❖ MODULE 9) PYTHON DB AND FRAMEWORK

➢ HTML in Python:

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

Ans:-

- When you build websites with **Django** or **Flask**, you don't write HTML directly inside Python code.

- Instead, you **separate HTML files** and let Python send data to those HTML files.

How it works:

1. Templates

- Both Django and Flask use **templates** (HTML files with placeholders).

Example placeholder:

```
<h1>Hello {{ name }}</h1>
```

2. Python sends data to the template

- You write Python code in your views and send data to the HTML template.

Django Example

views.py

```
def home(request):
   return render(request, "home.html", {"name": "Hensi"})
```

home.html

```
<h1>Welcome {{ name }}</h1>
```

Flask Example

app.py

```
@app.route("/")
def home():
   return render_template("home.html", name="Hensi")
```

home.html

```
<h1>Welcome {{ name }}</h1>
```

2.Generating dynamic HTML content using Django templates.

Ans:-

- Django allows you to create **HTML pages that change automatically** based on the data you send from your Python code (views).

This is done using **Django Templates**.

1. Use {{ }} to display dynamic values

- Your HTML file can show any value sent from views.

views.py

```
def home(request):
   return render(request, "home.html", {"username": "Hensi"})
```

home.html

```
<h1>Welcome, {{ username }}</h1>
```

Output:
Welcome, Hensi.

2. Use {% %} for logic (loops, conditions)

- Django templates allow loops and if-else.

Loop Example:

views.py

```
def product_list(request):
   products = ["Cream", "Lipstick", "Shampoo"]
   return render(request, "products.html", {"items": products})
```

products.html

```
<ul>
{% for p in items %}
   <li>{{ p }}</li>
{% endfor %}
</ul>
```

If Condition Example

```
{% if items %}
   <p>Total products: {{ items|length }}</p>
{% else %}
   <p>No products available</p>
{% endif %}
```

3. Use Filters to modify data

```
<h2>{{ username|upper }}</h2>
```

4. Use Template Inheritance (Optional but important)

base.html

```
<html>
<body>
   {% block content %}{% endblock %}
</body>
</html>
```

home.html

```
{% extends "base.html" %}
{% block content %}
<h1>Hello {{ username }}</h1>
{% endblock %}
```

➢ CSS in Python Theory:

1.Integrating CSS with Django templates.

Ans:-

- To use CSS in your Django project, you place your CSS files in the **static folder** and then load them inside your HTML templates.

1. Create a static folder

Your structure should look like:

```
project/
│
├─ app/
│   ├─ templates/
│   ├─ static/
│      └─ css/
│         └─ style.css
```

2. Write your CSS in style.css

```
body {
   background-color: lightpink;
}
h1 {
   color: purple;
}
```

3. Load static files in your template

At the top of your HTML template, write:

```
{% load static %}
```

4. Link the CSS file

Inside the <head> section:

```
<link rel="stylesheet"  href="{% static 'css/style.css' %}">
```

Final home.html Example

```
{% load static %}
<!DOCTYPE html>
<html>
<head>
   <link rel="stylesheet" href="{% static 'css/style.css' %}">
</head>
<body>

<h1>Welcome to Django</h1>

</body>
</html>
```

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

Ans:-

- Static files are used to style and add behavior to your website (CSS, JS, images).
- Django keeps static files separate from Python code for better organization.
- You create a **static folder** in your project or apps, and Django collects all static files from those folders.
- In HTML, you load the static tag and use {% static %} so Django can find and serve these files correctly during development and production.

How to serve static files in Django (CSS, JS, Images)

1. Create a folder named static in your app

Example:

```
myapp/
  static/
    myapp/
      style.css
```

2. Add this in settings.py

```
STATIC_URL = '/static/'
STATICFILES_DIRS = [BASE_DIR / "static"]
```

3. Load static in your HTML

At the top of your HTML:

```
{% load static %}
```

4. Use static files in HTML

```
<link rel="stylesheet" href="{% static 'myapp/style.css' %}">
```

➢ JavaScript with Python:

1.Using JavaScript for client-side interactivity in Django templates.

Ans:-

- You can use **JavaScript inside your Django HTML templates** to make your webpage interactive (like button clicks, form validation, sliders, alerts).

- Django sends data from the backend → your HTML template → then JavaScript runs in the browser.

simple Example:

Template:

```
{% load static %}
<!DOCTYPE html>
<html>
<head>
  <script src="{% static 'js/main.js' %}"></script>
</head>
<body>
  <h2>Hello, {{ name }}</h2>

  <button onclick="showMsg()">Click me</button>

  <p id="info"></p>

  <script>
    function showMsg() {
      document.getElementById("info").innerHTML = "Welcome " + "{{ name }}";
    }
  </script>
</body>
</html>
```

main.js (inside static/js/)

```
console.log("JS file loaded!");
```

2.Linking external or internal JavaScript files in Django.

Ans:-

1. Internal JavaScript (Inside the Template):

- You can write JavaScript directly inside your HTML file using the <script> tag.

Example:

```
<script>
   alert("This is internal JavaScript!");
</script>
```

Use this for small scripts.

2. External JavaScript File (Recommended):

- This is the correct way when you have bigger scripts.

Step 1: Create static folder

Inside your app:

```
your_app/
   static/
      your_app/
         js/
            main.js
```

Step 2: Put JavaScript in main.js

```
console.log("External JS file loaded!");
```

Step 3: Load static in template

```
{% load static %}
```

Step 4: Link the JS file

Place this before closing </body> tag:

```
<script src="{% static 'your_app/js/main.js' %}"></script>
```

➢ Django Introduction:

1.Overview of Django: Web development framework.

Ans:-

1. **Django is a high-level Python web framework** used to build secure and scalable web applications.

2. It follows the **Model–View–Template (MVT)** architecture, which separates data (Model), business logic (View), and user interface (Template).

3. Django provides an **Object Relational Mapper (ORM)** that allows developers to interact with the database using Python instead of SQL queries.

4. It includes a powerful **built-in admin panel**, which helps manage application data without creating a separate dashboard.

5. Django has strong **security features** that protect applications from SQL injection, XSS, CSRF, clickjacking, and other common attacks.

6. The framework supports **URL routing**, which maps URLs to specific views in a clean and organized way.

7. Django allows **rapid development**, enabling developers to build applications faster due to reusable components and built-in tools.

8. It provides a **templating engine** for creating dynamic web pages using HTML mixed with Django template tags.

9. Django is **highly scalable**, making it suitable for both small projects and large applications with millions of users.

10. It supports **multiple databases**, such as MySQL, PostgreSQL, SQLite, and more, without changing much code.

11. Django includes **middleware support**, which helps process requests and responses globally (e.g., authentication, sessions, security).

12. It has **excellent documentation** and a large community, making learning and troubleshooting easier.

13. Django is used by major companies like Instagram, Pinterest, Mozilla, and many more, proving its reliability and performance.

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

Ans:-

1. **Rapid Development:**
Django allows developers to build applications quickly due to its built-in tools and structured design.

2.**MVT Architecture:**
The Model–View–Template pattern helps keep the project organized and easy to maintain.

3.**Strong Security:**
Django provides protection against common attacks like SQL injection, XSS, CSRF, and clickjacking.

4.**Built-in Admin Panel:**
It automatically creates an admin interface to manage database content without extra coding.

5.**Powerful ORM (Object Relational Mapper):**
Developers can interact with the database using Python code instead of writing SQL queries.

6.**Scalability:**
Django can handle high traffic and large applications, making it suitable for enterprise-level projects.

7.**Reusable Components:**
Django encourages code reusability through apps, which reduces development time.

8.**Excellent Documentation:**
It has one of the best and most detailed documentation libraries, making learning and debugging easier.

9.**Cross-Platform Support:**
Django works on Windows, macOS, Linux, and many server environments.

10.**Large Community Support:**
A strong community provides plugins, tutorials, and third-party packages for various needs.

11.**Versatile Framework:**
Django can be used to build all types of applications: e-commerce, social media, blogs, APIs,and more.

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

Ans:-

Django vs Flask (Short Comparison):

- ✓ **Django:**

- **Django** is a full-stack framework with many built-in features like admin panel, authentication, and ORM.
- It follows the MVT architecture and is best for large, complex projects.

- ✓ **Flask:**

- **Flask** is a lightweight, micro-framework with minimal built-in features.
- It gives more flexibility and is ideal for small to medium projects or APIs.

Which to Choose and Why?

- Choose **Django** when you need fast development, built-in tools, and a structured project for big applications.

- Choose **Flask** when you need simplicity, full control, and flexibility for smaller or custom projects.

- **Virtual Environment:**

1.Understanding the importance of a virtual environment in Python projects.

Ans:-

- A **virtual environment** is an isolated workspace that allows you to install Python packages for one project without affecting other projects or the system Python.
- It keeps each project's dependencies separate and organized.

Why Is It Important?

1. **Avoids Version Conflicts:**

- Different projects may require different versions of the same package. A virtual environment keeps each project's packages separate so they don't clash.

2. **Clean and Organized Projects:**

- Each project gets its own environment with only the libraries it needs, making the setup clean and manageable.

3. **Safe Experimentation:**

- You can install or test new libraries without risking your system-wide Python setup.

4. **Easier Deployment:**

- Virtual environments help ensure that the project runs the same way on any computer or server by using the same package versions.

5. **Best Practice in Python Development:**

- Almost all professional Python frameworks (like Django, Flask) recommend using virtual environments for reliability and clean development.

2. Using venv or virtualenv to create isolated environments.

Ans:-

1. Using venv:

- venv is a built-in Python module (no installation needed).
- It creates a separate folder where Python and all packages for that project are stored.

How to Create a venv Environment:

```
python -m venv myenv
```

Activate the Environment

- **Windows:**
- myenv\Scripts\activate

Install Packages:

```
pip install package_name
```

2. Using virtualenv

- virtualenv is an external tool that provides more features and works with older Python versions.You need to install it first.

Install virtualenv:

```
pip install virtualenv
```

Create Environment:

```
virtualenv myenv
```

Activate Environment:

Same as venv:

- Windows: myenv\Scripts\activate

➢ Project and App Creation:

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

Ans:-

Steps to Create a Django Project and Individual Apps:

1. Install Django:

pip install django

2. Create a New Django Project:

django-admin startproject myproject

3. Go Inside the Project Folder:

cd myproject

4. Run the Development Server (Optional Check):

python manage.py runserver

5. Create an App Inside the Project:

python manage.py startapp myapp

6. Add the App to settings.py:

Open myproject/settings.py → find INSTALLED_APPS → add:

'myapp',

7. Create Views, URLs, and Templates:

- Write logic in myapp/views.py
- Create app-level URLs in myapp/urls.py
- Link them in project urls.py

8. Apply Migrations:

python manage.py migrate

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

Ans:-

1. manage.py:

- A command-line tool automatically created in every Django project.
- It helps run important commands like starting the server, creating apps, applying migrations, and managing the project.
- Example: python manage.py runserver

2. urls.py:

- Controls how URLs are mapped to specific views.
- Acts like a “traffic controller” that decides which view should run when a user opens a particular URL.
- Example: mapping /home to a home page view.

3. views.py:

- Contains the functions or classes that handle the request and return a response.
- It connects the database (models) with the templates (HTML pages).
- Example: returning an HTML page or sending data to the browser.

➢ MVT Pattern Architecture:

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

Ans:-

Django's MVT Architecture and Request–Response Cycle

1. MVT Architecture

Django follows the **Model–View–Template (MVT)** architecture, which separates the application into three main components:

• Model

- Represents the **database layer**.
- Defines the structure of data using Python classes.
- Handles CRUD operations (create, read, update, delete).

• View

- Contains **business logic**.
- Receives the request from the user, processes data, interacts with the model, and returns a response.

• Template

- Handles the **presentation layer**.
- Displays data using HTML mixed with Django Template Language (DTL).

2. How Django Handles the Request–Response Cycle:

1. **User sends a request** through the browser (e.g., www.example.com/home).
2. The request reaches **urls.py**, which decides which view should handle this URL.
3. The selected function/class in **views.py** receives the request.
4. The view may interact with the **Model** to fetch or update data from the database.
5. The view passes the processed data to a **Template** for rendering.
6. The Template generates an **HTML response**.
7. Django sends the final HTML back to the user's browser.

- **Django Admin Panel:**

1.Introduction to Django’s built-in admin panel.

Ans:-

Ddjango provides a **powerful built-in admin panel** that allows developers to manage the application's data without writing extra code.

It is automatically created when a new project is set up and is one of Django’s strongest features.

The admin panel is connected to the database through Django’s models.

After registering your models in admin.py, the admin interface lets you **add, edit, delete, and view** data easily.

It also includes built-in authentication, search, filters, and permissions to control user access.

To use it, developers create a superuser and log in at /admin/.

The admin dashboard provides a clean, ready-made interface for managing all application data during development or production.

2. Customizing the Django admin interface to manage database records.

Ans:-

- Django allows developers to **customize the built-in admin interface** to make database management easier and more user-friendly.

- By modifying the admin.py file, you can control how models appear in the admin panel.

Common customizations include:

- **Displaying selected fields** in the list view using list_display.

- **Adding search functionality** with search_fields.

- **Filtering records** using list_filter.

- **Organizing fields** into sections with fieldsets.

- **Custom forms** to validate or format data.

- **Registering models with Admin classes** to fully control their appearance and behavior.

These customizations make the admin panel more efficient for managing large datasets, improving usability, and tailoring the interface to project requirements.

➢ URL Patterns and Template Integration :

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

Ans:-

- In Django, the urls.py file is used to define **URL patterns** that connect specific URLs to corresponding view functions or classes.

- When a user sends a request, Django checks the URL pattern and routes the request to the correct view.

A URL pattern typically includes:

- The **path** (URL address)
- The **view** that should handle the request
- An optional **name** for easy referencing in templates

Example:

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

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

In this example, when the user visits /home/, Django sends the request to the home_view function defined in views.py.

This routing system ensures that each part of the website is properly connected to the right logic.

2.Integrating templates with views to render dynamic HTML content.

Ans:-

- django views collect data (from database or logic) and send it to **HTML templates**, which show that data on the webpage.

Short Theory:

- Views use render() to load an HTML file.
- We pass data as a dictionary to the template.
- Template shows the data using {{ }}.

Example:

views.py

```
from django.shortcuts import render

def home(request):
    data = {"name": "Hensi", "course": "Python & Django"}
    return render(request, "home.html", data)
```

home.html

```
<h2>Welcome, {{ name }}</h2>
<p>You are learning {{ course }}</p>
```

➢ Form Validation using JavaScript:

1.Using JavaScript for front-end form validation.

Ans:-

1. Introduction

Front-end form validation is the process of checking user input on the client-side (browser) before the form is submitted to the server. JavaScript is commonly used for this purpose because it can instantly detect errors and display messages without reloading the page.

2. What is JavaScript Form Validation?

JavaScript form validation is the use of JavaScript code to ensure that the user enters valid and complete data in the form fields.
It helps prevent incorrect, empty, or insecure input.

Example checks done by validation:

- Required fields must not be empty
- Email must follow correct pattern
- Password must meet security rules
- Phone number must contain digits only

3. Types of Front-End Validation

(A) Client-Side Validation

Validation performed directly in the browser using JavaScript.
✔ Fast
✔ No server load
✔ Better user experience

(B) HTML5 Built-in Validation

Uses attributes like required, pattern, maxlength, etc.
But JavaScript gives **more control and custom messages**.

4. Why JavaScript is Important for Form Validation?

- Prevents invalid data submissions
- Reduces server requests
- Improves user experience
- Gives instant feedback
- Helps maintain clean and correct data

5. Common JavaScript Validation Methods:

1. Checking Empty Fields

```
if (username === "") {
   alert("Username is required");
}
```

2. Email Pattern Checking

```
let emailPattern = /^[^ ]+@[^ ]+\.[a-z]{2,3}$/;
if (!email.match(emailPattern)) {
   alert("Enter a valid email address");
}
```

3. Password Length Check

```
if (password.length < 6) {
   alert("Password must be at least 6 characters");
}
```

4. Number Validation

```
if (isNaN(phone)) {
   alert("Phone number must contain digits only");
}
```

➢ Django Database Connectivity (MySQL or SQLite) :

1.Connecting Django to a database (SQLite or MySQL).

Ans:-

- Django is a powerful Python web framework that provides built-in support for databases.

- Every Django project contains a settings.py file where you configure which database your project will use.

Django supports multiple databases like:

- SQLite (default)
- MySQL
- PostgreSQL
- Oracle

2. Connecting Django to SQLite (Default):

What is SQLite?

- SQLite is a lightweight, file-based database that comes preconfigured in Django.
  It is ideal for beginners, small projects, and development/testing.

Default Django SQLite Configuration:

When you create a new Django project, the settings.py automatically contains:

```
DATABASES = {
   'default': {
      'ENGINE': 'django.db.backends.sqlite3',
      'NAME': BASE_DIR / "db.sqlite3",
   }
}
```

3. Connecting Django to MySQL:

What is MySQL?

- MySQL is a powerful relational database used in professional, large-scale applications.

Steps to Connect Django with MySQL

Step 1: Install MySQL Client

Install the connector using pip:

pip install mysqlclient

(or)

pip install pymysql

Step 2: Create a MySQL Database

Example:

CREATE DATABASE myprojectdb;

Step 3: Update settings.py

Replace the database section with:

```
DATABASES = {
   'default': {
      'ENGINE': 'django.db.backends.mysql',
      'NAME': 'myprojectdb',
      'USER': 'root',
      'PASSWORD': 'your_password',
      'HOST': 'localhost',
      'PORT': '3306',
   }
}
```

Step 4: Migrate tables

Run:

python manage.py migrate

This creates all Django tables inside the MySQL database.

2. Using the Django ORM for database queries.

Ans:-

- Django ORM (Object Relational Mapping) allows us to interact with the database using **Python code instead of SQL**.

- It helps to **insert, update, delete, and select** data easily through Django models.

1. Insert Data:

```
User.objects.create(name="Hensi", email="abc@gmail.com")
```

2. Select / Read Data:

```
users = User.objects.all()       # all data
user = User.objects.get(id=1)     # single record
users = User.objects.filter(name="Hensi")   # with condition
```

3. Update Data:

```
u = User.objects.get(id=1)
u.name = "New Name"
u.save()
```

4. Delete Data:

```
u = User.objects.get(id=1)
u.delete()
```

- **ORM and QuerySets :**

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

Ans:-

- **Django ORM (Object Relational Mapping)** allows developers to interact with the database using Python code instead of writing SQL queries.
- Each Django model represents a database table, and its fields represent the table's columns.
- **QuerySets** are Django objects that represent a collection of database records.
- They are used to retrieve, filter, update, and delete data using simple methods like all(), filter(), and get().
- QuerySets are **lazy**, meaning the database is queried only when the data is actually required, which improves performance.

➢ Django Forms and Authentication:

1.Using Django's built-in form handling.

Ans:-

- Django provides a built-in form handling system that makes it easy to create, validate, and process user input securely.

- Forms are defined using Python classes, and Django automatically generates HTML form fields and handles validation.

There are **two** types of forms in Django:

- **Forms (forms.Form)** :–

  Used when form data is not directly linked to a database.

- **ModelForms (forms.ModelForm)** :–

  Used when form data is connected to a Django model and can be saved directly to the database.

Django follows a simple process:-

1. display the form, handle the submitted data using POST, validate the data using is_valid(), and then process or save it.

2. It also provides automatic validation, error handling, and security features like CSRF protection.

Overall, Django's built-in form handling reduces code, improves security, and makes form management clean and efficient.

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

Ans:-

- Django provides a built-in authentication system that makes it easy to manage **user sign up, login, logout, and password management** securely.

Sign Up:-

User registration is usually implemented using Django's User model with a custom form or UserCreationForm.

It handles username and password validation and securely stores passwords using hashing.

Login:-

Django authenticates users using the authenticate() and login() functions.

Once logged in, user session data is maintained automatically.

Logout:-

Users can be logged out using the logout() function, which safely ends the user session.

Password Management:-

Django supports password change and reset features using built-in views and forms.
Passwords are never stored in plain text; they are securely hashed.

Overall, Django's authentication system provides a secure, ready-to-use solution for managing users with minimal code.

➢ CRUD Operations using AJAX :

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

Ans:-

- AJAX (Asynchronous JavaScript and XML) is a technique used in web development to communicate with the server **without refreshing the entire web page**.
- Instead of loading a new page every time data is requested, AJAX updates only the required part of the page.
- This makes websites faster, smoother, and more user-friendly.

How AJAX Works:

1. The user performs an action (like clicking a button).
2. JavaScript sends a request to the server using AJAX.
3. The server processes the request and sends back data.
4. JavaScript receives the data.
5. The webpage is updated dynamically **without reload**.

Why AJAX is Useful:

- Saves time and improves performance.
- Better user experience.
- Reduces server load.
- Allows real-time data updates

Example:

When we search on Google, suggestions appear while typing.

This happens because AJAX keeps sending requests to the server in the background and updates results instantly **without page reload**.

➢ Customizing the Django Admin Panel:

1.Techniques for customizing the Django admin panel.

Ans:-

- The Django admin panel is a powerful built-in tool that allows developers to manage database records easily.
- Django also provides several techniques to customize the admin panel to make it more user-friendly, attractive, and efficient for administrators.

1. Registering Models:

To display database tables in the admin panel, models must be registered using admin.site.register().

This allows us to manage records directly from the admin dashboard.

2. Using ModelAdmin Class:

By creating a **ModelAdmin** class, developers can control how data is displayed.

Common options include:

- list_display :

  – shows selected fields in list view

- search_fields :

  – adds a search bar

- list_filter :

  – adds filters in sidebar

- ordering :

  – sorts records

- list_per_page:

  – controls number of records per page

3. Customizing Forms:

We can customize the admin form layout using:

- fields :

  – to control form field order

- fieldsets :

  – to group fields with headings

- readonly_fields :

  – to make some fields uneditable

4. Adding Custom Actions:

Custom admin actions allow performing operations on multiple selected records, such as approve, delete, or update status in just one click.

5. Using Admin Themes:

Django supports themes and third-party packages like:

- Django Suit
- Django Jet
- Grappelli

Note:These improve design and user experience.

6. Overriding Admin Templates:

Developers can override default HTML templates to fully change layout, design, buttons, and styles of the admin panel.

➢ Payment Integration Using Paytm:

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

Ans:-

- A **payment gateway** is a secure service that allows users to make online payments using options such as debit/credit cards, UPI, wallets, and net banking.

- In Django projects, integrating a payment gateway helps in building real-world applications like e-commerce websites, fee payment systems, booking systems, etc.

Why Payment Gateway Integration is Needed:

- To accept online payments securely
- To provide multiple payment options to users
- To ensure safe transaction processing with encryption
- To automatically verify successful and failed payments

How Payment Gateway Integration Works:

1. The user selects a product/service and clicks “Pay”.
2. Django sends payment details to the payment gateway (like Paytm).
3. The user is redirected to the secure payment page.
4. The payment gateway processes the transaction.
5. After payment, the gateway sends a success or failure response back to Django.
6. Django updates the database and shows confirmation to the user.

Steps Involved in Integration:

- Create a merchant account on the payment gateway (Paytm etc.).
- Install payment gateway SDK or API in Django.
- Configure API keys (Merchant ID, Secret Key).
- Create payment request and redirect user to gateway.
- Handle callback/response after payment.
- Store transaction details in the database.

Security Features:

Payment gateways like Paytm provide:

- Encryption of sensitive data
- Secure transaction processing
- Fraud detection
- OTP and authentication support

➢ GitHub Project Deployment :

1. Steps to push a Django project to GitHub.

Ans:-

- Pushing a Django project to GitHub means uploading your full project from your computer to an online Git repository.

- This helps in backup, version control, teamwork, and sharing projects with others.

1. Install Git:

First, install Git on your system and configure it using:

- git config --global user.name "Your Name"
- git config --global user.email "your email"

2. Initialize Git in the Django Project:

Open the Django project folder and initialize Git:

- git init

This creates a hidden .git folder and starts version control.

3. Create a GitHub Repository:

- Login to GitHub
- Click **New Repository**
- Enter repository name
- Select Public or Private
- Click **Create Repository**

4. Add Project Files to Git:

Add files to staging:

- git add .

Then commit the changes:

- git commit -m "Initial Django project commit"

5. Connect Local Project to GitHub:

Copy the repository link from GitHub and connect using:

- git remote add origin <repository link>

6. Push the Project to GitHub:

Finally upload the project:

- git branch -M main
- git push -u origin main

7. Use .gitignore (Important):

Create a .gitignore file to avoid uploading unnecessary files like:

- venv/
- __pycache__/
- db.sqlite3
- .env

This keeps the project clean and secure.

➢ Live Project Deployment (PythonAnywhere) :

1.Introduction to deploying Django projects to live servers like PythonAnywhere.

Ans:-

- Deploying Django projects to live servers like PythonAnywhere is an essential step in web development.
- It allows developers to publish their applications online, test them in real environments, and make them accessible to users worldwide.
- PythonAnywhere provides a simple, beginner-friendly platform for smooth Django deployment.

Basic Steps for Deployment on PythonAnywhere:

1. **Create an Account:**

   Sign up on PythonAnywhere.

2. **Upload Project:**

   Upload project files or connect GitHub repository.

3. **Create Virtual Environment:**

   Install required Python packages using requirements.txt.

4. **Configure Settings:**

   - Set ALLOWED_HOSTS
   - Configure static files
   - Set database if required

5. **Setup Web App:**

   Select “Django Web App” and link project path.

6. **Run Migrations & Collect Static:**

   Apply database migrations and collect static files.

7. **Reload Server:**

   Reload the web app to make it live.

Benefits of Using PythonAnywhere:

- Very easy for beginners
- No need to manage complex servers
- Free plan available
- Built-in support for Django and databases
- Secure and reliable hosting

➢ Social Authentication:

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

Ans:-

- Social login allows users to sign in to a Django website using their existing accounts like Google, Facebook, or GitHub.
- Instead of creating a new username and password, users can log in securely through trusted platforms.
- This is done using **OAuth2**, which is a secure authorization protocol widely used for authentication in modern web applications.

What is OAuth2?

- OAuth2 is a security protocol that allows third-party applications (like Django websites) to access user information from social platforms **without sharing passwords**.
- It provides access tokens that verify user identity safely.

Why Social Login is Useful:

- Faster and easier login process
- No need to remember extra passwords
- Trusted authentication from companies like Google and Facebook
- Improves user experience and increases signups
- Secure and reliable

Basic Steps to Implement Social Login in Django:

1. **Install Required Libraries:**

   Commonly django-allauth or social-auth-app-django is used.

2. **Add App in Django Settings:**

   Configure installed apps, middleware, and authentication backends.

3. **Create Developer Account:**

   Register app on Google, Facebook, or GitHub developer portals.

4. **Get OAuth Credentials:**

   Each platform provides:
   - Client ID
   - Client Secret
   - Callback/Redirect URL

5. **Configure Credentials in Django:**

   Add keys and redirect URLs in Django settings.

6. **Add Login Buttons on Website:**

   Provide options like “Login with Google / Facebook / GitHub”.

7. **Test Authentication:**

   Login using social account and verify user is authenticated in Django.

Security Features:

- No password sharing
- Secure token-based authentication
- Trusted identity verification
- Protection from unauthorized access

➢ Google Maps API:

1.Integrating Google Maps API into Django projects.

Ans:-

- Integrating the Google Maps API in a Django project allows developers to display interactive maps, show locations, add markers, and provide navigation features directly on a website.

- It is commonly used in applications like travel websites, delivery systems, real estate platforms, and location-tracking applications.

What is Google Maps API?

- The Google Maps API is a service provided by Google that allows web applications to embed and control maps using JavaScript.

- It provides many features like markers, geolocation, directions, and distance calculation.

Why Integrate Google Maps in Django:

- To display location-based information

- To show user or business locations on a map

- To improve user experience with visual navigation

- Useful for real-world applications like tracking and location search

Basic Steps to Integrate Google Maps API in Django:

1. **Create a Google Developer Account:**

   Go to Google Cloud Console and create a project.

2. **Enable Google Maps JavaScript API:**

   Activate the Maps API for the project.

3. **Generate API Key:**

   Google provides a unique API key to access maps securely.

4. **Add API Key to Django Project:**

   Store it in settings or environment variables for security.

5. **Create a Template:**

   Use HTML and JavaScript to load Google Maps and display it on the webpage.

6. **Pass Location Data from Django:**

   Django can send latitude and longitude values to the template.

7. **Display Map:**

   The map loads dynamically with markers and required features.