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

RESTful web services are web APIs that follow the **REST (Representational State Transfer)** architectural style. RESTful APIs communicate over HTTP and allow clients to perform CRUD (Create, Read, Update, Delete) operations using standard HTTP methods:

- **GET** – Retrieve data
- **POST** – Create new data
- **PUT/PATCH** – Update existing data
- **DELETE** – Remove data

Key principles of RESTful services:

1. **Stateless** – Each request is independent; the server does not store client session data.
2. **Resource-based** – Data is represented as resources (e.g., `/users`, `/products`).
3. **Uses JSON/XML** – REST APIs commonly return data in JSON format.
4. **Client-Server architecture** – Separation of client and server concerns.
5. **Uniform Interface** – Uses standard HTTP methods and response codes.

Example of a REST API endpoint:

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

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

JSON-Server is a lightweight **fake REST API** for quick prototyping and testing. It allows developers to create a mock API from a JSON file without setting up a full backend.

How to use JSON-Server in React?

Install JSON-Server:

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

Create a `db.json` file in your project:

```
{
  "users": [
    { "id": 1, "name": "John Doe", "email": "john@example.com" },
    { "id": 2, "name": "Jane Doe", "email": "jane@example.com" }
  ]
}
```

Start the server:
`json-server --watch db.json --port 5000`

1. The API will be available at `http://localhost:5000/users`.

Using JSON-Server in React:

You can fetch data from JSON-Server using `fetch()` or `axios()`:

```
useEffect(() => {
  fetch("http://localhost:5000/users")
    .then(res => res.json())
    .then(data => setUsers(data))
    .catch(err => console.error(err));
}, []);
```

Question 3: How do you fetch data from a JSON-server API in React? Explain fetch() or axios() in making API requests.

To fetch data from a JSON-server API in React, you can use **fetch()** or **axios()**.

Using fetch():

`fetch()` is a built-in JavaScript method for making HTTP requests.

```
useEffect(() => {
  fetch("http://localhost:5000/users")
    .then(response => response.json())
    .then(data => setUsers(data))
    .catch(error => console.error("Error fetching data:", error));
}, []);
```

- **Pros:** Native to JavaScript, no need to install extra packages.
- **Cons:** More complex error handling.

Using axios():

`axios` is a third-party library that simplifies HTTP requests.

```
import axios from "axios";

useEffect(() => {
  axios.get("http://localhost:5000/users")
    .then(response => setUsers(response.data))
    .catch(error => console.error("Error fetching data:", error));
}, []);
```

- **Pros:** Supports automatic JSON conversion, better error handling.
- **Cons:** Requires installation (`npm install axios`).

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

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

Features of Firebase:

1. **Authentication** – Supports Google, Facebook, Email/Password login.
2. **Firestore Database** – NoSQL cloud database for real-time data storage.
3. **Realtime Database** – Stores and syncs data in real-time.
4. **Cloud Storage** – Stores and serves user-generated content (e.g., images, videos).
5. **Hosting** – Deploy web apps quickly on Firebase servers.
6. **Cloud Functions** – Serverless backend logic using JavaScript.
7. **Push Notifications** – Send real-time messages to users.
8. **Analytics & Crashlytics** – Monitor app performance and crashes.

Example use case: Firebase Authentication for user login:

```
import { getAuth, signInWithEmailAndPassword } from "firebase/auth";

const auth = getAuth();
signInWithEmailAndPassword(auth, "user@example.com", "password123")
  .then(userCredential => {
    console.log("Logged in:", userCredential.user);
  })
  .catch(error => {
    console.error("Login failed:", error.message);
  });
```

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

When working with APIs in React, handling errors and loading states improves user experience and prevents application crashes.

1. Handling Loading State

- While data is being fetched, show a loading indicator to inform users.

Example:

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

useEffect(() => {
  fetch("http://localhost:5000/users")
    .then(response => response.json())
    .then(data => {
      setUsers(data);
      setLoading(false);
    });
}, []);

return loading ? <p>Loading...</p> : <UserList users={users} />;
```

2. Handling Errors

- API requests may fail due to network issues, invalid URLs, or server errors.

Example error handling:

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

useEffect(() => {
  fetch("http://localhost:5000/users")
    .then(response => {
      if (!response.ok) {
        throw new Error("Failed to fetch data");
      }
      return response.json();
    })
    .then(data => setUsers(data))
```

```
    .catch(error => setError(error.message));
}, []);

return error ? <p>Error: {error}</p> : <UserList users={users} />;
```

3. Using Try-Catch with Axios for Better Handling

```
const fetchData = async () => {
  try {
    const response = await axios.get("http://localhost:5000/users");
    setUsers(response.data);
  } catch (error) {
    setError("Error fetching data");
  } finally {
    setLoading(false);
  }
};
```