React – JSON-server and Firebase Real Time Database

**Question 1: What do you mean by RESTful Web Services?**

**RESTful Web Services** follow the **REST (Representational State Transfer)** architecture. They allow communication between client and server over HTTP using standard methods like:

* GET – Read data
* POST – Create data
* PUT/PATCH – Update data
* DELETE – Delete data

**Key Features:**

* Stateless: Each request is independent.
* Resource-based: Uses URLs to access resources (e.g., /products/1)
* Uses JSON or XML for data exchange (mostly JSON today)

**Example:**

http

CopyEdit

GET https://api.example.com/products

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

**Json-Server** is a **fake REST API** tool used during development. It turns a JSON file into a full API with **CRUD** (Create, Read, Update, Delete) operations.

**Why use it?**

* For testing API calls without a backend
* Quick prototyping

**Installation:**

bash

CopyEdit

npm install -g json-server

**Start the server:**

bash

CopyEdit

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

**Using in React:**

* In React, make API calls (using axios or fetch) to http://localhost:3001/products (assuming products array is in your db.json).

**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.**

**Steps:**

1. Install axios or use built-in fetch()
2. Use useEffect() to make the call on component mount
3. Store the data using useState()

**Example using fetch():**

js

CopyEdit

useEffect(() => {

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

.then(res => res.json())

.then(data => setProducts(data))

.catch(err => console.error(err));

}, []);

**Example using axios:**

js

CopyEdit

import axios from "axios";

useEffect(() => {

axios.get("http://localhost:3001/products")

.then(res => setProducts(res.data))

.catch(err => console.error(err));

}, []);

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

* fetch() is native to JavaScript and returns promises
* axios() is a library that simplifies API calls with features like interceptors, automatic JSON conversion, and better error handling

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

**Firebase** is a **Backend-as-a-Service (BaaS)** platform by Google that provides ready-to-use backend services for web and mobile apps.

**Main Features:**

1. **Authentication** – Google, Facebook, Email/Password login
2. **Realtime Database** – Store and sync data in real-time
3. **Cloud Firestore** – Scalable NoSQL database
4. **Firebase Storage** – Upload and store images/files
5. **Hosting** – Deploy web apps easily
6. **Cloud Functions** – Serverless backend logic
7. **Analytics, Crashlytics, Push Notifications** – For monitoring and user engagement

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

**Why handle loading and error states?**

* **Good UX**: Users should see a loader or message while data is loading
* **Feedback on failures**: Show proper error messages if something goes wrong (like no internet)

**How to handle:**

1. Use useState() to manage loading, error, and data states
2. Set loading = true before the request
3. Set loading = false and update data after success
4. If error occurs, set error = true or store error message

**Example:**

js

CopyEdit

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

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

useEffect(() => {

axios.get("http://localhost:3001/products")

.then(res => {

setProducts(res.data);

setLoading(false);

})

.catch(err => {

setError(err.message);

setLoading(false);

});

}, []);

**Render logic:**

jsx

CopyEdit

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

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

LAB EXERCISE

**✅ Task 1: React + JSON Server (CRUD with Table View)**

**1.1 Display Public API Data in Table**

Use https://jsonplaceholder.typicode.com/users (a free user API)

**Steps:**

1. Create a component (UsersTable.js)
2. Use useEffect() + axios or fetch to call API
3. Display data in an HTML <table>

**1.2 Setup JSON Server**

1. Create a db.json file:

json

CopyEdit

{

"users": [

{ "id": 1, "name": "John", "email": "john@example.com" }

]

}

1. Run the server:

bash

CopyEdit

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

**1.3 CRUD with JSON Server**

In your React app:

* **GET** – Fetch and display users
* **POST** – Add user form
* **PUT/PATCH** – Update user
* **DELETE** – Remove user

Use axios.post, axios.put, axios.patch, axios.delete to perform actions on http://localhost:3001/users

**✅ Task 2: React App + Firebase CRUD & Auth**

**2.1 Firebase Setup**

1. Create Firebase project
2. Enable **Firestore**, **Authentication** (Email/Password + Google)
3. Get config and add to React app

**2.2 Firebase CRUD**

Use Firestore for:

* Create (addDoc)
* Read (getDocs)
* Update (updateDoc)
* Delete (deleteDoc)

**2.3 Authentication**

* Use Firebase signInWithEmailAndPassword, createUserWithEmailAndPassword
* Implement **Google login** with signInWithPopup()

**✅ Task 3: Error Handling & Loading Spinner**

**3.1 States**

js

CopyEdit

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

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

**3.2 Spinner Logic**

jsx

CopyEdit

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

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

**3.3 Integration**

Use try/catch or .catch() to handle errors in API call, and toggle loading states.

If you want, I can create all 3 projects’ folder structures, components, and file organization for you step-by-step. Let me know which one you want to start with:

* Task 1 (Public API + JSON-server CRUD)
* Task 2 (Firebase CRUD + Google Auth)
* Task 3 (Loading/Error spinner logic)