Bachelor of Science in Computer Science (English Program)

School of Information Technology

King Mongkut's University of Technology Thonburi

CSC105 Web Application Development (2/2022)

Learning Guide

Topic: Frontend Development

Lecturer: Mr.Sitiporn Wimolpunyakul (P' Neng)

Mr.sorrawit kwanja (P' Boss Sora),

Mr.Nattapat Teeranuntachai (P' Boss Nat)

E-mail: jigneng1@gmail.com (P' neng)

sorrawit02546@gmail.com (P' Boss sora)

bossnattapat238@gmail.com (P' Boss Nat)

Outline:

1. What is React.
2. Rendering
3. JSX
4. Component
5. Export / Import
6. State
7. Hook
8. Props
9. Handling Events
10. React router.

Contents

Chapter 1

What is React?

React

React (also known as React.js or ReactJS) is a free and open-source front-end JavaScript library for building user interfaces based on components. It is maintained by Meta (formerly Facebook) and a community of individual developers and companies.

React can be used as a base in the development of single-page, mobile, or server-rendered applications with frameworks like Next.js. However, react is only concerned with the user interface and rendering components to the DOM, so creating React applications usually requires the use of additional libraries for routing, as well as certain client-side functionality.

Front-End Framework Trends for 2023

According to Stack Overflow's survey, react is the most popular front-end framework, followed by jQuery and Angular.

Advantages of React

- React is a beginner-friendly framework.
- React can be used to create high-performance applications.
- Development is easy because of the DOM.
- React is high in demand and has a vast active developer community.
- Version control can be done quickly without disturbing the original workflow.

Disadvantages of React

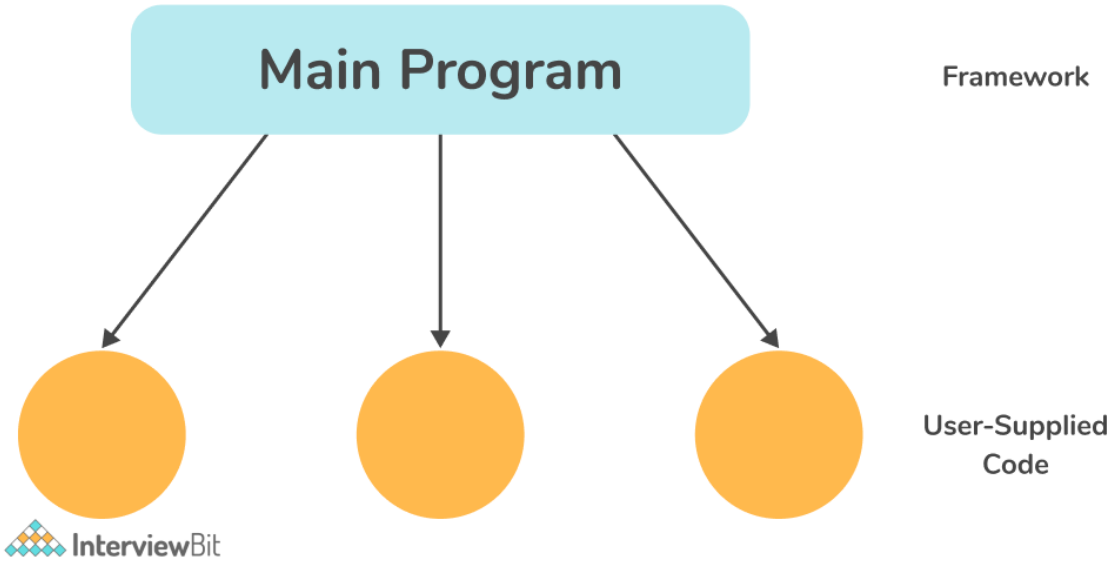
- JSX (JavaScript Extension) can be confusing for beginners.
- The documentation is not well maintained because of the rapid development of the framework.

What is Framework and Library?

Framework

Developers need not start from scratch when they have tools to assist them like Frameworks. A framework is like the foundation upon which developers build applications for specific platforms. It includes reusable pieces of code written to perform common tasks and uses code provided by a developer for custom functionality. The framework may include defined and undefined objects and functions that developers can use to create applications. In that way, we can add significant functionality to the system by using existing code around the structure.

Frameworks combine resources like image files and reference documents into one package. It is thus possible to modify that package to fit specific project requirements. With a framework, developers can integrate new features into an application and give it new capabilities.



Why do we use Frameworks?

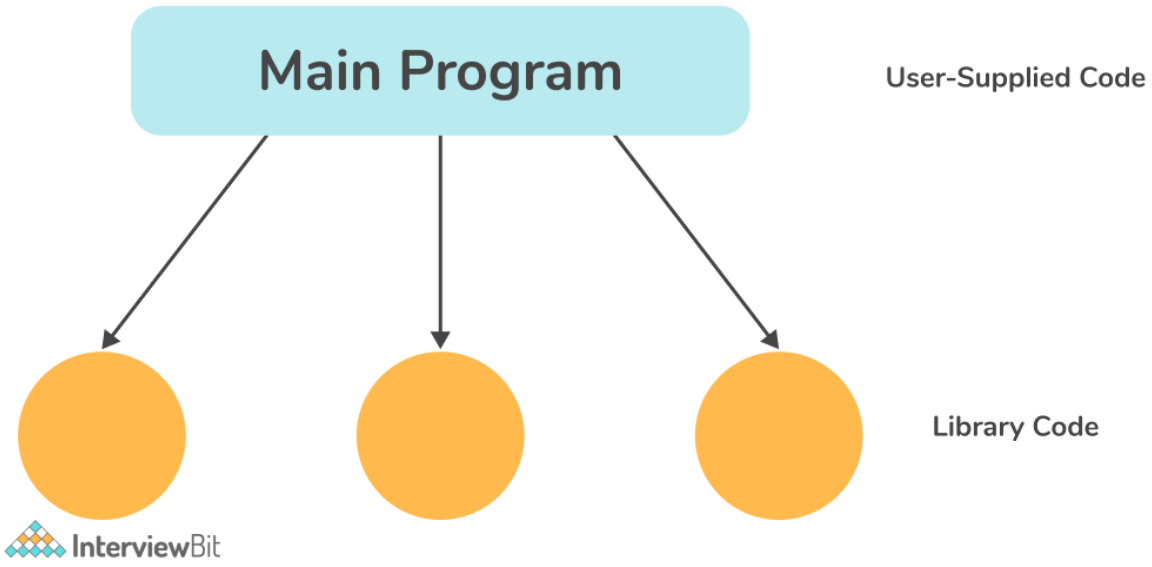
Software development is a complex process. This involves a lot of phrases like coding, designing, testing, etc. When it comes to coding, developers must worry about syntax, declarations, garbage collection, statements, exceptions, and more. Software frameworks make development easier by providing a common platform from which developers can control all or most of the software development process.

Advantages

- Software frameworks are highly versatile, robust, and efficient since they are often built, tested, and optimized by multiple software developers.
- Facilitates better programming practices and proper implementation of design patterns.
- Avoiding duplicates and redundant code results in fewer bugs and a more consistent development process
- It would be possible to create one's own software framework or contribute to an open-source framework. As a result, the functionality of software applications will continue to improve.
- Numerous code segments and functionalities have been built in framework and tested in advance, which makes applications more reliable.
- Even developers who do not own the code can test and debug it
- Developing an application takes significantly less time as it provides code to perform common tasks and uses code provided by a developer for custom functionality.
- As a result of using a software framework, you can focus on high-level functionalities, while the low-level functions are handled by the framework itself.

Library

A library is a collection of prewritten code that can be used to simplify tasks. The term library simply refers to a collection of code that is reused repeatedly. It is essentially a set of pre-defined functions and classes that programmers can use to simplify their work and speed up the development process. So, developers do not have to write code to accomplish specific functionality because the library already includes code for those functionalities. Standard libraries are available with most programming languages, but programmers can also make their own custom libraries.



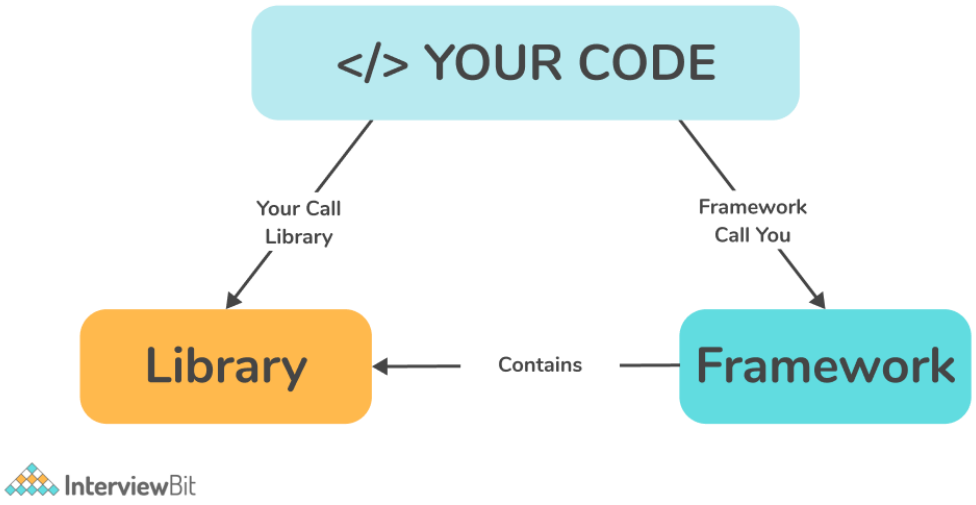
Why do we need a library?

There is only one answer to this question, and that is to reuse the code that has already been written by someone else (or another developer). Developers can avoid writing code for functionality that is already written in the library by using it. The result is more efficiency and less time spent on coding. Since it's more likely that other people are using it too, you'll benefit from them finding and fixing any bugs. This is one of the reasons libraries are useful.

What are different between React vs Library?

Put simply, the “Inversion of Control” (IoC) describes the difference between a library and a framework. In some ways, you can think of a framework as a collection of libraries, but it’s entirely different. By using a library, you control the flow of the program. The library can be invoked whenever and wherever you like. Contrary to this, when you use a framework, the flow is controlled by the framework. The framework instructs you where to put your code, but it will call your code as required. Simply put, our code calls the library’s code, but in a framework, it’s the framework’s code that calls our code as shown in the below diagram.

Developers can invoke libraries to perform specific tasks by using components, classes, and methods. A framework, however, already provides code to perform common tasks and uses code provided by a developer for custom functionality.



Chapter 2

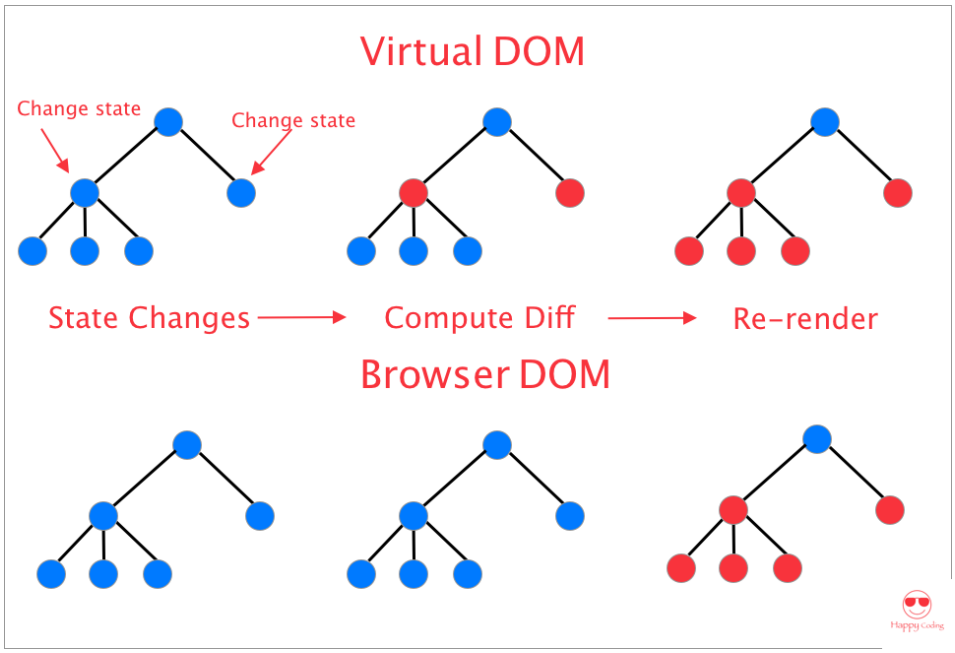
Rendering

Virtual DOM

Normally, whenever a user requests a webpage, the browser receives an HTML document for that page from the server. The browser then constructs a logical, tree-like structure from the HTML to show the user the requested page in the client.

This tree-like structure is called the Document Object Model, also known as the DOM. It is a structural representation of the web document as nodes and objects, in this case, an HTML document.

The DOM serves as an interface for the web document so that JavaScript and other scripting languages can access, manipulate, and programmatically interact with the document's content. For example, developers can use DOM APIs to add or remove elements, modify their appearance, and perform user actions on web elements.



How is the virtual DOM different from the real DOM?

A common misconception is that the virtual DOM is faster than or rivals the actual DOM, however, this is untrue. In fact, the virtual DOM's operations support and add on to those of the actual DOM. Essentially, the virtual DOM provides a mechanism that allows the actual DOM to compute minimal DOM operations when re-rendering the UI.

For example, when an element in the real DOM is changed, the DOM will re-render the element and all of its children. When it comes to building complex web applications with a lot of interactivity and state changes, this approach is slow and inefficient.

Instead, in the rendering process, React employs the concept of the virtual DOM, which conforms with its declarative approach. Therefore, we can specify what state we want the UI to be in, after which React makes it happen.

After the virtual DOM is updated, React compares it to a snapshot of the virtual DOM taken just before the update, determines what element was changed, and then updates only that element on the real DOM. This is one method the virtual DOM employs to optimize performance. We'll go into more detail later.

The virtual DOM abstracts manual DOM manipulations away from the developer, helping us to write more predictable and unruffled code so that we can focus on creating components.

Thanks to the virtual DOM, you don't have to worry about state transitions. Once you update the state, React ensures that the DOM matches that state. For instance, in our last example, React ensures that on every re-render, only Time gets updated in the actual DOM. Therefore, we won't lose the value of the input field while the UI update happens.

Purpose of Render

We can use the ReactDOM.render() in the application using the declaration of HTML code and the HTML element. The goal of this function is to represent the imposed HTML code within the specified HTML element tags. It helps to redirect the HTML page with the help of the render() function.

It can be immutable in a certain scenario when you create an element and then you can define it as constant, hence it would not allow changing this.

It helps to develop an application in a more presentable way using the below features:

- Open Source and Free
- Flexibility
- Decorators from ES7
- Server-side Intelligence
- Asynchronous Operations and Generators
- Flux Library Technique
- Restructuring Schemes
- Effectiveness of JSX

Chapter 3

JSX (JavaScript XML)

JSX

JSX stands for JavaScript XML, and it is a very useful tool for React developers. JSX is an extension of the JavaScript language which provides a way to structure component rendering using syntax like HTML. JSX gives us the ability to write HTML elements in JavaScript and place them in the DOM by converting the HTML tags into React elements without the need for other methods like createElement() or appendChild(). This combination of JavaScript and HTML leads to having more powerful applications with boosted performance.

```
const myElement = <h1>This is JSX!</h1>;

ReactDOM.render(myElement, document.getElementById('root'));
```

This JSX code snippet is an example of what it would take to output an <H1> of “This is JSX!” to your DOM using JSX syntax.

```
const myElement = React.createElement('h1', {}, 'This is not JSX!');

ReactDOM.render(myElement, document.getElementById('root'));
```

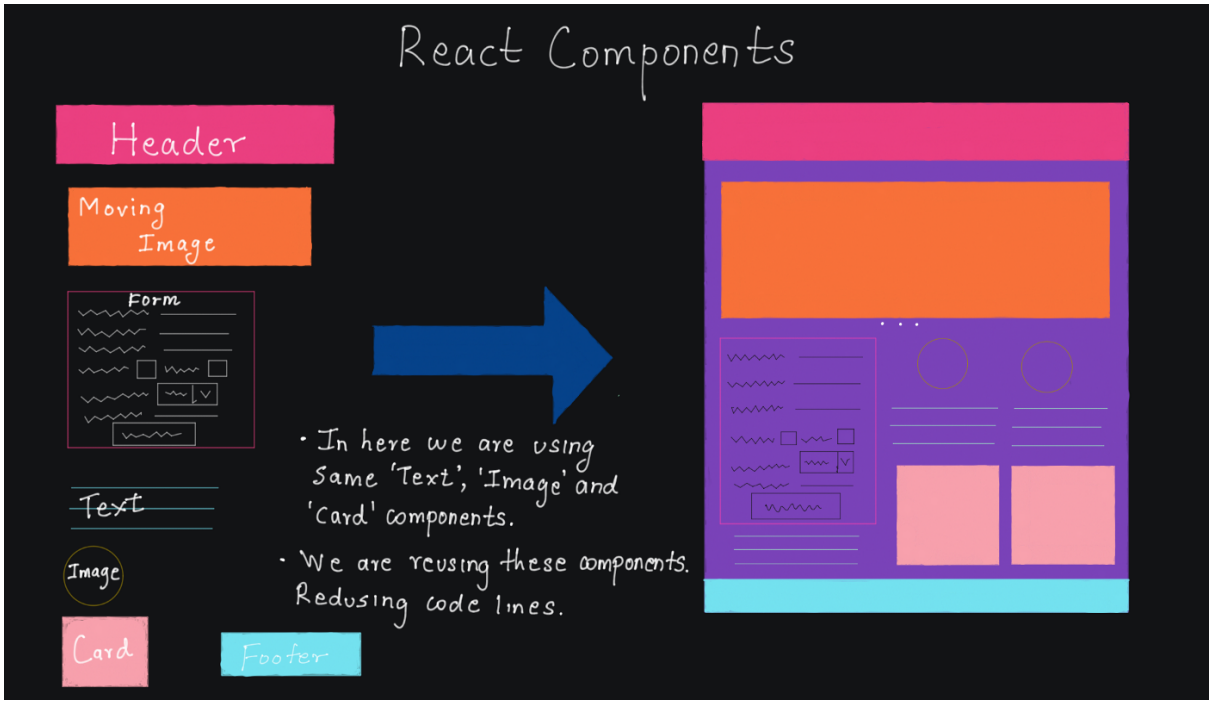
This second code snippet is an example of how to do the same exact thing as the first but without the use of JSX. As you can see, and most people would agree, the first example using JSX is much cleaner and clearer code that will be easier to read, edit and debug when the time comes.

Chapter 4

Components

Components

Components are one of the core concepts of React. They are the foundation upon which you build user interfaces (UI), which makes them the perfect place to start your React journey!

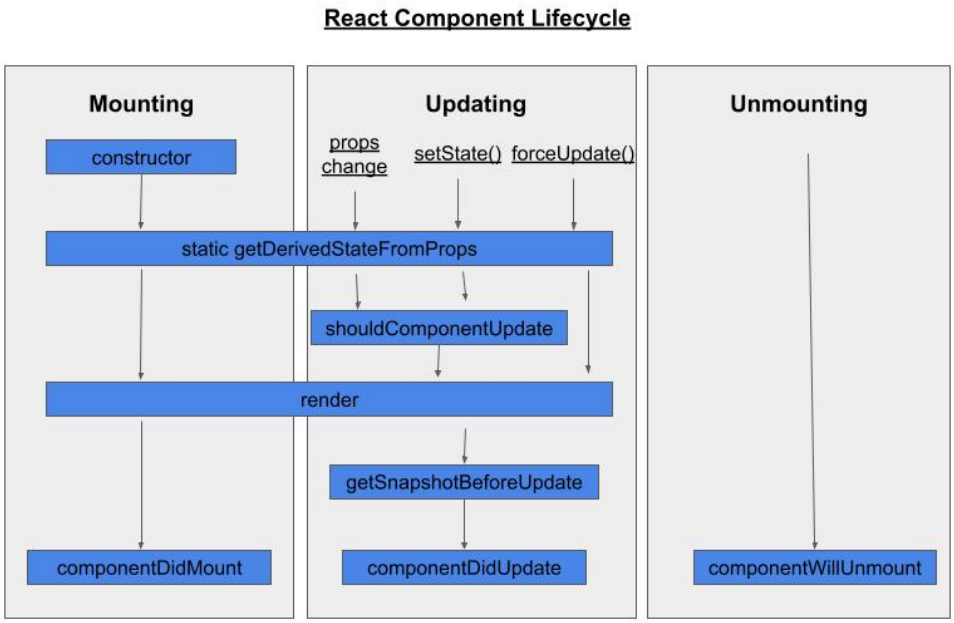


Components allow us to break down the user interface into smaller, self-organizing parts. and we can reuse it

There are two types of components in React

React Components Lifecycle

We have learned so far that the React web applications are actually a collection of independent components that run according to the interactions made with them. Each component in React has a life-cycle that you can monitor and change during its three main phases. There are three phases: Mounting, Updating, and Unmounting. You can use life-cycle methods to run a part of code at particular times in the process or application. You can refer to the below diagram to get a visual idea of the life-cycle methods



Chapter 5

Export / Import

Importing and Exporting Components

The magic of components lies in their reusability: you can create components that are composed of other components. But as you nest more and more components, it often makes sense to start splitting them into different files. This lets you keep your files easy to scan and reuse components in more places.

Exporting and importing a component.

What if you want to change the landing screen in the future and put a list of science books there? Or place all the profiles somewhere else? It makes sense to move Gallery and Profile out of the root component file. This will make them more modular and reusable in other files. You can move a component in three steps:

- Make a new JS file to put the components in.
- Export your function component from that file (using either default or named exports).
- Import it in the file where you'll use the component (using the corresponding technique for importing default or named exports).

Chapter 6

State

What Is 'State' in ReactJS?

The state is a built-in React object that is used to contain data or information about the component. A component's state can change over time; whenever it changes, the component re-renders. The change in state can happen as a response to user action or system-generated events and these changes determine the behavior of the component and how it will render.

- A state can be modified based on user action or network changes.
- Every time the state of an object changes, React re-renders the component to the browser.
- The state object is initialized in the constructor.
- The state object can store multiple properties.
- this.setState() is used to change the value of the state object.
- setState() function performs a shallow merge between the new and the previous state.

The setState() Method

State can be updated in response to event handlers, server responses, or prop changes. This is done using the setState() method. The setState() method enqueues all of the updates made to the component state and instructs React to re-render the component and its children with the updated state.

Always use the setState() method to change the state object, since it will ensure that the component knows it's been updated and calls the render() method

Chapter 7

Hook

React Hooks

Hooks are a new addition in React 16.8. They let you use state and other React features without writing a class.

Rules of Hooks

Only Call Hooks at the Top Level

Don’t call Hooks inside loops, conditions, or nested functions. Instead, always use Hooks at the top level of your React function before any early returns. By following this rule, you ensure that Hooks are called in the same order each time a component render. That’s what allows React to correctly preserve the state of Hooks between multiple useState and useEffect calls. (If you’re curious, we’ll explain this in depth below.)

Only Call Hooks from React Functions

Don’t call Hooks from regular JavaScript functions. Instead, you can:

✅ Call Hooks from React function components.

✅ Call Hooks from custom Hooks (we’ll learn about them on the next page).

By following this rule, you ensure that all stateful logic in a component is clearly visible from its source code.

Chapter 8

Props

Props

Props are arguments passed into React components.

Props are passed to components via HTML attributes.

Props stands for property.

React Props

React Props are like function arguments in JavaScript and attributes in HTML. To send props into a component, use the same syntax as HTML attributes:

State and Props in ReactJS

States are the type of built-in object in ReactJS.

We can create, handle, or manage our data within the component using the state object.Data can be passed by the props but the data cannot be passed by state itself. Using the state, the data is managed internally within the component.

We can combine both the state and props within our application in ReactJS. The steps for this combination process of the states and props within our ReactJS are as follows:

1. First, the state needs to be set within our parent component.
2. Then, the state can be passed as the props within the child component.

Chapter 9

Handle Events

Just like HTML DOM events, React can perform actions based on user events.

React has the same events as HTML: click, change, mouseover etc.

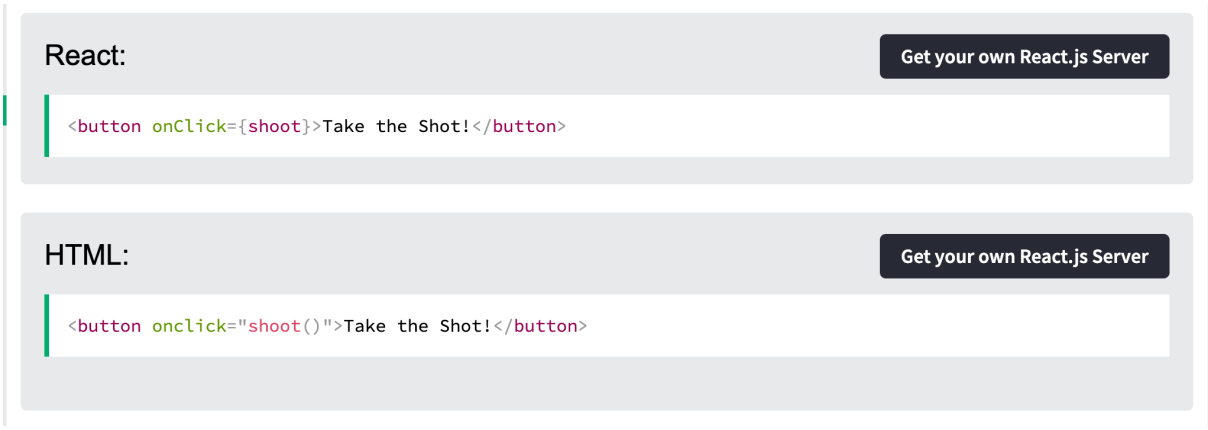
Adding Events

React events are written in camelCase syntax:

onClick instead of onclick.

React event handlers are written inside curly braces:

onClick={shoot} instead of onClick="shoot()".



Chapter 10

React Router

What Is a React Router?

The Router in React JS is a pure JavaScript package that allows you to use React to create complicated client-side apps. Initially launched in 2013, it has become one of the most prominent routing libraries in today's online applications.

React Router makes it simple to manage the URL and state of your application. You specify all of the potential URL patterns in your app and which UI component should be displayed for each one using React Router. This Router decreases the amount of code an app requires to maintain its state and makes adding new features more accessible.

Although there are significant differences, React may use a router on both the server and the browser.

Need for React Router

We will need to utilize Router in React JS to create a React application with navigation across multiple pages. React Router is a JavaScript framework that lets us handle client and server-side routing in React applications. It enables the creation of single-page web or mobile apps that allow navigating without refreshing the page. It also allows us to use browser history features while preserving the right application view.

A Router in React JS routes using a component-based architecture. It offers various routing components as required by the application. If you wish to learn more about its applications, check out this blog: Navigate React Router programmatically

What is Single Page Application?

Single Page Application (SPA) is a type of web application that loads a single HTML page and dynamically updates its content as the user interacts with the application, without requiring a full page refresh. In a SPA, most of the application logic is executed on the client-side using JavaScript, instead of on the server-side.

The main advantage of a SPA is that it provides a faster and more responsive user experience, as the user doesn't have to wait for a new page to load every time they interact with the application. Additionally, SPAs can provide a more seamless user experience by allowing for dynamic content updates, without disrupting the user's context or requiring them to reload the page.

SPAs are commonly built using JavaScript frameworks like React, Angular, or Vue.js, which provide tools for managing the application state, handling client-side routing, and building reusable UI components. While SPAs can be more complex to build and maintain than traditional server-rendered web applications, they offer a more modern and interactive user experience that is increasingly in demand in today's web development landscape.

What is react-router-dom?

react-router-dom is a routing library for React that provides a way to handle client-side routing within a single-page application. It allows you to define routes for your application and map them to different components, which can be rendered dynamically based on the current URL. This allows you to create a SPA that behaves more like a traditional desktop application, with seamless navigation and a more interactive user experience.

react-router-dom is built on top of the React framework and provides a declarative API for defining routes and handling navigation. It supports a variety of routing features, including nested routes, dynamic URLs with parameters, query strings, and programmatic navigation. It also includes tools for handling authentication and protecting routes based on user roles or permissions.

BrowserRouter and other Router types

After install react-router-dom with command:

```
npm install react-router-dom
```

You will be able to access components for setup routing with react-router-dom. The first thing to do in setting up routing is picking a Router. We would not focus on other types of router but we will provide you a quick look of what they are.

BrowserRouter

Most of the time you going to use BrowserRouter component to setup your routing in your index.js (or main.js). A <BrowserRouter> stores the current location in the browser's address bar using clean URLs and navigates using the browser's built-in history stack.

Clean URLs, also sometimes referred to as RESTful URLs, user-friendly URLs, pretty URLs or search engine-friendly URLs, are URLs intended to improve the usability and accessibility of a website or web service by being immediately and intuitively meaningful to non-expert users.

Original URL	Clean URL
`http://example.com/about.html`	`http://example.com/about`
`http://example.com/user.php?id=1`	`http://example.com/user/1`
`http://example.com/index.php?page=name`	`http://example.com/name`
`http://example.com/kb/index.php?cat=1&id=23`	`http://example.com/kb/1/23`
`http://en.wikipedia.org/w/index.php?title=Clean_URL`	`http://en.wikipedia.org/wiki/Clean_URL`

We would not go deep into what browser's built-in history stack is but if you interested, you should check out this MDN document https://developer.mozilla.org/en-US/docs/Web/API/History_API

HashRouter

<HashRouter> is for use in web browsers when the URL should not (or cannot) be sent to the server for some reason. This may happen in some shared hosting scenarios where you do not have full control over the server. In these situations, <HashRouter> makes it possible to store the current location in the hash portion of the current URL, so it is never sent to the server.

Ex. www.example.com/doc/#/welcome

MemoryRouter

A <MemoryRouter> stores its locations internally in an array. Unlike <BrowserHistory> and <HashHistory>, it isn't tied to an external source, like the history stack in a browser. This makes it ideal for scenarios where you need complete control over the history stack, like *testing*.

NativeRouter

<NativeRouter> is the recommended interface for running React Router in a React Native app.

StaticRouter

<StaticRouter> is used to render a React Router web app in node application (server-side rendering). Provide the current location via the location prop.

Create Router with createBrowserRouter() for new Data API in v6.4

The new version of react-router-dom recommend to use new way of setting up route by using createBrowserRouter and pass the returned object to <RouterProvider> as router prop. The router created by this method has access to the new Data API. You might have a look at the document for more information but we will stick to the old way by using component.

https://reactrouter.com/en/main/routers/picking-a-router

```
import {
  createBrowserRouter,
  createRoutesFromElements,
  Route,
  RouterProvider,
} from "react-router-dom";

const router = createBrowserRouter(
  createRoutesFromElements(
    <Route path="/" element={<Root />}>
      <Route path="dashboard" element={<Dashboard />} />
      {/* ... etc. */}
    </Route>
  )
);

ReactDOM.createRoot(document.getElementById("root")).render(
  <React.StrictMode>
    <RouterProvider router={router} />
  </React.StrictMode>
);
```

Normally we do this in the index.js or main.js

```
import { BrowserRouter } from "react-router-dom";

ReactDOM.createRoot(document.getElementById("root")).render(
  <React.StrictMode>
    <BrowserRouter>
        ... your app here
    </BrowserRouter>
  </React.StrictMode>
);
```

Routes and Route components

This two components are the essential part that make routing possible. Routes (with the ‘s’) contains collection of Route (without the ‘s’). The Route component used for defining path for the application and determine what component should be render on that specific URL.

Routes

Rendered anywhere in the app, <Routes> will match a set of child routes from the current location. Whenever the location changes, <Routes> looks through all its child routes to find the best match and renders that branch of the UI. <Route> elements may be nested to indicate nested UI, which also correspond to nested URL paths.

```
<Routes>
  <Route path="/" element={<Dashboard />}>
    <Route
      path="messages"
      element={<DashboardMessages />}
    />
    <Route path="tasks" element={<DashboardTasks />} />
  </Route>
  <Route path="about" element={<AboutPage />} />
</Routes>
```

https://reactrouter.com/en/main/components/routes

Route

In react-router-dom, the Route component is used to define a route within your application, which determines which components should be rendered based on the current URL.

When you define a Route component, you typically specify two key props: path and element. The path prop is used to specify the URL path that should match this route.

For example, if you set path="/about", this route will match any URL that starts with /about.

The element prop is used to specify the React component that should be rendered when this route is matched.

For example, if you set element={AboutPage}, the AboutPage component will be rendered when this route is matched.

```
import { Route } from 'react-router-dom';

function App() {
  return (
    <div>
      <Route path="/" exact element={HomePage} />
      <Route path="/about" element={AboutPage} />
    </div>
  );
}
```

"Splats" Also known as "catchall" and "star" segments. If a route path pattern ends with /* then it will match any characters following the /, including other / characters.

<Route path="/search/*" element={SearchResultsPage} />

In this example, we've defined a splat path for the SearchResultsPage component, which will match any path that starts with /search/ followed by any number of additional path segments. For example, the following paths would all match this route:

/search/term

/search/term1/term2

/search/term1/term2/term3

https://reactrouter.com/en/main/route/route

How to navigate with react-router-dom?

There are plenty of ways to navigate in your application with react-router-dom. Either with component like Link, NavLink or programmatically navigate with useNavigate hook.

Link Component

A <Link> is an element that lets the user navigate to another page by clicking or tapping on it. In react-router-dom, a <Link> renders an accessible <a> element with a real href that points to the resource it's linking to. This means that things like right-clicking a <Link> work as you'd expect. You can use <Link reloadDocument> to skip client side routing and let the browser handle the transition normally (as if it were an <a href>).

"Relative" By default, links are relative to the route hierarchy, so ".." will go up one Route level. Occasionally, you may find that you have matching URL patterns that do not make sense to be nested, and you'd prefer to use relative path routing. You can opt into this behavior with relative:

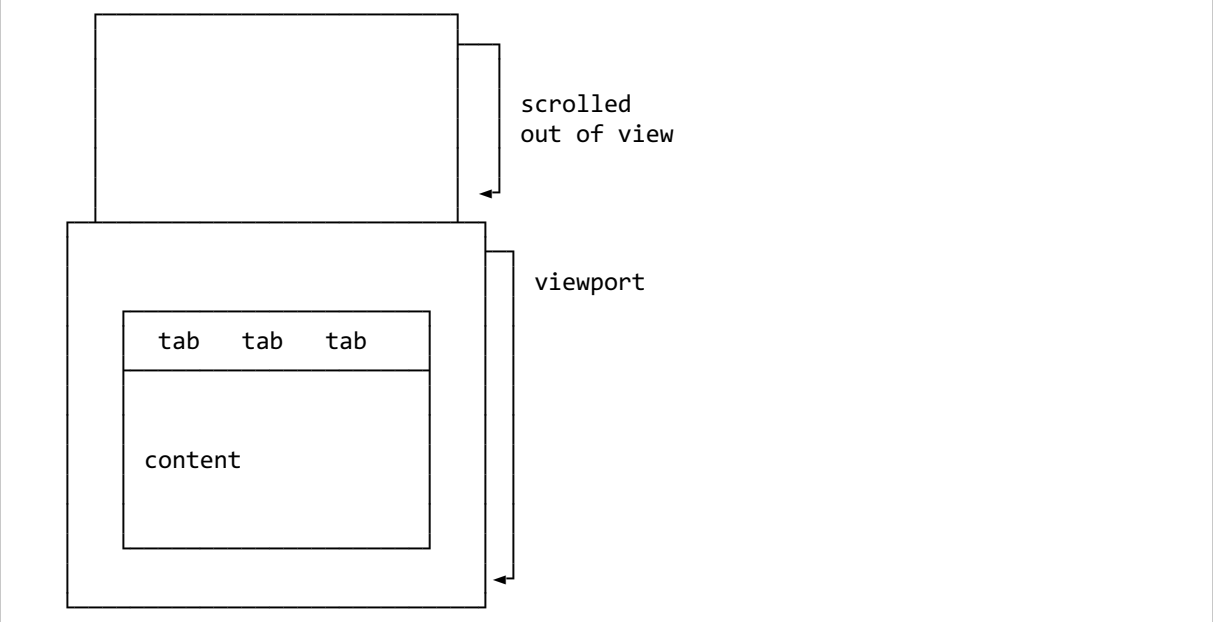
```
// Contact and EditContact do not share additional UI layout
<Route path="/" element={<Layout />}>
  <Route path="contacts/:id" element={<Contact />} />
  <Route
    path="contacts/:id/edit"
    element={<EditContact />}
  />
</Route>;

function EditContact() {
  // Since Contact is not a parent of EditContact we need to go up one level
  // in the path, instead of one level in the Route hierarchy
  return (
    <Link to=".." relative="path">
      Cancel
    </Link>
  );
}
```

“preventScrollReset” If you are using <ScrollRestoration>, this lets you prevent the scroll position from being reset to the top of the window when the link is clicked.

<Link to="?tab=one" preventScrollReset={true} />

This does not prevent the scroll position from being restored when the user comes back to the location with the back/forward buttons, it just prevents the reset when the user clicks the link.

An example when you might want this behavior is a list of tabs that manipulate the url search params that aren't at the top of the page. You wouldn't want the scroll position to jump up to the top because it might scroll the toggled content out of the viewport!



“replace” The replace property can be used if you'd like to replace the current entry in the history stack via history.replaceState instead of the default usage of history.pushState.

“state” The state property can be used to set a stateful value for the new location which is stored inside history state. This value can subsequently be accessed via useLocation().

The “replace”, “state”, “preventScrollReset” and relative link exist in all of navigating component eg. NavLink, Navigate

https://reactrouter.com/en/main/components/link#state

NavLink Component

A <NavLink> is a special kind of <Link> that knows whether or not it is "active". This is useful when building a navigation menu such as a breadcrumb or a set of tabs where you'd like to show which of them is currently selected. It also provides useful context for assistive technology like screen readers.

you can pass a function to either style or className that will allow you to customize the inline styling or the class string based on the component's active state. You can also pass a function as children to customize the content of the <NavLink> component based on their active state, specially useful to change styles on internal elements.

Passing function with isActive to style prop

```
<NavLink
  to="messages"
  style={({ isActive }) =>
    isActive ? activeStyle : undefined
  }
>
  Messages
</NavLink>
```

Passing function with isActive to className prop

```
<NavLink
  to="tasks"
  className={({ isActive }) =>
    isActive ? activeClassName : undefined
  }
>
  Tasks
</NavLink>
```

Passing function with isActive as children prop

```
<NavLink to="tasks">
  {({ isActive }) => (
    <span
      className={
        isActive ? activeClassName : undefined
      }
    >
      Tasks
    </span>
  )}
</NavLink>
```

https://reactrouter.com/en/main/components/nav-link

Navigate Component

A <Navigate> element changes the current location when it is rendered. It's a component wrapper around useNavigate, and accepts all the same arguments as props.

https://reactrouter.com/en/main/components/navigate

useNavigate Hook

The useNavigate hook returns a function that lets you navigate programmatically.

The navigate function has two signatures:

Either pass a To value (same type as <Link to>) with an optional second { replace, state } arg or

Pass the delta you want to go in the history stack. For example, navigate(-1) is equivalent to hitting the back button.

https://reactrouter.com/en/main/hooks/use-navigate

Dynamic Route

dynamic route is a route that includes parameters in the URL path, allowing you to pass data or information between components based on the current URL.

How to make dynamic route

To define a dynamic route in react-router-dom, you can use the path prop to specify a pattern that includes parameter placeholders. For example, if you want to define a route that includes a product ID parameter, you might define the route like this:

<Route path="/products/:id" element={ProductPage} />

In this example, we've defined a Route component with a path prop that includes a parameter placeholder (:id). This tells react-router-dom to match any URL that starts with /products/ followed by a parameter value, and to pass that parameter value to the ProductPage component as a prop.

Extract params from URL with useParams hook

To access the parameter value in the ProductPage component, you can use the useParams hook provided by react-router-dom. For example:

```
import { useParams } from 'react-router-dom';

function ProductPage() {
  const { id } = useParams();
  // ...use the id parameter value here...
}
```

In this example, we're using the useParams hook to extract the id parameter value from the URL and store it in a variable. We can then use this value to fetch product data from a server, render the appropriate content for the page, or perform any other logic based on the current URL.

Nested Route

In react-router-dom, a nested route is a route that is defined within the scope of another route, allowing you to create more complex, nested navigational structures within your application.

How to make nested route

To define a nested route in react-router-dom, you simply define a Route component inside the component that is rendered by another Route component. For example, if you have a UserPage component that should render different content based on the current URL, you might define nested routes like this:

```
<Route path="/user/:user">
  <Route index element={<User />} />
  <Route path="profile" element={<User />} />
  <Route path="blog" element={<UserBlog />} />
</Route>
```

Index route

an index route is a special type of route that is matched when the URL path matches the base path of a component, without any additional path segments.

Additional Resources

React

- React official document: https://reactjs.org/
- React new document (beta) : https://beta.reactjs.org/

Virtual Dom

- What is the virtual DOM in react : https://blog.logrocket.com/virtual-dom-react/

Framework and Library

- Framework and Library full Comparison : https://www.interviewbit.com/blog/framework-vs-library/

React Router DOM

- React Router DOM official document : https://reactrouter.com/en/main
- What is React Router DOM : https://www.geeksforgeeks.org/what-is-react-router-dom/