Unit-2 : Introduction to Web Development Frameworks

2.1 Flask

2.2 Django

2.3 GIS Web Services

Web Development Frameworks

- Web Application Framework or simply Web Framework represents a collection of libraries and modules that enables a web application developer to write applications without having to bother about low-level details such as protocols, thread management etc.

Flask

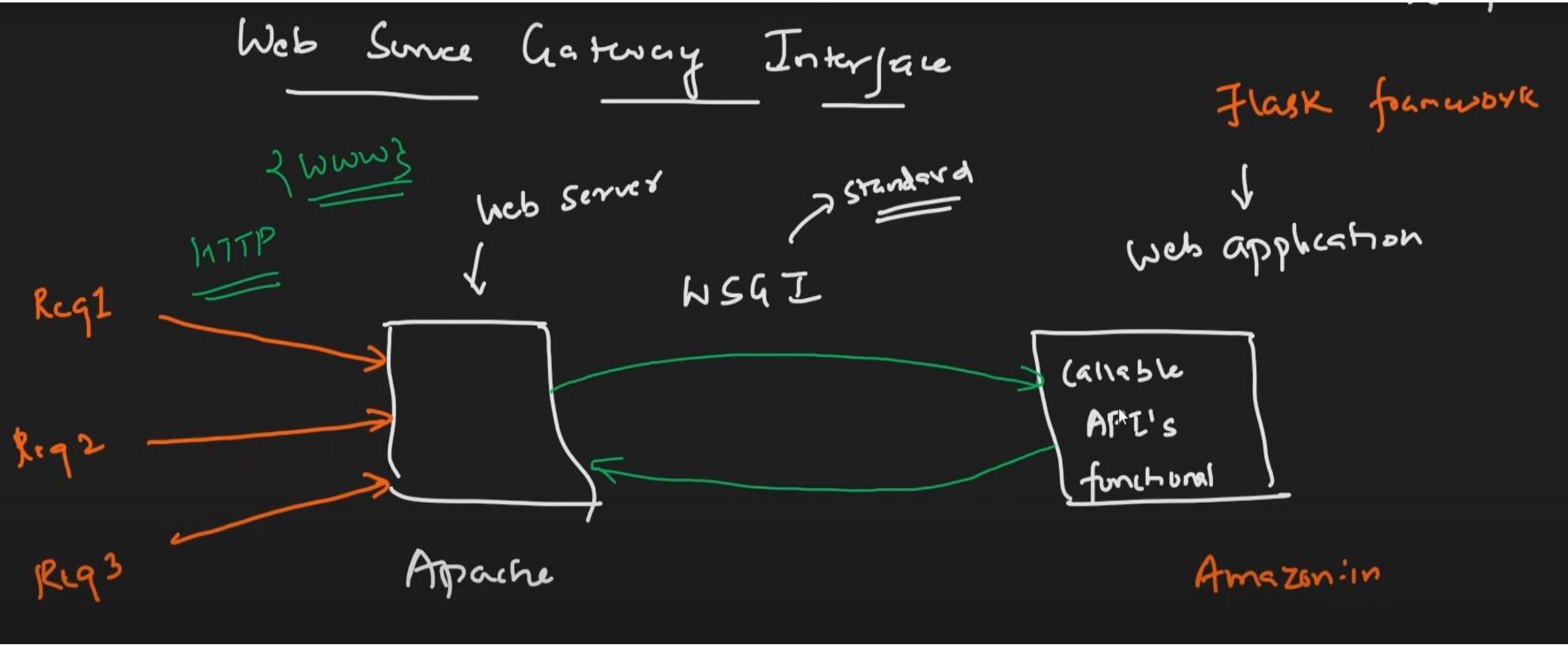
- Flask is a web application framework written in python.
- Armin Ronacher, who leads an international group of enthusiasts named Pocco, develops it.
- Flask is based on Werkzeug WSGI toolkit and Jinja2 template engine.
- Both are Pocco projects.

WSGI

- Web Server Gateway Interface (WSGI) has been adopted as a standard for web application development.
- WSGI is a specification for a universal interface between the web server and the web applications.

Jinja2

- Jinja2 is a popular templating engine for.
- A web templating system combines a template with a certain data source to render dynamic web pages.


Web Server Gateway Interface
{WWW}
HTTP
Web server
standard
WSGI
Flask framework
Web application
Req1
Req2
Req3
Callable
API's
functional
Apache
Amazon.in

- Flask is often referred to as a micro framework.
- It aims to keep the core of an application simple yet extensible.
- Flask does not have built-in abstraction layer for database handling, nor does it have form a validation support.
- Instead, Flask supports the extensions to add such functionality to the application.

Steps (How to install flask)

1. Open a Terminal or Command Prompt.
2. Check if is installed using ` --version`.
3. Install Flask using `pip install Flask`.
4. Verify the installation with `flask --version`.
5. Create a new file (e.g., `app.py`) and add a Flask application code.
6. Run the Flask application with ` app.py`.

In order to test**Flask** installation, type the following code in the editor as **Hello.py**

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
   return 'Hello World’

if __name__ == '__main__':
   app.run()
```

1. Importing Flask:

from flask import Flask

flask: This is the name of the module, and it's written in lowercase.

Flask: This is the name of the class within the flask module, and it's written in CamelCase.

2. Creating an instance of the Flask class:

app = Flask(__name__)

We create an instance of the Flask class and pass `__name__` as an argument.

This is a common pattern in Flask to help Flask determine the root path of the application.

When you run a script directly the __name__ variable is set to "__main__".

3. Defining a route:

@app.route('/')

This decorator tells Flask that the function below it should be executed when someone accesses the root URL ('/') of the web application.

@app.route('/'): This is a decorator in .In Flask, it is used to define a route. The route is the URL path that the function below it should respond to ‘/’.

In this case, it's the root path, which means it will respond when someone accesses the main page of your website (e.g.,http://127.0.0.1:5000/).

4. Defining a function for the route:

def hello_world():

return 'Hello World'

The function `hello_world` is what will be executed when a user accesses the root URL. In this case, it simply returns the string 'Hello World'.

@app.route('/') is saying that the function hello_world() should be executed when the root URL (/) is accessed.

The hello_world() function, when executed, simply returns the string 'Hello, World!'.

So, when someone goes to the main page of your Flask web application, the hello_world() function is called, and the text 'Hello, World!' is displayed in their web browser.

5. Running the application:

```
if __name__ == '__main__':
app.run()
```

This block ensures that the Flask web server is started only if the script is executed directly, not if it's imported as a module.

The `app.run()` starts the development server, allowing you to access the web application locally.

The app.run(debug=True) line in a Flask application is responsible for starting the development server.

To use this code, students would need to have Flask installed (`pip install flask`) and then run this script.

After running the script, they can open a web browser and navigate to `http://127.0.0.1:5000/` to see the "Hello World“ message.

Important for Practical

1. Open Your Folder in VS Code.
2. Open terminal.
3. Type “pip install virtualenv” (for creating virtual environment)
4. To create a virtual environment type “virtualenv env”
5. If there is an error type “Set-ExecutionPolicy unrestricted” in Windows PoweShell. Type ‘A’.
6. For activate environment : In other terminal “.\env\Scripts\activate.ps1”

7. For installing flask : pip install flask
8. For run a program : python flask1.py (python file name)

Important Python Program of flask

1. First Flask Application
2. Flask App Routing
3. Flask URL Building
4. Flask HTTP Methods
5. Flask Templates
6. Flask Request Object

1. First Flask Application

- Creating a Flask App:

```
from flask import Flask
app = Flask(__name__)
@app.route('/')
def hello_world():
        return 'Hello, World!'
if __name__ == '__main__’:
        app.run()
```

2. Flask App Routing

- Defining Routes:

```
from flask import Flask
app = Flask(__name__)
@app.route("/")
def hello_world():
    return "Hello, Students! We are on Home page."

@app.route("/products")
def products():
    return "Hello, Students! We are on Product page."

if __name__ == '__main__':
    app.run(debug=True)
```

Create a static and templates folder in your flask folder.
In static folder there is files are loaded as it is.
In templates folder we can add our html files.

3. Flask Templates

- Using Templates:

```
from flask import Flask, render_template
app = Flask(__name__)
@app.route('/')
def hello_world():
    return render_template('index.html')
if __name__ == '__main__':
    app.run(debug=True)
```

1. Install flask-sqlalchemy (provide a facilities to change in database using python)
2. Repr method for which columns you want to see.
3. Open python in terminal

```
conda activate myenv
python -m venv myvirtualenv
.\myvirtualenv\Scripts\Activate.ps1
ractical\Flask> python
```

4. Install Jinja2 Snippet Kit

3. Flask URL Building

- URL Building:

```
from flask import Flask, url_for
app = Flask(__name__)
@app.route('/')
def index():
return 'index'
@app.route('/login')
def login():
return 'login'
@app.route('/user/<username>')
def profile(username):
return f'{username}\'s profile'
with
app.test_request_context():
print(url_for('index')) Output: /
print(url_for('login')) Output:
/login
print(url_for('profile',
username='John Doe'))
Output: /user/John%20Doe
```

4. Flask HTTP Methods

- Handling HTTP Methods:

```
from flask import request
@app.route('/login', methods=['GET', 'POST'])
def login():
if request.method == 'POST':
return 'do_login()'
else:
return 'show_login_form()'
```

6. Flask Request Object

- Accessing Request Data:

```
from flask import request
@app.route('/login', methods=['POST', 'GET'])
def login():
if request.method == 'POST':
return request.form['username']
else:
return 'GET request'
```

Django

- Django is a Python-based web framework that allows you to quickly create efficient web applications.
- It is also called batteries included framework because Django provides built-in features for everything including Django Admin Interface, default database – SQLlite3, etc.
- When you’re building a website, you always need a similar set of components: a way to handle user authentication (signing up, signing in, signing out), a management panel for your website, forms, a way to upload files, etc.
- Django gives you ready-made components to use and that too for rapid development.

Why Django Framework ?

- Excellent documentation and high scalability.
- Used by Top MNCs and Companies, such as Instagram, Spotify, Youtube, Dropbox, etc. and the list is never-ending.
- Easiest Framework to learn and rapid development.
- The last but not least reason to learn Django is Python, Python has huge library and features such as Web Scraping, Machine Learning, Image Processing, Scientific Computing, etc.
- One can integrate it all this with web application and do lots and lots of advance stuff.

Django architecture

- Django is based on MVT (Model-View-Template) architecture.
- MVT is a software design pattern for developing a web application.
- MVT Structure has the following three parts –
- **Model:** Model is going to act as the interface of your data. It is responsible for maintaining data. It is the logical data structure behind the entire application and is represented by a database (generally relational databases such as MySql, Postgres).

Django architecture

- **View:** The View is the user interface — what you see in your browser when you render a website. It is represented by HTML/CSS/Javascript and Jinja files.
- **Template:** A template consists of static parts of the desired HTML output as well as some special syntax describing how dynamic content will be inserted.

In a Django project, there are several important files and directories that play crucial roles in the development and deployment of the application.
Here are some of the most important ones:

1. **manage.py:**

- This is a command-line utility that is automatically created when you create a new Django project using the `django-admin startproject` command.
- It provides various commands for managing the Django project, such as running the development server, creating database migrations, and running tests.

2. settings.py:

- This file contains configuration settings for the Django project.
- It includes settings related to database configuration, security, middleware, installed apps, static files, template directories, and more.
- It's one of the most important files in a Django project as it defines how the project behaves and interacts with various components.

3. urls.py:

- This file contains URL patterns for the Django project.
- It maps URLs to view functions or classes, which handle requests and return responses.
- It acts as a router for directing incoming requests to the appropriate views.

4. models.py:

- This file contains definitions of database models for the Django project.
- Models define the structure and behavior of the data stored in the database.
- Each model class corresponds to a database table, and its attributes represent fields in the table.
- **5. views.py:**
- This file contains view functions or classes for the Django project
- Views receive HTTP requests, process them, and return HTTP responses.
- Views interact with models to retrieve or manipulate data and render templates to generate HTML responses.

6. admin.py:

- This file is used to register models with the Django admin interface.
- It allows you to manage and interact with models using the Django admin site.
- You can customize the appearance and behavior of models in the admin interface by defining ModelAdmin classes.

7. templates/:

- This directory contains HTML template files used for rendering dynamic content in the Django project.
- Templates use the Django template language to include variables, logic, and template tags.
- By default, Django looks for templates in this directory and its subdirectories.

8. static/:

- This directory contains static files such as CSS, JavaScript, images, and other assets used in the Django project.
- Static files are served directly by the web server (e.g., Nginx or Apache) in production for better performance.

9. migrations/:

- This directory contains database migration files generated by Django's migration system.
- Migrations are Python files that describe changes to the database schema over time.
- They allow you to version-control and apply database schema changes to different environments.

Installation of Django

- Install python3 if not installed in your system (according to configuration of your system and OS) from **here**.
- Try to download the latest version of python.
- Install pip-Open command prompt and enter following command-
- Install virtual environment-Enter following command in cmd-
- Set Virtual environment-Setting up the virtual environment will allow you to edit the dependency which generally your system wouldn’t allow.
- Follow these steps to set up a virtual environment1.

1. Create a virtual environment by giving this command in cmd :

```
python -m virtualenv env_site
```

2. Change directory to env_site by this command :

```
cd env_site
```

3. Go to Script directory inside env_site and activate virtual environment :

```
Cd Scripts
activate
```

4. Install Django- Install django by giving following command

```
pip install Django
```

5. To initiate a project of Django on Your PC, open Terminal and Enter the following command

```
django-admin startproject projectName
```

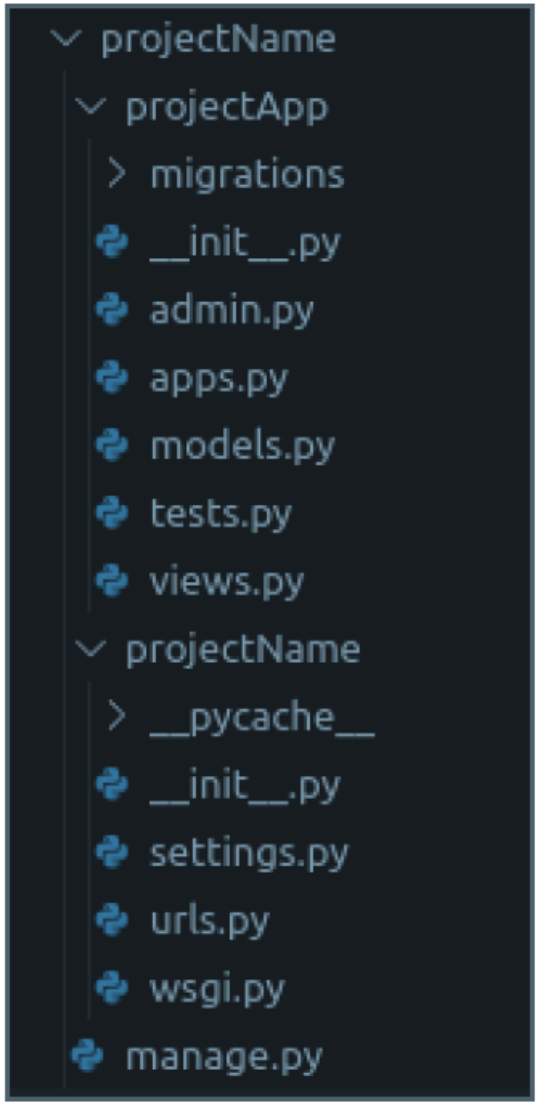
6. A New Folder with name projectName will be created. To enter in the project using terminal enter command

```
cd projectName
```

7. To create a basic app in your Django project you need to go to directory containing manage.py and from there enter the command:

```
python manage.py startapp projectApp
```

Now you can see your directory structure as under :



1.Django

- Django works on the DRY (Don't Repeat Yourself) principle, aiming to minimize code redundancy.

- It's designed for rapid development, offering a clean and pragmatic way to build web applications.

2. Core Components of Django

Models :

- Django models are the source of information about your data. They contain essential fields and behaviors of the data stored in your database.

- Django follows an ORM (Object-Relational Mapping) approach, allowing easy data manipulation through Python objects, rather than SQL queries.

Views:

- Views in Django are responsible for processing user requests and returning responses.
- Django views can be function-based or class-based. Class-based views provide more flexibility and are reusable.

Templates:

- Django's templating engine provides a powerful way to generate HTML dynamically.
- Templates contain static parts of the desired HTML output as well as some special syntax describing how dynamic content will be inserted.

URL Dispatcher:

- URL patterns are defined in Django’s URL dispatcher, which uses regular expressions to capture URL patterns for processing.

3. Advanced Features

Django Admin Interface:

- One of Django's most powerful components is its automatically-generated admin interface. It reads metadata from your models to provide a robust interface for managing your data.

Forms:

- Django Forms simplify the task of creating and processing HTML forms.

They handle data validation and convert form inputs to Python types for further processing.

3. Advanced Features

Authentication:

- Django comes with a built-in authentication system that manages user accounts, groups, permissions, and cookie-based user sessions.

Security Features:

- Django is highly focused on security, with features protecting against CSRF, SQL injection, XSS, and other common web attacks.

4. Working with Databases

ORM and Migrations:

- Django’s ORM allows you to interact with your database as if you were using Python code instead of SQL.

- Migrations in Django are a way of propagating changes made in models (like adding a field, deleting a model, etc.) into the database schema.

5. Extending Django

Django REST Framework:

- For building APIs, Django REST Framework (DRF) is a powerful and flexible toolkit. It provides features for serialization, authentication, and viewsets for RESTful API development.

Customizations:

- Django is highly customizable. You can extend or replace almost every component of the framework to fit your needs.

6. Deployment and Performance

Deployment:

- Deploying a Django application involves several steps, including setting up a web server (like Gunicorn or uWSGI), configuring a database, and serving static and media files.

Performance Optimization:

- Django offers various ways to optimize your application’s performance, like database indexing, query optimization, caching, and asynchronous processing.

7. Testing in Django

Test Framework:

- Django has a built-in framework for writing unit tests. It uses a Python standard library module: unittest.

Test Types:

- Django supports various types of tests including unit tests, integration tests, and end-to-end tests.

GIS Web Services

- GIS (Geographic Information Systems) Web Services are specialized web services designed to handle, manipulate, and display geospatial data.
- They play a crucial role in modern web applications, especially in areas like mapping, spatial analysis, urban planning, environmental analysis, and more.

Key Concepts:

1. Geospatial Data:

Definition: This involves data that is linked to specific geographical locations.

Types: Includes raster data (like satellite imagery), vector data (like roads, boundaries), and attribute data (additional information like population, land use).

2. GIS Web Service Functions:

Mapping: Displaying geospatial data in map form.

Spatial Analysis: Performing operations like routing, geocoding, buffer analysis, etc.

Data Management: Storage, retrieval, and manipulation of geospatial data.

Types of GIS Web Services:

1. WMS (Web Map Service):

Delivers maps as images, which are generally produced by a map server from data provided.

Offers layers that can be turned on/off, transparency control, but no direct interaction with individual map elements.

2. WFS (Web Feature Service):

Provides direct access to geospatial features.

Allows for querying and retrieval of geospatial data as vector features, enabling more interaction and customization.

Types of GIS Web Services:

3. WCS (Web Coverage Service):

Designed for raster data retrieval.

Provides access to geospatial data in forms that can be directly used for analysis and processing.

4. WPS (Web Processing Service):

Focuses on the processing and manipulation of geospatial data.

Enables the execution of spatial algorithms and operations.

Implementation Technologies:

Servers: Software like GeoServer, MapServer, or proprietary solutions like ArcGIS Server.

Data Formats: Common formats include GeoJSON, KML (Keyhole Markup Language), and GML (Geography Markup Language).

Client-Side Libraries: JavaScript libraries like Leaflet, OpenLayers, and the Google Maps API for building interactive maps.

Challenges and Considerations:

1. Performance: Handling and processing geospatial data can be resource-intensive.

2. Scalability: Managing large datasets and high user loads requires efficient scaling strategies.

3. Data Quality and Accuracy: Ensuring that the geospatial data is accurate and upto-date.

4. Security: Protecting sensitive data, especially when dealing with data that has privacy implications.

Applications of GIS Web Services:

Urban Planning: Analysis and visualization of urban growth, infrastructure planning.

Environmental Monitoring: Tracking changes in land use, climate change impacts.

Logistics: Route planning, fleet management, and optimization.

Disaster Management: Real-time mapping of disaster zones for efficient response.

Conclusion:

- GIS Web Services are integral to the handling and presentation of geographical information on the web.
- They offer powerful tools for visualization, analysis, and interaction with spatial data.
- Teaching web development, introducing these concepts can greatly enhance your students' understanding of the diverse applications of web technology in solving real-world problems.