Date:06/02/2025

Day 4:

What is Rest API?

A **REST API (Representational State Transfer API)** is a way for different software applications to communicate over the internet using standard web protocols. It follows REST principles, which emphasize scalability, simplicity, and statelessness.

1. Stateless: Each request from a client to a server must contain all the information needed to process the request. The server does not store client session data.

2. Client-Server Architecture: The client and server are independent, meaning the front-end (UI) and back-end (database, logic) can evolve separately.

3. Resource-Based: Data is treated as resources (e.g., users, products, orders) and is accessed using HTTP methods.

4. It Uses **HTTP** Method:

- **GET** – Retrieves data from a server (read-only, no modification).
- **POST** – Sends data to the server to create a new resource.
- **PUT** – Updates or replaces an existing resource.
- **DELETE** – Removes a resource from the server.
- **PATCH** – Partially updates an existing resource.

JSON and XML….

JSON (JavaScript Object Notation)

- A lightweight data-interchange format.

- Uses key-value pairs and is easy to read and write.
- Example:

```
{
  "name": "Pavan",
  "age": 21,
  "skills": ["Python", "Java", "Blockchain"]
}
```

XML (eXtensible Markup Language)

- A markup language used for storing and transporting data.
- Uses a hierarchical structure with opening and closing tags.
- Example:

```
<person>
  <name>Pavan</name>
  <age>21</age>
  <skills>
    <skill>Python</skill>
    <skill>Java</skill>
    <skill>Blockchain</skill>
  </skills>
</person>
```

Differences Between JSON and XML

Feature	JSON	XML
Format	Lightweight and key-value based	Uses tags and attributes
Readability	Easier for humans and machines	More verbose and complex
Data Type Support	Supports numbers, strings, arrays, and objects	Stores all data as text
Parsing Speed	Faster and easier to parse	Slower due to complex structure

Feature	JSON	XML
Usage	Common in APIs (REST, AJAX)	Used in configuration files and SOAP APIs
Extensibility	Limited (fixed structure)	More flexible with schemas

JSON is preferred for most modern web applications due to its simplicity and efficiency.

Status code…..

An **HTTP status code** is a three-digit response code that a server sends to indicate the result of a client's request.

Categories of HTTP Status Codes:

1. **1xx (Informational)** – Request received, continuing process.
    - 100 Continue – Initial request received; continue sending data.
2. **2xx (Success)** – Request was successfully received, understood, and processed.
    - 200 OK – Request succeeded.
    - 201 Created – New resource created.
3. **3xx (Redirection)** – Further action needed to complete the request.
    - 301 Moved Permanently – URL has changed permanently.
    - 302 Found – Temporary redirection.
4. **4xx (Client Errors)** – The request contains bad syntax or cannot be fulfilled.
    - 400 Bad Request – Invalid request from the client.
    - 401 Unauthorized – Authentication required.
    - 403 Forbidden – Access denied.
    - 404 Not Found – Requested resource not found.
5. **5xx (Server Errors)** – The server failed to process a valid request.

- 500 Internal Server Error – Generic server failure.
- 502 Bad Gateway – Invalid response from an upstream server.
- 503 Service Unavailable – Server is overloaded or under maintenance.

URL Structure…

A **URL (Uniform Resource Locator)** is the address of a web resource. It consists of multiple parts:

https://example.com/api/users/123?status=active

1. URL Components

Component	Example	Description
Protocol	https://	Defines the communication method (HTTP/HTTPS).
Base URL	example.com	The domain name or IP address of the server.
Path Parameter	/api/users/123	Identifies a specific resource (e.g., user with ID 123).
Query Parameter	?status=active	Key-value pairs used to filter or modify the request.

---

2. Path Parameter vs. Query Parameter

Parameter Type	Example	Use Case
Path Parameter	/users/{id} → /users/123	Identifies a specific resource (fixed value).
Query Parameter	/users?status=active	Provides extra filters or options (dynamic).

Example Breakdown

Path Parameter Example:

https://api.example.com/users/123

- Fetches user **123** (fixed resource).

Query Parameter Example:

https://api.example.com/users?status=active&sort=asc

- Fetches users with **status = active**, sorted in **ascending order**.

3. Resource

A **resource** is the data being accessed or manipulated in a web request (e.g., users, products, orders).
In the example /api/users/123, **users** is the resource being accessed

Challenges and Limitations in APIs…

APIs come with several challenges and limitations, such as security risks, performance issues, and data consistency. Some key challenges include:

1. **Security Issues** – APIs can be vulnerable to attacks like **SQL injection, DDoS, and unauthorized access**.
2. **Scalability** – Handling a large number of requests efficiently.
3. **Latency** – Ensuring fast responses, especially in distributed systems.
4. **Versioning** – Managing changes without breaking existing integrations.
5. **Rate Limiting** – Preventing excessive API usage to maintain performance.
6. **Error Handling** – Providing clear and consistent error responses.

---

1. Rate Limiting

Rate limiting restricts the number of requests a user or system can make to an API within a specific time frame. This prevents abuse and ensures fair resource usage.

- **Example:** A free-tier API allows only **100 requests per minute** per user.
- **Methods of Rate Limiting:**
  - **Token Bucket** – Users get a limited number of tokens per time period.
  - **Leaky Bucket** – Requests are processed at a constant rate.
  - **Fixed Window** – Limits requests per fixed time interval.

Example Response When Rate Limited:

```
{
  "error": "Too many requests",
  "status": 429,
  "retry_after": "60 seconds"
}
```

2. Versioning

API versioning allows updates and changes to be introduced without breaking existing integrations.

- **Methods of API Versioning:**
  1. **URI Versioning:**
     - Example: https://api.example.com/v1/users
  2. **Header Versioning:**
     - Example: Accept: application/vnd.example.v2+json
  3. **Query Parameter Versioning:**
     - Example: https://api.example.com/users?version=2

Using versioning ensures backward compatibility and allows smooth upgrades.

---

3. Error Handling

Error handling ensures that APIs provide meaningful responses when things go wrong.

Common HTTP Error Codes in APIs:

Status Code	Meaning	Example Scenario
400 Bad Request	Invalid request format	Missing required parameters
401 Unauthorized	Authentication failed	Invalid API key or token
403 Forbidden	Access denied	No permission to access a resource
404 Not Found	Resource doesn't exist	Incorrect API endpoint
500 Internal Server Error	Server-side issue	API service failure

Example Error Response:

```
{
  "error": "Invalid API key",
  "status": 401,
  "message": "Please provide a valid API key."
}
```

Backword compatibility….

Backward compatibility means that **a new version of a system (software, API, or hardware) continues to support functionality from its previous versions** without breaking existing applications or users.

Example in Different Contexts

1. API Backward Compatibility

If an API is updated, older clients should still be able to interact with it without errors.

✅ **Backward Compatible Change:**

- Adding a new optional field
- Keeping existing fields unchanged
- Supporting older API versions (versioning)

❌ **Breaking Change (Not Backward Compatible):**

- Renaming or removing existing fields
- Changing request/response structure
- Removing support for an older API version

2. Software Backward Compatibility

A new version of software should still run applications built for the older version.

✅ Example: Windows 11 can run applications that were designed for Windows 10.

3. Hardware Backward Compatibility

New hardware should support older systems or components.

✅ Example: A PlayStation 5 can run PlayStation 4 games.

Why is Backward Compatibility Important?

- Prevents breaking existing users' applications
- Reduces migration costs for developers
- Ensures smooth transitions to new versions

Maintaining **backward compatibility** is critical in software development to avoid disruptions when upgrading systems.