**Module 3**

**Topic 1: Linking and Routing**

**Lecture 1:** **Basic Types of navigation**

✅ **Evolution of Web Design Standards**

* Early web lacked design standards; developers experimented freely.
* Over time, best practices were established by the web development community.
* Similar to aviation history: experimentation led to standardized models

✅ **Focus of Modern Website Navigation**

* Modern navigation prioritizes **utility and user experience**.
* Stephen Krug's book *“Don’t Make Me Think”* emphasizes intuitive navigation.
* Avoid confusing users with non-standard or overly clever navigation patterns.

✅ **Accepted Types of Website Navigation**

* **Horizontal Navbar**: Top-aligned menus, common in most sites.
* **Vertical Sidebar**: Side menus, often used for dashboard layouts.
* **Burger Menu**: Collapsible menu icon (three lines), common in mobile views.
* **Mega Menu**: Drop-down menus for e-commerce or complex sites.
* **Footer Navigation**: Links grouped at the bottom, often in columns.

✅ **Combining Multiple Navigation Styles**

* Websites often use a mix of navigation types for better responsiveness.
* Example: Horizontal navbar becomes a burger menu on smaller screens.

✅ **How Navigation Works in React**

* Visually, React navigation appears similar to HTML/CSS.
* Technically, React apps load within a **single <div>**, controlled by the virtual DOM.
* Navigation doesn't reload pages but changes the displayed view inside the app.
* Mimics a multi-page experience using dynamic content rendering.

✅ **React Router for Navigation**

* React itself doesn't provide navigation by default.
* Use **React Router library** to manage navigation between views.
* Works by injecting new content into the main app container based on route changes.
* Example analogy: like pressing buttons in a stationary elevator that loads new floors inside the same space.

**Lecture 2:** **Navigation (Readings)**

**🔄 General Differences**

|  |  |  |
| --- | --- | --- |
| Feature | Multi-Page Applications (MPAs) | Single Page Applications (SPAs) |
| Page Load Behavior | Loads full new HTML page on each link click | Loads data or component dynamically without page reload |
| Server Dependency | Heavy (renders and sends new pages every time) | Light (server sends data like JSON, frontend updates view) |
| Speed/Performance | Slower, especially on poor networks | Faster, smoother user experience |
| Initial Load Time | Usually fast (just one page) | Can be slower if everything is bundled |
| Content Update | Full-page refresh | Dynamic, partial updates |

**⚙️ Technical Workflow**

|  |  |  |
| --- | --- | --- |
| Feature | MPA | SPA |
| Navigation | Uses regular <a> tags to load new pages | Uses Link components (e.g., from React Router) |
| Rendering | Handled mostly by the server | Handled by the client/browser |
| Data Handling | Page reload fetches new HTML | AJAX/Fetch/GraphQL fetches JSON data |
| DOM Manipulation | Reloads real DOM entirely | Updates via virtual DOM |

**📍 Practical Examples**

|  |  |  |
| --- | --- | --- |
| Example Action | MPA Behavior | SPA Behavior |
| Click "Profile" link | Browser requests full new HTML page | Browser loads component and fetches data via API |
| Show random movie on button | Button sends POST → returns new page with movie title | Button sends POST → gets JSON → updates only movie label |
| Navigation bar | Entire page refreshes on link click | Just swaps views/components; nav bar remains unchanged |

**🔗 Anchor Tags and Routing**

|  |  |  |
| --- | --- | --- |
| Feature | MPA | SPA |
| Anchor Tag Behavior | Default behavior (refreshes page) | Prevent default; simulate navigation with JS |
| URL Routing | Handled by server routing | Handled by client-side routing (e.g., React Router) |
| Templates | Each page has a separate HTML file | Views/components injected into one root page |

**Lecture 3: Conditional rendering**

**✅ React Dynamically Renders Components**

* React updates page content dynamically by rendering different components rather than loading new pages.
* This helps in creating seamless user experiences in single-page applications (SPAs).

**✅ Importance of Clear Rendering Logic**

* Developers must give React specific instructions on what to render and when.
* Event-driven rendering (e.g., on click) adds complexity to component logic.
* Writing clear, conditional logic simplifies this process.

**✅ Understanding Conditional Rendering**

* Conditional rendering allows components to display based on state values.
* Helps in rendering only the necessary content based on specific conditions or user interaction.

**✅ State vs Props**

* **State**: Internal to the component; can be changed by the component itself.
* **Props**: External data passed to components; cannot be modified internally.

**✅ Real-World Example: Sidebar Toggle**

* A button can control a sidebar's visibility using state.
* When clicked, a toggleSidebar state changes from false to true, rendering the sidebar component conditionally.

**✅ Conditional Logic in JavaScript**

* React supports standard JavaScript conditional logic.
* if statements and ternary operators are commonly used.

**✅ Ternary Operator for Rendering**

* A shorthand syntax for if-else:

condition ? expressionIfTrue : expressionIfFalse;

* Useful for inline rendering logic in JSX.

**✅ Example: Productivity App**

* Uses getDay() to check the current day.
* Displays:
  + **"Get it done!"** on weekdays (Mon–Fri).
  + **"Get some rest."** on weekends (Sat–Sun).

**✅ Components Used in the Example**

* CurrentMessage component uses the day logic.
* Conditionally renders:
  + Workdays component for weekdays.
  + Weekends component for weekends.

**✅ Logical AND in Conditions**

* day >= 1 && day <= 5 ? <Workdays /> : <Weekends />
* Ensures both conditions are met to render the Workdays component.

**✅ Simplified Boolean Example**

* Component IsItSummerYet:

const summer = true;

return summer ? "Let's go to the beach" : "Waiting for summer...";

* Demonstrates ternary usage with a boolean.

**Topic 2:** **Using Assets in React**

**Lecture 1:** **What Are Assets in React?**

**✅** **What Are Assets in React?**

* Assets include **images, stylesheets, fonts, media files**, and any other files needed at runtime.
* These are essential for a fully functional and user-friendly application.
* Missing assets can lead to **unexpected behavior**, like default fonts or missing images.

**✅ Where to Store Assets**

* **src/assets/ folder**: For assets that must be imported into components (e.g., images used in JSX).
* **public/ folder**: For static assets not directly imported into components (e.g., favicon.ico, logo512.png).
* **General Rule**:
  + If React **doesn’t need the asset to compile**, store it in public.
  + If React **needs to import the asset into a component**, store it in assets.

**✅ Example Use Case**

* A pet adoption app displays images of animals.
* Images are placed inside the assets/ folder to be used dynamically in components.

**✅ Importing Assets (Standard Method)**

* Use import to bring an asset into a component:

import cat from './assets/cat.jpg';

* In the JSX:

<img src={cat} alt="Adoptable cat" />

* Use **descriptive names** for imported assets (e.g., cat, dog, headerLogo).

**✅ Alternative: Using require()**

* Skip the import statement and use:

<img src={require('./assets/cat.jpg')} alt="Adoptable cat" />

* Useful when importing dynamically or conditionally.
* The require keyword loads the image at runtime.

**✅ Best Practices for Asset Management**

* Keep assets **organized** in dedicated folders (assets/images, assets/fonts, etc.).
* Name files clearly and consistently.
* Only place in public/ if the asset doesn’t interact directly with JSX imports.

**Lecture 2: Bundling assets**

**✅ What Is Bundling?**

* **Bundling** = combining all imported files into one (or more) output files.
* Tools like **webpack** automate bundling in React (used by create-react-app).

**✅ What Is Webpack?**

* A **module bundler** for JavaScript apps.
* Builds a **dependency graph** and outputs optimized bundles.
* Handles multiple file types: .js, .css, .scss, .svg, etc.

**✅ Why Use Webpack?**

* Automatically resolves **complex dependency graphs**.
* Makes apps **load faster** and code easier to manage.
* Supports:
  + **Transpiling** modern JS to older versions (e.g., ES7 to ES5).
  + **SCSS to CSS** conversion.
  + **Source maps** for debugging.
  + **Template-based outputs** (like HTML files).

**✅ Webpack Modes**

* **Development mode**:
  + Builds **source maps**.
  + Enables **fast rebuilds** during development.
* **Production mode**:
  + **Minifies** and **optimizes** code for speed.
  + Reduces file size for **faster online delivery**.

**✅ Asset Embedding Trade-offs**

* **Pro**: All required assets are loaded up front.
* **Con**: Large bundles can **slow down** load time.

**✅ Performance Optimization**

* **Code splitting**:
  + Breaks the bundle into **smaller parts**.
  + **Lazy loads** modules as needed.
* **Improves** load time and **user experience**.

**✅ Rendering Options**

* **Client-side rendering (CSR)**:
  + Browser loads index.html, then React renders in the DOM.
  + Default in create-react-app.
* **Server-side rendering (SSR)**:
  + Server renders HTML before sending it to browser.
  + **Faster initial load**, better SEO.
* **Isomorphic rendering**:
  + Combines SSR + CSR.
  + Also called **universal rendering**.

**Lecture 3: Using embedded assets**

**🖼️ 1. Using import Statement**

* Create a relative path using import at the top of the file.

import rooftops from './assets/images/CentralPark.jpeg';

* Use it in JSX:

<img src={rooftops} height="200" alt="Central Park view" />

* ✅ **Best for** local assets known at build time.

**🖼️ 2. Using require() Function**

* No need to import at the top.
* Use directly in JSX:

<img src={require('./assets/images/CentralPark.jpeg')} height="200" alt="Central Park view" />

* ✅ Useful when importing conditionally or dynamically.

**🌐 3. Using an Online Image URL**

* Store a URL in a variable:

const randomImageURL = 'https://example.com/photo.jpg';

* Use in JSX:

<img src={randomImageURL} height="200" alt="Random park view" />

* ✅ Best for external or dynamically fetched images.

**Lecture 4: Audio and video**

✅ **Introduction to Media in React Apps**

* Smartphones and high-speed Wi-Fi have made audio/video content creation and sharing common.
* React developers often need to integrate audio and video in their apps.

✅ **Using HTML5 <video> Tag for Local Videos**

* HTML5’s <video> element can be used to embed local video files.
* Simply declare a variable and use it in the src attribute in JSX.
* Works similar to loading images or other static assets.

✅ **Limitations with Social Media Videos**

* HTML5 video tag may not work for content hosted on platforms like YouTube or Vimeo.
* These platforms usually provide **embed codes** to integrate their media.

✅ **Embedding Third-Party Videos as React Components**

* Embed videos as **separate React components** for reusability.
* Allows easy switching of videos by passing **video ID as a prop**.
* Gives more control over which video is displayed dynamically.

✅ **Using NPM Packages for Video Integration**

* NPM ecosystem provides a variety of video packages.
* Search for “React video” on [npmjs.org](https://www.npmjs.org) for popular packages.
* Benefits:
  + Streamlined integration
  + Additional features (e.g., responsive players, custom controls)

✅ **How to Choose the Right NPM Package**

* Check the following before choosing a package:
  + ⭐ Frequency of updates
  + 👥 Number of contributors
  + 🔗 GitHub maintenance activity
  + 🔍 Perform additional research (e.g., package reviews)

✅ **Recommended Package: react-player**

* GitHub URL: [github.com/CookPete/react-player](https://github.com/CookPete/react-player)
* Features:
  + Over 6,000 GitHub stars
  + 115+ contributors
  + Actively maintained and trusted by many developers

✅ **Understanding GitHub Stars**

* A "star" on GitHub indicates developer appreciation and interest.
* High star count = strong community support and reliability.

✅ **Overview: Media Packages in React**

* Learn how to install and use the react-player npm package to handle media content in React applications.

✅ **Installation of react-player**

* Package URL: <https://www.npmjs.com/package/react-player>
* Install via terminal:

bash

CopyEdit

npm install react-player

✅ **Importing the Package**

* To import the **full package functionality**:

import ReactPlayer from "react-player";

* To **optimize bundle size** by importing only YouTube support:

import ReactPlayer from "react-player/youtube";

✅ **Basic Usage Example**

import React from "react";

import ReactPlayer from "react-player/youtube";

const App = () => {

return (

<div>

<MyVideo />

</div>

);

};

* Use ReactPlayer inside your custom components (e.g., <MyVideo />).
* Helps embed and control media players in your app efficiently.