

## **Question 1: What do you mean by RESTful web services?**

**Ans :** RESTful Web Services (Representational State Transfer) are a way to design and create web-based APIs that allow systems to communicate with each other over the HTTP protocol in a simple, stateless, and standardized manner.

### **Definition:**

A RESTful web service is an API that follows the principles of REST architecture to perform CRUD operations (Create, Read, Update, Delete) on resources using standard HTTP methods.

### **Key Principles of RESTful Web Services:**

#### **1. Client-Server Architecture:**

The client (frontend) and server (backend) are separate.

- Client: sends requests (e.g., browser, mobile app).
- Server: processes requests and sends back responses (usually JSON or XML).

#### **2. Statelessness:**

Each request from a client must contain all the information needed —

the server does **not store client session data**.

#### **3. Cacheable:**

Responses can be cached to improve performance.

## **Question 2: What is Json-Server? How we use in React ?**

**Ans :** JSON Server is a simple tool that lets you create a fake RESTful API using a JSON file as a database.

It's mainly used for testing, prototyping, and front-end development when you don't have a real backend yet.

**In simple terms:**

JSON Server = Fake backend made from a JSON file

**Example:**

```
{  
  "users": [  
    { "id": 1, "name": "Raj", "email": "raj@gmail.com" },  
    { "id": 2, "name": "Amit", "email": "amit@gmail.com" }  
  ]  
}
```

**Question 3: How do you fetch data from a Json-server API in React? Explain the role of `fetch()` or `axios()` in making API requests.**

**Ans :** When you run **JSON Server**, it gives you RESTful API endpoints like: <http://localhost:3001/users>

To use this data inside a React component, you need to **fetch it from the API**.

That's done using either **fetch()** (a built-in JavaScript function) or **axios** (a popular library).

### **Step-by-Step Explanation**

Start your JSON Server

```
json-server --watch db.json --port 3001
```

#### **Use `useEffect()` and `useState()` in React**

`useState()` → store the data you get from API

`useEffect()` → fetch data when the component loads

#### **In short:**

You fetch data from a JSON Server API in React using either `fetch()` or `axios()` inside `useEffect()`.

The function sends an HTTP request (GET, POST, PUT, DELETE) to your API and updates React state with the received data.

#### **In short:**

You fetch data from a JSON Server API in React using either `fetch()` or `axios()` inside `useEffect()`.

The function sends an HTTP request (GET, POST, PUT, DELETE) to your API and updates React state with the received data.

#### **Question 4: What is Firebase? What features does Firebase offer?**

**Ans :** Firebase is a Backend-as-a-Service (BaaS) platform developed by Google.

It helps developers build web and mobile apps without managing servers.

It provides ready-made backend tools like:

- **Database**
- **Authentication**
- **Hosting**
- **Storage**
- **Notifications**
- **Analytics**

So you can focus on your **frontend (React, Android, iOS, etc.)** while Firebase handles the backend.

#### **In Simple Terms:**

Firebase = Google's ready-made backend platform for building, hosting, and managing apps easily.

## **Question 5: Discuss the importance of handling errors and loading states when working with APIs in React.**

**Ans :** When your React app fetches data from an API (like a JSON Server, Firebase, or backend), the data doesn't come instantly —

it takes time and can sometimes fail.

That's why we must handle loading and error states properly.

A **loading state** indicates that the app is **waiting for a response** from the API.

### **Why it matters:**

- Improves **user experience (UX)** — users see that something is happening.
- Prevents showing empty screens or broken UI.
- Helps manage multiple requests smoothly.

### **Error State**

An **error state** handles what happens if the API call fails (e.g., network error, wrong URL, server down).

### **Why it matters:**

- Prevents app crashes.
- Informs users something went wrong.
- Helps developers debug issues easily.