React – JSON-server and Firebase Real Time Database

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

Ans.

RESTful Web Services are web services that follow the principles of REST (Representational State Transfer) to enable communication between a client and a server over the internet, usually using the HTTP protocol.

Simple Definition :

A RESTful web service allows different applications to communicate with each other using standard web methods like GET, POST, PUT, and DELETE in a lightweight and scalable way.

Key Principles of REST

1. **Client–Server Architecture**
   - Client handles UI
   - Server handles data & business logic
2. **Statelessness**
   - Each request from the client contains all the information needed
   - Server does not store client state
3. **Uniform Interface**
   - Resources are accessed using **URLs**
   - Uses standard HTTP methods

4. **Resource-Based**
    - Everything is treated as a **resource**
    - Example:

        /users

        /products/10

5. **Representation**
    - Resources are represented using formats like **JSON** or **XML**
    - JSON is most common

Common HTTP Methods in REST

Method	**Purpose**
GET	Retrieve data
POST	Create new data
PUT	Update entire data
PATCH	Update partial data
DELETE	Remove data

Example of a RESTful API :

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

Why RESTful Web Services Are Used

- Platform independent
- Lightweight and fast
- Easy to understand and implement
- Widely used in React, Angular, Mobile Apps
- Works well with microservices

Real-World Example

- Fetching users in a React app from a backend API
- Payment gateway APIs
- Social media APIs (Instagram, Twitter/X)

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

Ans.

JSON Server is a lightweight fake REST API tool that allows developers to quickly create a mock backend using a simple JSON file. It is commonly used for frontend development and testing when a real backend is not ready. It provides full CRUD operations (Create, Read, Update, Delete) with zero backend coding.

Why Use JSON Server?

- No real backend required
- Quickly create RESTful APIs

- Ideal for React / frontend projects
- Supports GET, POST, PUT, PATCH, DELETE
- Returns data in JSON format

How JSON Server Works

You define data in a file called db.json, and JSON Server automatically creates REST APIs for it.

Example db.json:

```
{
 "users": [
  { "id": 1, "name": "Uday", "age": 22 },
  { "id": 2, "name": "Parmar", "age": 24 }
 ]
}
```

APIs created automatically:

- GET /users
- GET /users/1
- POST /users
- PUT /users/1
- DELETE /users/1

How to Install JSON Server :

```
npm install -g json-server
```

or (recommended for project)

```
npm install json-server --save-dev
```

Run JSON Server:

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

How We Use JSON Server in React :

```
import { useEffect, useState } from "react";

function Users() {
  const [users, setUsers] = useState([]);

  useEffect(() => {
    fetch("http://localhost:3001/users")
      .then(res => res.json())
      .then(data => setUsers(data));
  }, []);
  return (
    <ul>
      {users.map(user => (
        <li key={user.id}>{user.name}</li>
      ))}
    </ul>
  );
}

export default Users;
```

2. Add Data (POST)

```
fetch("http://localhost:3001/users", {
  method: "POST",
  headers: {
    "Content-Type": "application/json"
  },
  body: JSON.stringify({ name: "New User", age: 20 })
});
```

3. Update Data (PUT / PATCH)

```
fetch("http://localhost:3001/users/1", {
  method: "PATCH",
  headers: {
    "Content-Type": "application/json"
  },
  body: JSON.stringify({ age: 25 })
});
```

4. Delete Data (DELETE)

```
fetch("http://localhost:3001/users/1", {
  method: "DELETE"
});
```

JSON Server in Real React Projects

- Used during UI development
- Helpful for learning API integration
- Perfect for assignments, demos, interviews
- Later replaced with real backend (Node, Spring, Django)

Limitations of JSON Server

- Not for production use
- No authentication by default
- Limited business logic

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.

In React, data from a JSON Server API is typically fetched using fetch() (built-in JavaScript API) or axios (a popular third-party library). Both are used to make HTTP requests to the backend and retrieve data asynchronously.

1. Using fetch() in React

```
import { useEffect, useState } from "react";

function Users() {
 const [users, setUsers] = useState([]);

 useEffect(() => {
  fetch("http://localhost:3001/users")
   .then(response => response.json())
   .then(data => setUsers(data))
   .catch(error => console.error(error));
 }, []);
```

```
  return (
    <ul>
      {users.map(user => (
        <li key={user.id}>{user.name}</li>
      ))}
    </ul>
  );
}

export default Users;
```

Role of fetch()

- Built-in browser API
- Sends HTTP requests (GET, POST, PUT, DELETE)
- Returns a **Promise**
- Requires manual JSON conversion using .json()
- Requires explicit error handling

2. <u>Using axios in React :</u>

Install Axios :

```
npm install axios
```

```
import { useEffect, useState } from "react";
import axios from "axios";
```

```
function Users() {
  const [users, setUsers] = useState([]);

  useEffect(() => {
    axios.get("http://localhost:3001/users")
      .then(response => setUsers(response.data))
      .catch(error => console.error(error));
  }, []);

  return (
    <ul>
      {users.map(user => (
        <li key={user.id}>{user.name}</li>
      ))}
    </ul>
  );
}

export default Users;
```

Role of axios

- External library
- Automatically converts response to JSON
- Cleaner syntax
- Better error handling
- Supports request/response interceptors

3. Why useEffect() is Used

- API calls are side effects
- Prevents infinite re-rendering
- Runs once when component mounts (empty dependency array [])

4. fetch vs axios (Quick Comparison)

Feature	**fetch()**	**axios**
Built-in	✅	❌
JSON parsing	Manual	Automatic
Error handling	Basic	Better
Interceptors	❌	✅
Request canceling	❌	✅

5. Real JSON Server Example

GET http://localhost:3001/users

Response:

```
[
 { "id": 1, "name": "Uday" },
 { "id": 2, "name": "Parmar" }
]
```

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

Ans.

Firebase is a Backend-as-a-Service (BaaS) platform by Google that helps developers build, run, and scale web and mobile applications without managing backend infrastructure. It provides ready-to-use backend services like databases, authentication, hosting, and notifications, allowing frontend developers (React, Angular, Android, iOS) to focus on UI and business logic.

Key Features of Firebase :

1. Realtime Database

- NoSQL, cloud-hosted database
- Data syncs in real time across all connected clients
- Works with JSON data
- Offline support

2. Cloud Firestore

- Advanced NoSQL document database
- Scalable and flexible
- Real-time updates
- Better querying than Realtime Database
- Supports collections & documents

3. Firebase Authentication

Easy user authentication

Supports:

- Email & Password
- Phone Authentication
- Google, Facebook, GitHub, Twitter login

Secure and production-ready

4. Firebase Hosting

- Fast, secure web hosting
- Free SSL certificate
- Ideal for React, Angular, Vue apps
- One-command deployment

5. Cloud Functions

Serverless backend logic

Run code in response to events

Used for backend APIs, triggers, validations

6. Cloud Storage

Store images, videos, documents

Secure file access with rules

Commonly used for profile pictures

7. Firebase Cloud Messaging (FCM)

Push notifications

Web and mobile support

Used for alerts and reminders

8. Analytics

Track user behavior

App usage insights

Integrates with Google Analytics

9. Performance Monitoring

Monitor app performance

Identify slow API calls

10. Crashlytics

Real-time crash reporting

Helps fix bugs faster

Firebase in React (Simple Use Case)

- Authentication (Login / Signup)
- Store user data in Firestore
- Host React app

- Send notifications

Advantages of Firebase

- No backend server management
- Scales automatically
- Easy integration with React
- Free tier available

Limitations of Firebase

- Vendor lock-in
- Limited complex querying
- Not ideal for heavy relational data

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

Ans.

When working with APIs in React, requests are asynchronous. Properly handling loading and error states is essential to ensure a smooth user experience, better reliability, and maintainable code.

1. <u>Why Handling Loading State Is Important</u>

What is a Loading State?

A loading state indicates that data is being fetched and the UI should inform the user to wait.

Why It Matters

- Prevents blank or confusing screens
- Improves user experience (UX)
- Avoids showing incomplete or stale data
- Gives feedback that the app is working

Example

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


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

2. Why Handling Error State Is Important

   What is an Error State?

   An error state captures failures such as:

- Network issues
- Server errors (404, 500)
- Invalid responses

Why It Matters

- Prevents app crashes
- Shows meaningful error messages
- Helps debugging
- Builds user trust

Example

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

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

3. <u>Combined Example (Loading + Error + Data)</u>

```
import { useEffect, useState } from "react";

function Users() {
  const [users, setUsers] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);

  useEffect(() => {
    fetch("http://localhost:3001/users")
      .then(res => {
        if (!res.ok) throw new Error("Failed to fetch data");
        return res.json();
      })
      .then(data => setUsers(data))
      .catch(err => setError(err.message))
      .finally(() => setLoading(false));
  }, []);

  if (loading) return <p>Loading users...</p>;
  if (error) return <p>{error}</p>;

  return (
    <ul>
      {users.map(user => (
        <li key={user.id}>{user.name}</li>
      ))}
    </ul>
  );
}
```

4. Benefits of Handling Loading & Error States

Better User Experience

- Users know what's happening
- Clear feedback for success/failure

Prevents UI Issues

- Avoids undefined or null errors
- Prevents rendering before data is ready

Improves App Reliability

- Graceful failure handling
- No app crashes due to API issues

Easier Debugging & Maintenance

- Clear error messages
- Better monitoring and logs

5. Best Practices

Always show a loading indicator

Display user-friendly error messages

Handle HTTP status codes

Reset error state on retry

Use try-catch with async/await
