MODULE 10) REST FRAMEWORK

➢ Introduction to APIs:

1.What is an API (Application Programming Interface)?

Ans:

- **API (Application Programming Interface)** is a set of rules that allows one software application to communicate with another.
- In simple words, an API works like a **messenger**. It takes a request from one program, sends it to another program or server, and then brings back the response.
- For example, when you use a mobile app to check the weather, the app does not store weather data itself. It sends a request to a weather server using an API, and the server sends back the weather information.

APIs are used to:

- Share data between different applications.
- Connect frontend (website or app) with backend (server or database).
- Save time by reusing existing services (like payment, login, maps)

Example:

Login with Google, payment through UPI, or showing maps in an app — all use APIs.

2.Types of APIs: REST, SOAP.

Ans:

1. REST API (Representational State Transfer):

- REST API is the most commonly used type of API.
- It works over the internet using standard HTTP methods such as **GET, POST, PUT, and DELETE**.
- REST APIs usually exchange data in **JSON format**, which is lightweight and easy to understand.

Features of REST API:

- Simple and easy to use
- Fast and lightweight
- Uses HTTP methods
- Mostly uses JSON data format
- Widely used in web and mobile applications

2. SOAP API (Simple Object Access Protocol):

- SOAP API is a more structured and secure type of API.
- It uses **XML format** for sending messages and follows strict rules and standards.
- SOAP is commonly used in **banking and enterprise applications** where high security is required.

Features of SOAP API:

- Uses XML only
- Highly secure
- Strict rules and standards
- Supports complex operations
- Slower compared to REST

3.Why are APIs important in web development?

Ans:

- APIs are very important in web development because they allow different parts of a web application to **communicate with each other** easily and efficiently.

1. **Connect Frontend and Backend:**

APIs help the frontend (HTML, CSS, JavaScript) send requests to the backend (server, database) and receive data in return.

2.**Data Sharing:**

APIs allow different applications to share data securely, such as user information, products, or orders.

3.**Reuse Existing Services:**

Developers can use ready-made services like **payment gateways, Google login, maps, and email services** instead of building everything from scratch.

4.**Better Security:**

APIs control how data is accessed, keeping databases safe from direct access.

5.**Faster Development:**

APIs save time and effort by allowing developers to focus on features rather than rebuilding common functionalities.

6.**Scalability:**

APIs make it easy to update or expand applications without changing the entire system.

➢ Requirements for Web Development Projects:

1.Understanding project requirements.

Ans:

- Understanding project requirements means **clearly knowing what needs to be built before starting development**.
- It helps developers understand the client's needs, project goals, and expected results.

Key points to understand project requirements:

1. **Project Objective:**

   Understand the main purpose of the project and what problem it should solve.

2. **User Requirements:**

   Identify who will use the system and what features they need.

3. **Functional Requirements:**

   Define what the system should do, such as login, data entry, reports, etc.

4. **Non-Functional Requirements:**

   Understand performance, security, speed, and usability expectations.

5. **Tools and Technology:**

   Decide which programming languages, frameworks, and databases will be used.

6. **Timeline and Budget:**

   Know the project deadline and cost limitations.

2. Setting up the environment and installing necessary packages.

Ans:

- Setting up the environment means **preparing your system for development** by installing all required software, tools, and packages before starting the project.

Why it is important:

- Ensures the project runs smoothly
- Prevents errors during development
- Saves time later

Steps involved in setting up the environment:

1. **Install Required Software:**

   Install necessary software such as an operating system, code editor (VS Code), browser, and programming language (Python, Java, etc.).

2. **Set Up Development Tools:**

   Configure tools like terminal/command prompt, version control (Git), and virtual environment if needed.

3. **Install Required Packages:**

   Install libraries and frameworks needed for the project using package managers (example: pip, npm).

4. **Configure Project Settings:**

   Set environment variables, database connections, and configuration files.

5. **Test the Setup:**

   Run a simple program or test command to make sure everything is installed correctly.

➢ Serialization in Django REST Framework:

1.What is Serialization?

Ans:

- **Serialization** is the process of **converting an object or data into a format that can be easily stored or sent over a network**, and later converted back into its original form.
- In web development, serialization is commonly used to **convert data into JSON or XML format** so it can be shared between the frontend and backend through APIs.

Why serialization is important:

- Helps in sending data through APIs.
- Makes data easy to store in files or databases.
- Allows communication between different systems.

Example:

A user object with name, email, and age is converted into JSON before sending it to a client.

When received, it is converted back into an object.

2. Converting Django QuerySets to JSON.

Ans:

- In Django, a **QuerySet** contains data fetched from the database.
- To send this data through an API or to the frontend, it must be **converted into JSON format**. This process is called **serialization**.

Why convert QuerySets to JSON?

- Frontend applications understand JSON easily
- Required for API responses
- Helps in data exchange between server and client

Common ways to convert Django QuerySets to JSON:

1. **Using Django's built-in serializers:**

   Django provides a serializer to convert QuerySets into JSON format.

2. **Using JsonResponse:**

   JsonResponse converts Python data into JSON automatically and is useful for API responses.

3. **Using Django REST Framework (DRF):**

   DRF provides powerful serializers that make converting QuerySets to JSON easy and flexible.

Example explanation:

When you fetch a list of users from the database, the QuerySet cannot be sent directly to the browser.

It is first converted into JSON so the browser can display or use the data.

3.Using serializers in Django REST Framework (DRF).

Ans:

- In **Django REST Framework (DRF)**, **serializers** are used to **convert complex data like Django models and QuerySets into JSON format**, and also to **convert JSON data back into Python objects**.

Why serializers are used in DRF:

- Convert model data into JSON for APIs
- Validate incoming data
- Save data to the database
- Make API responses clean and structured

Main uses of serializers:

1. **Data Conversion:**

   Serializers transform Django model instances into JSON and JSON into Python objects.

2. **Data Validation:**

   They check whether the incoming data is correct before saving it.

3. **Easy CRUD Operations:**

   Serializers help in creating, reading, updating, and deleting data.

Example explanation:

When a client sends data to an API, the serializer validates it.

If valid, it saves the data to the database.

When data is fetched, the serializer converts it into JSON and sends it to the client.

➢ Requests and Responses in Django REST Framework:

1.HTTP request methods (GET, POST, PUT, DELETE).

Ans:

- HTTP request methods are used by a client (browser or app) to **communicate with a server**.

- Each method has a specific purpose in web development and APIs.

1. GET:

GET is used to **retrieve data** from the server. It does not change any data on the server.

- Used to fetch information
- Data is visible in the URL
- Safe and read-only

Example:

Getting a list of users from a database.

2. POST:

POST is used to **send new data** to the server.

- Used to create new records
- Data is sent in the request body
- More secure than GET

Example:

Submitting a signup form or adding a new user.

3. PUT:

PUT is used to **update existing data** on the server.

- Replaces or updates a record
- Sends complete data to be updated

Example:

Updating a user's profile details.

4. DELETE:

DELETE is used to **remove data** from the server.

- Deletes a record
- Used carefully to avoid data loss

Example: Deleting a user account.

2.Sending and receiving responsesin DRF.

Ans:

- In **Django REST Framework (DRF)**, sending and receiving responses means **handling data between the client (frontend or app) and the server through APIs**.

Sending Responses in DRF:

When a request is processed, DRF sends data back to the client using a **Response** object.

- Data is usually sent in **JSON format**
- Response includes **status codes** (200, 201, 400, etc.)
- Uses Response() from rest_framework.response

Example explanation:

After fetching data from the database, DRF converts it into JSON using serializers and sends it as a response.

Receiving Requests in DRF:

DRF receives data sent by the client using HTTP methods like **POST or PUT**.

- Data is accessed using request.data
- DRF automatically parses JSON data
- Incoming data is validated using serializers

Example explanation:

When a client submits a form, DRF receives the data, validates it, saves it to the database, and sends a success response.

➢ Views in Django REST Framework:

1.Understanding views in DRF: Function-based views vs Class-based views.

Ans:

There are two main types of views in DRF:

1. Function-Based Views (FBV):

- Function-based views use **Python functions** to handle requests.

Features:

- Simple and easy to understand
- Good for small or simple APIs
- Uses decorators like @api_view

Example explanation:

A single function handles GET or POST requests and returns a response.

Advantages:

- Easy to write and read
- Good for beginners

Disadvantages:

- Not reusable
- Becomes messy for large projects

2. Class-Based Views (CBV):

- Class-based views use **Python classes** to handle requests.

Features:

- Organized and reusable
- Better for large and complex APIs
- Supports generic views (List, Create, Update, Delete)

Example explanation:

Different HTTP methods (GET, POST, PUT, DELETE) are handled inside one class.

Advantages:

- Clean and structured code
- Reusable and scalable
- Less code using generic views

Disadvantages:

- Slightly harder to understand for beginners.

➢ URL Routing in Django REST Framework:

1. Defining URLs and linking them to views.

Ans:

- In Django, **URLs** are used to decide **which page or function should run when a user opens a particular web address**.

- The URL configuration acts as a bridge between the browser request and the Django view.

What is a URL in Django?

A URL is a web address that a user enters in the browser, such as:

http://127.0.0.1:8000/home/

Django checks this URL and connects it to a **view**, which returns a response (HTML page, JSON data, etc.).

What is a View?

A **view** is a Python function or class that contains the **logic** of the application.

It receives a request and sends back a response.

Example view:

```
from django.http import HttpResponse

def home(request):
    return HttpResponse("Welcome to Home Page")
```

Defining URLs in Django

URLs are defined in a file called **urls.py**.

Example:

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

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

Here:

- 'home/' → URL path
- views.home → View function
- name='home' → Used for URL linking

Linking URLs to Views

When a user opens:

/home/

1. Checks urls.py
2. Finds the matching path
3. Calls the connected view
4. Displays the response in the browser

Linking URLs in HTML Templates:

Django provides a **URL tag** to link pages safely.

Example:

```
<a href="{% url 'home' %}">Home</a>
```

This helps avoid hardcoding URLs and makes the project easier to maintain.

➢ Pagination in Django REST Framework:

1. Adding pagination to APIs to handle large data sets.

Ans:

- When an API returns a large amount of data at once, it can make the application **slow** and **difficult to use**.
- To solve this problem, **pagination** is used.
- Pagination divides large data into **small pages** and sends only a limited number of records per request.

What is Pagination?

- Pagination is a technique where data is split into multiple pages.
- Instead of returning all records, the API returns a fixed number of items per page.

Example:

- Page 1 → 10 records
- Page 2 → next 10 records

Pagination in Django REST Framework (DRF):

➢ Django REST Framework provides built-in pagination classes.

Common pagination types:

- **PageNumberPagination**
- **LimitOffsetPagination**
- **CursorPagination**

Example: Page Number Pagination

settings.py:

```
REST_FRAMEWORK = {
   'DEFAULT_PAGINATION_CLASS':
   'rest_framework.pagination.PageNumberPagination',
   'PAGE_SIZE': 5,
}
```

How It Works

API request:

/api/students/?page=1

API response:

```
{
  "count": 50,
  "next": "http://api/students/?page=2",
  "previous": null,
  "results": [
    { "id": 1, "name": "Amit" },
    { "id": 2, "name": "Neha" }
  ]
}
```

Benefits of Using Pagination:

- Handles large data efficiently
- Makes APIs scalable
- Prevents loading unnecessary data
- Suitable for mobile and web applications

➢ Settings Configuration in Django:

1. Configuring Django settings for database,static files, and API keys.

Ans:

In Django, all main configurations are done in the settings.py file.

- **Database Configuration:**

  The DATABASES setting is used to connect Django with a database like SQLite or MySQL.

  It includes details such as database name, username, password, host, and port.

  This allows Django to store and retrieve application data.

- **Static Files Configuration:**

  Static files like CSS, JavaScript, and images are managed using STATIC_URL and STATICFILES_DIRS.

  These settings help Django know where static files are located and how to serve them in the project.

- **API Keys Configuration:**

  API keys are used to connect with third-party services (like payment gateways or email services).

  They are usually stored in settings.py or as environment variables to keep them secure and easy to manage.

➢ Project Setup:

1.Setting up a Django REST Framework project.

Ans:

To set up a Django REST Framework project, first install Django and Django REST Framework using pip.

Django is the main web framework, and DRF is an extension that helps in building RESTful APIs easily.

After installation, create a new Django project using the django-admin startproject command.

Inside the project, create an app using python manage.py startapp app_name.

This app will contain models, serializers, views, and URLs related to the API.

Next, open the settings.py file and add rest_framework and your app name to the INSTALLED_APPS list.

This step is important because it activates Django REST Framework features in the project.

Then configure the database (SQLite or MySQL) so Django can store and manage data properly.

Now, create models if your API needs to store data.

After defining models, create serializers using DRF.

Serializers convert Django model data into JSON format and also validate incoming request data.

This makes communication between the backend and frontend or mobile apps easier.

After that, create API views using function-based views or class-based views provided by DRF.

These views handle HTTP methods like GET, POST, PUT, and DELETE.

Then connect these views to URLs using the urls.py file so API endpoints can be accessed through a browser or tools like Postman.

Finally, run database migrations and start the development server. At this point, your Django REST Framework project is ready to handle API requests and responses in a clean, structured, and scalable way.

➢ Social Authentication, Email, and OTP Sending API:

1.Implementing social authentication (e.g., Google, Facebook) in Django.

Ans:

1.**Install Packages:**

Install Django and django-allauth using pip.

2.**Update Settings:**

Add django.contrib.sites, allauth, allauth.account, allauth.socialaccount, and provider apps (Google/Facebook) to INSTALLED_APPS.

Set SITE_ID and authentication backends in settings.py.

3. **Configure URLs:**

Include allauth URLs in the project's urls.py.

4.**Create OAuth Credentials:**

Generate Client ID and Client Secret from Google Developer Console or Facebook Developer Portal.

5. **Add Social App in Admin:**

Enter Client ID and Client Secret in Django Admin → Social Applications.

6.**Update Templates:**

Add "Login with Google/Facebook" buttons in the login page.

7.**Test Authentication:**

Run the server and test social login.

2.Sending emails and OTPs using third-party APIslike Twilio, SendGrid.

Ans:

1. **Choose a Service:**

Use **SendGrid** for emails and **Twilio** for SMS/OTP delivery.

2.**Create an Account & Get API Keys:**

Sign up on Twilio/SendGrid and generate API keys or credentials.

3. **Install Required Packages:**

Install the official SDKs using pip.

4.**Configure Settings:**

Store API keys securely in settings.py or environment variables.

5.**Generate OTP:**

Create a random numeric OTP in Django and save it temporarily (session or database).

6. **Send Email or SMS:**

Use SendGrid API to send emails and Twilio API to send OTP via SMS.

7.**Verify OTP:**

Compare the user-entered OTP with the stored OTP to verify the user.

➢ RESTful API Design:

1.REST principles: statelessness, resource-based URLs, and using HTTP methods for CRUD operations.

Ans:

REST APIs follow stateless communication, use clear resource-based URLs, and apply HTTP methods to perform CRUD operations in a clean and standard way.

1. **Statelessness:**

   Each API request is independent and contains all required information.

   The server does not store client session data between requests.

2. **Resource-Based URLs:**

   URLs represent resources (data) instead of actions.

   Example: /users/, /products/1/

3. **Using HTTP Methods for CRUD Operations:**

   - **GET**:-

     Read data.

   - **POST**:

     Create new data.

   - **PUT / PATCH**:-

     Update existing data.

   - **DELETE**:-

     Remove data.

➢ CRUD API (Create, Read, Update, Delete):

1. What is CRUD, and why is it fundamental to backend development?

Ans:

- CRUD stands for **Create, Read, Update, and Delete**.

These are the four basic operations performed on data in any backend application.

- **Create:**

  Add new data to the database

- **Read:**

  Retrieve or view existing data

- **Update:**

  Modify existing data

- **Delete:**

  Remove data from the database

CRUD is fundamental to backend development because almost every application (such as websites, mobile apps, or APIs) needs to manage data.

Backend logic, databases, and APIs are mainly designed around these four operations to store, display, update, and remove information efficiently.

➢ Authentication and Authorization API:

1.Difference between authentication and authorization.

Ans:

Authentication	Authorization
Verifies **who the user is**	Verifies **what the user can access**
Uses login details like username, password, OTP	Uses roles and permissions
Performed at login time	Performed after authentication
Confirms user identity	Controls access to resources
Example: Login with email & password	Example: Admin can delete users

2.Implementing authentication using Django REST Framework's token-based system.

Ans:

1.**Install DRF Token Package:**

Add rest_framework.authtoken to INSTALLED_APPS and install required packages.

2.**Run Migrations:**

Apply migrations to create the token table in the database.

3.**Update DRF Settings:**

Configure DEFAULT_AUTHENTICATION_CLASSES to use token authentication in settings.py.

4. **Generate Token:**

Create a token for each user (automatically or via login API).

5. **Create Login API:**

Build an API endpoint that returns the token after successful user login.

6. **Send Token in Requests:**

The client sends the token in the request header:
Authorization: Token <token>

7. **Protect APIs:**

Use permission classes to allow access only to authenticated users.

➢ OpenWeatherMap API Integration:

1. Introduction to OpenWeatherMap API and how to retrieve weather data.

Ans:

- **OpenWeatherMap API** is a web service that provides real-time weather data like temperature, humidity, wind speed, and forecasts for any city in the world.

- Developers use it to add weather information to websites and applications.

How to Retrieve Weather Data (Easy Steps):

1. **Create an account** on OpenWeatherMap

2. **Get an API key** (this key is required to access data)

3. **Make an API request** using a URL

   Example:

   https://api.openweathermap.org/data/2.5/weather?q=London&appid=YOUR_API_KEY

4. **Receive data in JSON format**

   The response includes:

- Temperature
- Weather condition (clear, rain, etc.)
- Humidity
- Wind speed

➢ Google Maps Geocoding API:

1.Using Google Maps Geocoding API to convert addresses into coordinates.

Ans:

- The **Google Maps Geocoding API** converts a human-readable address (like a city or street) into **latitude and longitude**, which can be used in maps, weather apps, and location-based services.

Steps to Use Google Geocoding API (Easy)

1. **Create a Google Cloud account**
2. **Enable Geocoding API**
3. **Generate an API key**

API Request Format:

https://maps.googleapis.com/maps/api/geocode/json?address=New+York&key=YOUR_API_KEY

Sample JSON Response (Important Fields)

- lat → Latitude
- lng → Longitude

Python Example (Simple & Clear)

```
import requests

address = "New York"
url = "https://maps.googleapis.com/maps/api/geocode/json"
params = {
   "address": address,
   "key": "YOUR_API_KEY"
}

response = requests.get(url, params=params)
data = response.json()

location = data["results"][0]["geometry"]["location"]
print("Latitude:", location["lat"])
print("Longitude:", location["lng"])
```

➢ GitHub API Integration:

1.Introduction to GitHub API and how to interact with repositories, pull requests, and issues.

Ans:

The **GitHub API** is a web service provided by GitHub that allows developers to interact with GitHub using code instead of the website.

With the help of the GitHub API, we can access repositories, manage pull requests, and handle issues programmatically.

GitHub API works using the **REST architecture** and returns data in **JSON format**.

It uses standard HTTP methods like **GET, POST, PUT, and DELETE**.

Interacting with Repositories:

Using the GitHub API, we can get information about repositories such as repository name, owner, number of stars, forks, and description.

Example:

- API Endpoint: GET /repos/{owner}/{repository}
- This request returns details of a specific repository.

Repositories are commonly used to store project code and manage versions.

Interacting with Pull Requests:

A **Pull Request (PR)** is used to propose changes to a repository. Using the GitHub API, we can:

- View all pull requests
- Check their status (open or closed)
- Read pull request details

Example:

- API Endpoint: GET /repos/{owner}/{repository}/pulls

Pull requests help in code review and team collaboration.

Interacting with Issues:

Issues are used to report bugs, request features, or track tasks. With the GitHub API, we can:

- View issues
- Create new issues
- Update existing issues

Example:

- API Endpoint: GET /repos/{owner}/{repository}/issues

Issues make project management easier and organized.

➢ Twitter API Integration:

1.Using Twitter API to fetch and post tweets, and retrieve user data.

Ans:

The **Twitter API** allows developers to interact with Twitter using code.

With the help of this API, we can **fetch tweets, post new tweets, and get user information** without using the Twitter website.

To use the Twitter API, a developer must create a Twitter Developer account and generate **API keys and access tokens** for authentication.

Fetching Tweets:

Using the Twitter API, we can fetch tweets from:

- A specific user
- A hashtag
- Recent or popular tweets

Example Use:
Fetching recent tweets helps in analyzing trends, opinions, and user activity.

Posting Tweets:

The Twitter API also allows users to post tweets programmatically. By sending a request with proper authentication, we can publish tweets directly from an application.

Example Use:
Automatic posting of updates, notifications, or promotional messages.

Retrieving User Data:

With the Twitter API, we can get user-related information such as:

- Username
- Profile details
- Followers and following count
- Tweet count

Example Use:
Displaying user profiles or analyzing follower growth.

➢ REST Countries API Integration:

1.Introduction to REST Countries API and how to retrieve country-specific data.

Ans:

The **REST Countries API** is a free web service that provides information about countries around the world.

It allows developers to retrieve **country-specific data** such as country name, capital, population, region, currency, and languages using simple API requests.

The API follows the **REST architecture** and returns data in **JSON format**, which is easy to read and use in web and mobile applications.

How to Retrieve Country-Specific Data:

To get data from the REST Countries API, we send an **HTTP GET request** to the API endpoint with the required country information like name or code.

Example API Request

https://restcountries.com/v3.1/name/india

Types of Country Data Available:

Using the REST Countries API, we can retrieve:

- Country name
- Capital city
- Population
- Region and sub-region
- Currency
- Languages
- Flag information

➢ Email Sending APIs (SendGrid, Mailchimp):

1.Using email sending APIs like SendGrid and Mailchimp to send transactional emails.

Ans:

Email sending APIs like **SendGrid** and **Mailchimp** allow developers to send emails automatically from applications.

These emails are called **transactional emails**, which are sent when a specific action occurs, such as user registration, password reset, order confirmation, or payment receipt.

Both SendGrid and Mailchimp provide **REST APIs** that work using HTTP requests and return responses in **JSON format**.

Using SendGrid API:

- **SendGrid** is mainly used for sending transactional emails.

Features:

- Fast and reliable email delivery
- Easy API integration
- Supports HTML and text emails

Use Case:
Send automatic login alerts or password reset links.

Using Mailchimp API:

- **Mailchimp** is mostly used for marketing emails but also supports transactional emails through its API.

Features:

- Email templates
- Audience management
- Analytics and reports

Use Case:
Send welcome emails and notifications.

➢ SMS Sending APIs (Twilio):

1.Introduction to Twilio API forsending SMS and OTPs.

Ans:

The **Twilio API** is a cloud-based communication service that allows developers to send **SMS messages and OTPs (One-Time Passwords)** using code.

It is widely used for **user verification, authentication, and notifications** in web and mobile applications.

Twilio works using **REST APIs** and communicates through **HTTP requests**, returning responses in **JSON format**.

Sending SMS Using Twilio API:

With the Twilio API, developers can send SMS messages to users automatically.

Common Uses:

- Account notifications
- Transaction alerts
- Appointment reminders

Sending OTPs Using Twilio API:

OTPs are temporary codes sent to users for verification and security purposes.

Common Uses:

- Login verification
- Signup verification
- Password reset confirmation

Twilio ensures fast and reliable delivery of OTP messages.

➢ Payment Integration (PayPal, Stripe):

1.Introduction to integrating payment gateways like PayPal and Stripe.

Ans:

Integrating payment gateways like **PayPal** and **Stripe** allows applications to accept online payments securely.

Using their APIs, developers can easily manage payments, refunds, and transaction details in web applications.

PayPal Payment Gateway:

PayPal is a widely used online payment system that allows users to pay using their PayPal account or cards.

Features:

- Secure and trusted worldwide
- Supports multiple currencies
- Easy checkout process

Use Case:
Used in e-commerce websites for fast and secure payments.

Stripe Payment Gateway:

Stripe is a modern payment platform designed for developers.

Features:

- Simple API integration
- Supports cards, UPI, and wallets
- Strong security and fraud protection

Use Case:
Commonly used for subscription-based and international payment systems.

➢ Google Maps API Integration:

1.Using Google Maps API to display maps and calculate distances between locations.

Ans:

The **Google Maps API** allows developers to add interactive maps and location features to web and mobile applications.

It helps in displaying maps, marking locations, and calculating distances between two or more places.

Google Maps API works using **REST and JavaScript APIs** and returns data in **JSON format**.

Displaying Maps Using Google Maps API:

Using the Google Maps JavaScript API, developers can display maps on a web page.

Features:

- Zoom in and out
- Switch map views (roadmap, satellite)
- Add markers for locations

Use Case:
Showing office locations, stores, or user addresses on a map.

Calculating Distance Between Locations:

Google Maps provides the **Distance Matrix API** to calculate distance and travel time between locations.

Common Uses:

- Delivery distance calculation
- Route planning
- Cab and delivery apps

The distance can be calculated using latitude and longitude or place names.