Module 18 – Reactjs for Full Stack

11. Routing in React (React Router)

Question 1: What is React Router? How does it handle routing in single-page applications?

- React Router is a routing library for React that enables navigation between different components or views without reloading the page.
- It allows you to build Single Page Applications (SPAs) with multiple views or pages using URL paths.

1. Client-side Routing (no page reloads):
   - In a traditional website, navigating to a new page sends a request to the server.
   - In SPAs using React Router, routing is handled entirely in the browser using JavaScript.
   - The page doesn't reload — only the URL changes and the matching component is rendered.
2. URL Mapping:
   - React Router matches the URL path to a specific component and renders it.
   - Routes are defined using <Route> inside a <BrowserRouter>.
3. Main Components:
   - <BrowserRouter>: Enables routing using HTML5 history API.
   - <Routes>: A container for multiple <Route>s.
   - <Route>: Defines the path and the component to render.
   - <Link>: Replaces <a> tag for client-side navigation without reload.

Feature	Description
Type	Client-side routing
Page reloads	No reload (SPA)
URL-based navigation	Yes
Key components	<BrowserRouter>, <Routes>, <Route>, <Link>
Benefit	Fast navigation and smooth user experience

Question 2: Explain the difference between BrowserRouter, Route, Link, and Switch components in React Router.

1. BrowserRouter

- It is the **main container** that enables routing in your React app using the **HTML5 History API**.
- Wrap your entire app inside it to use routing features.

Purpose: Enables routing in a React app.
Usage: Only **one** BrowserRouter should be used in your app.

2. Route

- Defines a **mapping between a URL path and a component**.
- When the URL matches the path, the specified component is rendered.

Purpose: Display a component based on the current URL.

3. Link

- Used for **client-side navigation** (like an <a> tag but without reloading the page).
- Updates the URL and renders the matched route **without refreshing**.

Purpose: Navigate between routes in the app.

4. Switch (React Router v5 only)

- Renders **only the first <Route>** that matches the current URL.
- Helps prevent multiple components from rendering for one URL.

Purpose: Avoid rendering multiple routes at once.

Component	Purpose	Version
BrowserRouter	Enables routing in the app	v5 & v6
Route	Maps a URL path to a component	v5 & v6
Link	Navigates between routes without reload	v5 & v6
Switch	Renders first matching <Route> only	✅ v5 only
Routes	Replaces Switch in React Router v6	✅ v6 only

12. React – JSON-server and Firebase Real Time Database

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

RESTful web services are web services based on REST (Representational State Transfer) architecture.
They allow different systems or applications to communicate over the web using HTTP protocols.

Key Features of RESTful Web Services:

1. Stateless
   - Each request from a client to the server must contain all necessary information.
   - The server does not store any client context between requests.
2. Uses HTTP Methods
   - GET: Retrieve data
   - POST: Create new data
   - PUT: Update existing data
   - DELETE: Remove data
3. Resource-Based
   - Everything (user, order, product, etc.) is treated as a resource.
   - Each resource is accessed using a unique URI (Uniform Resource Identifier).
     Example: https://api.example.com/users/1
4. JSON or XML Format
   - Data is usually exchanged in JSON or XML, with JSON being more common today.

A simple RESTful API for managing users:

HTTP Method	Endpoint	Description
GET	/users	Get all users
GET	/users/1	Get user by ID
POST	/users	Create a new user
PUT	/users/1	Update user by ID
DELETE	/users/1	Delete user by ID

Benefits of RESTful Web Services

- Scalable and lightweight
- Easy to use with HTTP
- Platform-independent (works with any language or system)
- Widely supported in modern web and mobile apps

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

json-server is a fake REST API server that allows you to create a complete backend using just a JSON file.
It’s useful for mocking APIs during frontend development, especially with React.

Key Features

- Creates a full REST API with CRUD operations (GET, POST, PUT, DELETE)
- No need to write backend code
- Works great for testing and prototyping
- Fast and lightweight

How to Use json-server in a React App:-

1. Install json-server

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

Or as a dev dependency:

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

2. Create a db.json File

```
{
  "users": [
   { "id": 1, "name": "John" },
   { "id": 2, "name": "Alice" }
  ]
}
```

3. Start the JSON Server

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

4. Fetch Data in React

```
import React, { 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;
```

Feature	Description
Purpose	Mock backend for frontend development
Data source	JSON file (db.json)
Methods supported	GET, POST, PUT, PATCH, DELETE
Useful for	React app testing without real API backend

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.

You can use either the built-in fetch() function or the axios library to make HTTP requests.

1. Using fetch()

2. Using axios()

First, install axios:

npm install axios

Feature	fetch()	axios()
Built-in	Yes (native to browser)	No (external library)
Syntax	More verbose for JSON and errors	Cleaner and shorter
Response Type	Needs res.json() manually	Auto-parses JSON
Error Handling	Only catches network errors by default	Handles HTTP errors more easily
Extra Features	Basic	Supports interceptors, timeouts, etc.

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

Firebase is a Backend-as-a-Service (BaaS) platform developed by Google that helps developers build and manage web and mobile applications without managing server infrastructure.

It provides ready-to-use backend services like authentication, real-time database, hosting, storage, and more — so you can focus on building the frontend.

Key Features of Firebase

Feature	Description
Authentication	Provides easy user login using email/password, Google, Facebook, etc.
Firestore & Realtime Database	Cloud-hosted NoSQL databases for storing app data in real-time
Cloud Storage	Store and serve user-generated content like images, files, videos
Firebase Hosting	Fast and secure hosting for web apps, static content, and SPA deployments
Firebase Cloud Messaging (FCM)	Send push notifications across platforms
Firebase Analytics	Track user behavior and app usage for insights
Firebase Functions	Write backend logic (Node.js) that runs on serverless cloud functions
Firebase Testing & Crashlytics	Test apps and track errors/crashes in real time

Benefits of Using Firebase

- No need to manage servers
- Scales automatically
- Real-time syncing of data
- Easy integration with Android, iOS, and web
- Secure and reliable (backed by Google)

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

When working with APIs in React, handling loading and error states is crucial for providing a smooth and reliable user experience.

1. Loading State: Improves User Experience

- Why important:
  While waiting for data, the user should be informed that something is happening in the background.
- How it's used:
  Show spinners, "Loading..." text, or skeleton screens.

2. Error State: Handles Failures Gracefully

- Why important:
  API calls can fail due to network issues, wrong URLs, server errors, etc. Without handling this, your app may crash or behave unexpectedly.
- How it's used:
  Display user-friendly messages like: “Something went wrong. Please try again.”

State	Purpose	Result
Loading	Indicates data is being fetched	Spinner or “Loading...” message
Error	Informs the user of issues	User-friendly error message

13. Context API

Question 1: What is the Context API in React? How is it used to manage global state across multiple components?

The Context API is a built-in feature in React that allows you to share data (global state) across multiple components without passing props manually at every level (known as "prop drilling").

- Avoids prop drilling (passing props through intermediate components)
- Useful for global data like user info, theme, language, auth status, etc.
- Simple alternative to external state libraries like Redux for small to medium apps

Context API – Step-by-Step

1. Create a Context

```
import { createContext } from "react";

const MyContext = createContext();
```

2. Create a Provider Component

Wrap your app or component tree and pass shared data.

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

export const MyContext = createContext();

export function MyProvider({ children }) {

  const [user, setUser] = useState("John");

  return (

    <MyContext.Provider value={{ user, setUser }}>

      {children}

    </MyContext.Provider>  );}
```

3. Use Context in Child Components

Access the shared data using useContext.

```
import React, { useContext } from "react";

import { MyContext } from "./MyProvider";

function ChildComponent() {

  const { user } = useContext(MyContext);

  return <h1>Hello, {user}</h1>;}
```

4. Wrap Your App with Provider

```
import { MyProvider } from "./MyProvider";
```

```
function App() {
 return (
  <MyProvider>
   <ChildComponent />
  </MyProvider>
 );
}
```

Feature	Description
Context API	Shares global data without prop drilling
Provider	Supplies the data
useContext()	Hook to access context value inside components
Best For	Theme, Auth, Language, Cart, User Info, etc.

Question 2: Explain how createContext() and useContext() are used in React for sharing state.

React provides createContext() and useContext() to easily share state or values between components without passing props manually.

◆ 1. createContext()

- Used to create a Context object.
- It returns a Provider and Consumer.
- Usually exported so it can be used across the app

2. useContext()

- A React hook used to consume context data inside any functional component.
- Works only inside child components wrapped by the Provider.

Function	Purpose
createContext()	Creates a context object
Provider	Wraps components to provide shared data
useContext()	Reads data from the nearest Provider

14. State Management (Redux, Redux-Toolkit or Recoil)

Question 1: What is Redux, and why is it used in React applications? Explain the core concepts of actions, reducers, and the store.

Redux is a state management library for JavaScript applications (commonly used with React). It helps manage global application state in a predictable, centralized way.

Use Redux in React:-

- To manage complex state across multiple components
- Avoids prop drilling
- Ensures predictable state updates
- Makes debugging and testing easier
- Great for apps with shared or frequently updated data (e.g., shopping carts, authentication, UI themes)

Core Concepts in Redux

Concept	Description
Store	Holds the global state of the app in one place
Action	A plain JavaScript object that describes what happened
Reducer	A pure function that takes the current state and an action, and returns a new state

1. Store

- Created using createStore() (or configureStore() with Redux Toolkit)
- Central place where the whole app's state is stored

```
import { createStore } from 'redux';

const store = createStore(myReducer);
```

2. Action

- Describes the event or change
- Must have a type field

```
const incrementAction = { type: 'INCREMENT' };
```

3. Reducer

- Pure function: (state, action) => newState

```
const counterReducer = (state = 0, action) => {
 switch (action.type) {
  case 'INCREMENT': return state + 1;
  case 'DECREMENT': return state - 1;
  default: return state;
 }
};
```

Flow of Redux

1. Component dispatches an action
2. Action is sent to the reducer
3. Reducer processes it and returns a new state
4. Store updates the state
5. Subscribed components re-render with updated state

Summary Table

Term	Role
Store	Holds global state
Action	Describes what happened
Reducer	Defines how state changes
Dispatch	Sends an action to the reducer
useSelector / useDispatch	React hooks to connect with Redux

Question 2: How does Recoil simplify state management in React compared to Redux?

Recoil vs Redux — Key Differences

Feature	Recoil	Redux
Setup	Minimal setup, no boilerplate	Requires setup of store, reducers, actions
React Integration	Built specifically for React	External library integrated via React bindings
State Access	Uses useRecoilState (like useState)	Uses useSelector, useDispatch
Learning Curve	Easier to learn and use	Steeper learning curve
State Structure	Atom-based (modular, flexible)	Global single store (monolithic)
Performance	Fine-grained updates (component-level)	May re-render large parts of tree

Key Concepts in Recoil

Concept	Description
Atom	A piece of state (like useState but globally shareable)
Selector	Derived/computed state based on atoms
RecoilRoot	Provider that wraps your app (like Redux's <Provider>)

Example of Recoil Usage

1. Install Recoil

2. Setup RecoilRoot

3. Create an Atom

4. Use it in a Component