REST API

1. **WHAT IS AN API?**

- An API, or Application Programming Interface, is basically a way for two systems to talk to each other. It defines a set of rules or endpoints that allow one application to request data or perform operations on another.
- For example, in a MERN stack app, the frontend (React) might call an Express API to fetch user data from MongoDB. So the API acts as the bridge between the frontend and backend.

2. **WHAT IS REST API? EXPLAIN ITS FEATURES?**

A REST API (Representational State Transfer) is an architectural style for designing networked applications. They are stateless web services operating on resources via standard HTTP method like GET, POST, PUT, DELETE — to perform CRUD operations on resources identified by URLs. (Optional: A web service is a software system designed to communicate and exchange data between different applications or systems, typically over the internet.)

- **Why REST?**
  🔺 Simplicity: Uses standard HTTP protocols, making it straightforward to implement and consume.
  🔺 Scalability: Its stateless nature means each request contains all necessary information so the servers need not communicate much with each other while serving requests.
  🔺 Separation of Concerns: Decouples the client and server, allowing them to evolve independently.
- **Features of RESTful Web Services:**
  - RESTful web services follow a set of principles that make communication between client and server really clean and scalable. Some core features are:
    - **Statelessness**

    - Each request from the client contains all the info needed. The server doesn’t store any session data between requests. *(Like: if I make two requests back-to-back, the server treats them independently.)*
  - **Client-Server Architecture**
    - The frontend and backend are totally separate. The client doesn’t care how the server handles the logic, and the server doesn’t care how the UI looks.
  - **Uniform Interface**
    - There’s a consistent way to interact with resources. For example, we use standard HTTP methods:
      - `GET` → fetch data
      - `POST` → create new data
      - `PUT/PATCH` → update existing data
      - `DELETE` → remove
  - **Resource-Based**
    - Everything is treated as a resource—like `/users` . These resources are represented by URIs.
  - **Representation in JSON**
    - Data is typically sent and received in a format like JSON, which makes it easier for frontend to parse and render.
  - **Cacheable**
    - API Responses can be cached on the client side to improve performance and reducing the need to repeatedly fetch data from the server.

3. **WHAT IS URI?**

So URI stands for Uniform Resource Identifier. It’s basically a string of characters that uniquely identifies a resource — like a webpage, an image, an API endpoint, anything. For example: It is generally used in XML (or Extensible Markup Language, is a markup language designed to store and transport data in a structured way, It's similar to HTML, but unlike HTML, it doesn't have predefined tags)

- A URL (Uniform Resource Locator) is a specific type of URI that provides the location of a resource on the internet.

4. **EXPLAIN STATELESSNESS CONCEPT IN REST?**

- So the concept of statelessness in REST means that every single request from the client to the server is completely independent. There's **no memory of previous requests**—so the server doesn't store any session data between calls. Each request must carry everything the server needs to fulfill it—usually in the form of headers, parameters, or tokens like an `access_token`.
- ❓ **Why is REST stateless?:**
  - Because it improves scalability and simplicity.
  - Servers don't need to track client state → they just respond to each request as it comes.
  - That makes it easier to load balance and cache responses.
  Like if we imagine 1 million users hitting our API— and if we had to maintain session info for all of them, the server would get overloaded.
- **But what about SOAP?:**
  - SOAP is **stateful by default**, or at least it supports it.
  - It can maintain context over multiple requests using things like sessions or message headers.
- **Disadvantages of REST:**
  - As REST follows the idea of statelessness, it is not possible to maintain sessions. (Session simulation responsibility lies on the client-side to pass the session id)

5. **WHAT IS HTTP AND HOW IT WORKS?**

HTTP is a **stateless protocol** used for **request-response communication** between a **client (browser/frontend)** and a **server (backend)**. HTTP request methods define **what action** the client wants to perform on the resource.

- *HTTP Methods:*
  - **GET**: **Retrieve** data from the server
    - **POST**: Send data to the server to **create** a new resource
    - **PUT**: Used to **replace** and **existing resource entirely**.

    - **PATCH**: Used to **update** a part of **existing resource**
    - **DELETE**: Used to **delete a resource**

6. **WHAT ARE HTTP CODES?**

These are the standard codes that refer to the predefined status of the task at the server.
Most commonly used status codes are:

- 200 - success/OK
- 201 - CREATED - used in POST or PUT methods.
- 400 - BAD REQUEST - This can be due to validation errors or missing input data.
- 401 - FORBIDDEN - sent when the user does not have access (or is forbidden) to the resource.
- 404 - NOT FOUND - Resource method is not available.
- 500 - INTERNAL SERVER ERROR - server threw some exceptions while running the method.
- 502 - BAD GATEWAY - Server was not able to get the response from another upstream server.

7. **DIFFERENCE BETWEEN PUT AND POST IN REST?**

So both `POST` and `PUT` are used to send data to the server, but they're meant for different intentions.

- **POST:**
    - We use `POST` when we are sending data to the server and expecting it to create a new resource.
    - It's **non-idempotent**, meaning if you hit it multiple times with the same data, it'll create **multiple** resources.
    - Target URL is general like in below example we are using `/students`.
    - ex:

```
POST /api/students
Body: { "name": "Ankit", "email": "ankit@example.com" }
```

- **PUT:**
    - `PUT` is idempotent. Hitting it once or 10 times with the same data results in the **same outcome**.
    - Usually used to update an existing resource, but if the resource doesn't exist, it can create one.
    - Target URL is specific like in below example we are using `/students/123`.
    - ex:

```
PUT /api/students/123
Body: { "name": "Ankit", "email": "ankit@example.com" }
```

8. **WHAT IS MAX PAYLOAD SIZE THAT CAN BE SENT IN POST REQUESTS?**

Theoretically, there is no restriction on the size of the payload that can be sent. But its important to remember that the greater the size of the payload, the larger would be the bandwidth consumption and time taken to process the request.

9. **HOW DOES HTTP BASIC AUTHENTICATION WORK?**

So when a client makes a request to the server, it sends the user's credentials (usually username and password) encoded in **Base64** inside the `Authorization` header. The server decodes that header, checks if the credentials are valid, and if they are, it allows access.

- It's **not encrypted**, just encoded—so if someone intercepts the request, they can easily decode your credentials.
- That's why **Basic Auth should always be used over HTTPS**—never plain HTTP.
- It's stateless—no sessions, no tokens. Just send creds every time.

10. **DIFFERENCE BETWEEN AUTHORIZATION AND AUTHENTICATION?**

So, authentication and authorization might sound similar but they're totally different steps in security.

- 🔐 **Authentication → ‘Who are you?’**
    - It’s the process of **verifying identity**.
    - Like when a user logs in using a username/password or via Google OAuth, that’s authentication.
    - Once the system knows *who* you are, only then can it move to the next step.
    - We can just think of it like showing our college ID card to the gatekeeper.
- ✅ **Authorization → ‘What are you allowed to do?’**
    - Once you're authenticated, authorization checks **what you're allowed to access**.
    - For example, an admin user might be able to access `/api/admin/students`, but a normal user can’t.
    - We can just think of this as well like getting into the college building (which represents 'authentication'), but only staff can enter the faculty room (which represent 'authorization').

11. **WHAT ARE IDEMPOTENT METHODS? HOW IS IT RELEVANT IN RESTFUL WEB SERVICES DOMAIN?**

- So, idempotent methods are HTTP methods that can be called multiple times without changing the result beyond the first call. Basically, no matter how many times I hit the same API with the same request, the **state of the server stays the same.**
- **Examples of Idempotent Methods in REST:**
    - `GET`:
        - Just fetches data. Doesn’t change anything.
        - Hitting `/api/user/123` five times gives the same result, no side effects.
- **Why does this matter in REST?**
    - Idempotency is important for **reliability** and **safety** in distributed systems. Like:
        - Let’s say a network glitch makes the client resend a request—idempotent methods ensure the server doesn't process unintended duplicates.

- It makes RESTful APIs more predictable and robust.