ApexaiQ: 06feb_Day4

1) Difference Between JSON and XML

Feature	JSON (JavaScript Object Notation)	XML (eXtensible Markup Language)
Syntax	- Lightweight and simpler syntax.	- Verbose and complex syntax with custom tags.
	- Uses key-value pairs for objects and arrays for data grouping.	- Uses opening and closing tags for each element, which can be nested.
Readability	- More compact and easier to read, especially for humans.	- Less readable due to the presence of many tags and attributes.
Data Representation	- Represents data as key-value pairs and arrays.	- Represents data as elements enclosed in tags.
	- Allows nested objects and arrays for complex data structures.	- Allows nesting but data is always enclosed in tags, making it more verbose.
Data Types	- Supports native data types such as strings, numbers, Booleans, arrays, and objects.	- All data is treated as text. Data types need to be manually specified or inferred.
Complexity	- Easier to parse, manipulate, and work with programmatically	- More complex, requires parsers and specialized handling.

- **JSON**: It is compact, easy to read and write, and is commonly used for data exchange between servers and web clients. Its simplicity and lightweight nature make it the go-to choice for modern web APIs, mobile applications, and services requiring fast, efficient communication.

- **XML**: It is highly flexible and extensible, making it useful for complex data representations, document storage, and services where metadata and strict validation are important. XML's verbose nature is less efficient for quick data transfer, but its robustness makes it ideal for highly structured, long-term data storage.

What is API?

An **API** (Application Programming Interface) is a set of rules, protocols, and tools that allow different software applications to communicate with each other. It defines the methods and data formats that applications can use to request and exchange information.

APIs can be used to access the functionalities of an external service, such as a web application or a database, without needing to understand its internal workings. They are commonly used to enable integration between different systems, applications, or platforms.

Types of APIs:

1. **Web APIs**: Allow interaction with web services over HTTP/HTTPS. Examples include REST (Representational State Transfer) and SOAP (Simple Object Access Protocol).
2. **Library/Framework APIs**: Provide a set of functions and procedures that allow developers to interact with libraries or frameworks.
3. **Operating System APIs**: Allow applications to interact with the operating system. For example, Windows API for interacting with Windows OS features.
4. **Database APIs**: Allow communication with databases, such as SQL-based APIs for querying databases.

What is RestAPI?

A REST API (Representational State Transfer Application Programming Interface) is a web service that allows applications to communicate over the internet using standard HTTP methods. It follows REST principles, which emphasize scalability, statelessness, and resource-based interactions.

Key Features of REST API:

1. **Stateless** – Each request from a client to the server must contain all necessary information, and the server does not store client state.
2. **Client-Server Architecture** – The client and server are separate, allowing independent development and scaling.
3. **Resource-Based** – Everything is treated as a resource (e.g., users, products), identified by unique URLs.
4. **Uses Standard HTTP Methods**:
    - **GET** – Retrieve data.
    - **POST** – Create a new resource.
    - **PUT** – Update an existing resource.
    - **DELETE** – Remove a resource.
5. **Uses JSON or XML** – Data is commonly exchanged in **JSON (JavaScript Object Notation)**, but XML is also supported.
6. **Stateless Communication** – Each API request is independent, with no stored session on the server.
7. **Layered System** – REST API architecture can have multiple layers (e.g., security, caching) without affecting client-server interactions.

Example of REST API Request & Response:

1. GET Request (Fetching user data)

Request:

```
vbnet
CopyEdit
GET /users/1 HTTP/1.1
Host: example.com
```

Response (JSON format):

```
json
CopyEdit
{
    "id": 1,
    "name": "John Doe",
    "email": "johndoe@example.com"
}
```

2. POST Request (Creating a new user)

Request:

```
bash
CopyEdit
POST /users HTTP/1.1
Host: example.com
Content-Type: application/json

{
    "name": "Jane Doe",
    "email": "janedoe@example.com"
}
```

Response:

```
json
CopyEdit
{
    "id": 2,
    "name": "Jane Doe",
    "email": "janedoe@example.com",
    "message": "User created successfully"
}
```

Common Use Cases of REST APIs

- Web and mobile application backend communication
- Integration between different software applications
- Cloud-based services and IoT applications
- Data fetching for front-end applications (React, Angular, Vue.js)

RestAPI Status Codes:

200 OK	Request was successful.

201 Created	A new resource was successfully created.
204 No Content	Request was successful, but there is no content to return.
400 Bad Request	Invalid request from the client.
401 Unauthorized	Authentication is required.
403 Forbidden	Access to the resource is not allowed.
404 Not Found	Requested resource does not exist.
500 Internal Server Error	Server encountered an error.

What is an HTTP Status Code?

An **HTTP status code** is a **three-digit number** returned by a web server in response to a request made by a client (such as a web browser or API). These codes indicate whether a request was successful, redirected, encountered an error, or requires further action.