# **Json – Server & Firebase Real Time Database**

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

**Ans:** RESTful web services are web services that adhere to the principles of REST (Representational State Transfer). REST is an architectural style used to build scalable and lightweight APIs. RESTful services use HTTP methods (GET, POST, PUT, DELETE, etc.) to perform CRUD (Create, Read, Update, Delete) operations on resources, which are typically represented as URLs.

**Key features of RESTful services:**

1. **Stateless:** Each request from a client contains all the information needed for the server to process it, and no client-specific session is stored on the server.
2. **Resource-based:** Resources are represented as URIs (e.g., /users, /products).
3. **Uses standard HTTP methods:** GET (read), POST (create), PUT/PATCH (update), DELETE (delete).
4. **Data Format:** Typically uses JSON or XML for request/response payloads.

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

**Ans:** Json-Server is a lightweight, mock REST API server that allows you to create a RESTful API with minimal configuration. It uses a db.json file to simulate a backend database and supports GET, POST, PUT, DELETE, and PATCH requests.

**How to use Json-Server in React:**

**Steps 1**

**Install Json-Server:**

npm install -g json-server

**Steps 2**

**Create a db.json file with sample data:**

{

"users": [

{ "id": 1, "name": "nilu" },

{ "id": 2, "name": "nitu" }

]

}

**Steps 3**

**Start the server:**

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

**Steps 4**

In React, fetch data from the Json-Server API (e.g., http://localhost:3001/users) using fetch() or axios.

**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:** To fetch data from a Json-Server API in React, you can use fetch() or axios():

**Using fetch():**

import {useEffect , useState} from "react";

function App() {

const [data, setData] = useState([]);

useEffect(() => {

fetch("http://localhost:3001/users")

.then((response) => response.json())

.then((data) => setData(data))

.catch((error) => console.error("Error:", error));

}, []);

return (

<ul>

{data.map((user) => (

<li key={user.id}>{user.name}</li>

))}

</ul>

);

}

**Using axios():**

import axios from "axios";

import { useEffect, useState } from "react";

function App() {

const [data, setData] = useState([]);

useEffect(() => {

axios

.get("http://localhost:3001/users")

.then((response) => setData(response.data))

.catch((error) => console.error("Error:", error));

}, []);

return (

<ul>

{data.map((user) => (

<li key={user.id}>{user.name}</li>

))}

</ul>

);

}

**Role of fetch() or axios():**

* **fetch()**: A built-in browser API for making HTTP requests. It returns a promise that resolves to the Response object.
* **axios()**: A third-party library for making HTTP requests. It provides more features than fetch(), like automatic JSON parsing, better error handling, and request/response interceptors.

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

**Ans:** Firebase is a Backend-as-a-Service (BaaS) platform by Google that provides tools for building and managing web and mobile applications.

**Key features of Firebase:**

1. **Realtime Database**: A NoSQL cloud-hosted database for syncing data in real time.
2. **Firestore**: A scalable and flexible NoSQL database.
3. **Authentication**: Provides user authentication services (email/password, Google, Facebook, etc.).
4. **Hosting**: Deploy and host web apps on fast and secure global servers.
5. **Storage**: Store and retrieve user-generated files like images and videos.
6. **Cloud Functions**: Run serverless backend code in response to events.
7. **Crashlytics**: Real-time crash reporting and diagnostics for mobile apps.
8. **Push Notifications**: Send notifications to app users via Firebase Cloud Messaging (FCM).

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

**Ans:Importance of Handling Errors:**

1. **User Feedback**: Proper error handling ensures users are informed when something goes wrong (e.g., network failure or server error).
2. **Improved UX:** Helps maintain a seamless user experience by displaying appropriate error messages or fallback options.
3. **Debugging:** Logging errors helps developers identify and fix issues.
4. **Security:** Prevents sensitive error information from being exposed to users.

**Importance of Handling Loading States:**

1. **Visual Feedback:** Indicates to users that the app is processing their request, reducing confusion.
2. **Prevent Unnecessary Actions:** Disables UI elements like buttons while the data is being fetched, avoiding multiple requests.

**Example: Handling Errors and Loading States:**

import { useEffect, useState } from "react";

function App() {

const [data, setData] = useState([]);

const [loading, setLoading] = useState(true);

const [error, setError] = useState(null);

useEffect(() => {

fetch("http://localhost:3001/users")

.then((response) => {

if (!response.ok) {

throw new Error("Failed to fetch data");

}

return response.json();

})

.then((data) => setData(data))

.catch((error) => setError(error.message))

.finally(() => setLoading(false));

}, []);

if (loading) return <p>Loading...</p>;

if (error) return <p>Error: {error}</p>;

return (

<ul>

{data.map((user) => (

<li key={user.id}>{user.name}</li>

))}

</ul>

);

}