFC, Dominos)

Models:

- What are models (Intro and where we create)
- Need of models (eg. of previous project)
- How to create model class
- Model Fields.
- Learning Migrations (Imp).

Admin Panel:

- Creating a Super User.
- Logging in to the Admin page using the credentials created.
- Registering Models into admin panel.
- Doing CRUD operations using the Admin panel.

CRUD Operation:

- Understanding CRUD operations using Django Shell.
- Implementing the same into our views.
- Passing GET/POST requests through forms.
- Fetching the data using the same
- {% csrf_token %}
- redirect() function

Django Forms:

- Creating Django forms in forms.py (forms.Form)
- Importing forms in views…
- Manual rendering of form fields in templates {{ form.label }}
- Rendering using for loop
- Form field arguments (label, required, initial, disabled)
- Posting data and Form validation.
- Model form (forms.ModelForm), Meta, Widgets etc…

Mini Project:

- To-do list application.
- Using all the CRUD operations.

User Auth:

- Basics about Authentication and Authorization.
- Understanding User model.
- User Authentication system using Django admin.
- User Login & Signup system
- User Logout
- Authorization (login_required() decorator) or request.user.is_authenticated
- User Foreign Key relationship
- Ex: Task.objects.filter(user=request.user)
- **(Advanced)**
    - UserCreationForm & UserLoginForm
    - Custom User Model (AbstractUser)
    - Custom User Creation Form.

APIs (Django REST Framework):

- Introduction to API
- Introduction to **Django REST Framework** (DRF)
- Installation and registering.

Serializers & Serialization:

- Understanding the **JSON** package (dumps(), loads())
- Understanding Serialization.
- **Serializers** class in DRF.
- Creating views to serialize the data.
- **JSONRenderer()** class.
- Displaying data using **HttpResponse()** class.
- Fetching data using **requests** module.
- Serializer Fields.

API Responses

- Displaying Through HttpResponse() class
- JsonResponse() class
- Response() class in DRF.
- Along with understanding api_view() decorator

CRUD Operations in API:

- Creating views for all types of operations.
- Creating endpoints
- Postman.