React 4 - Client Side Routing and React Router

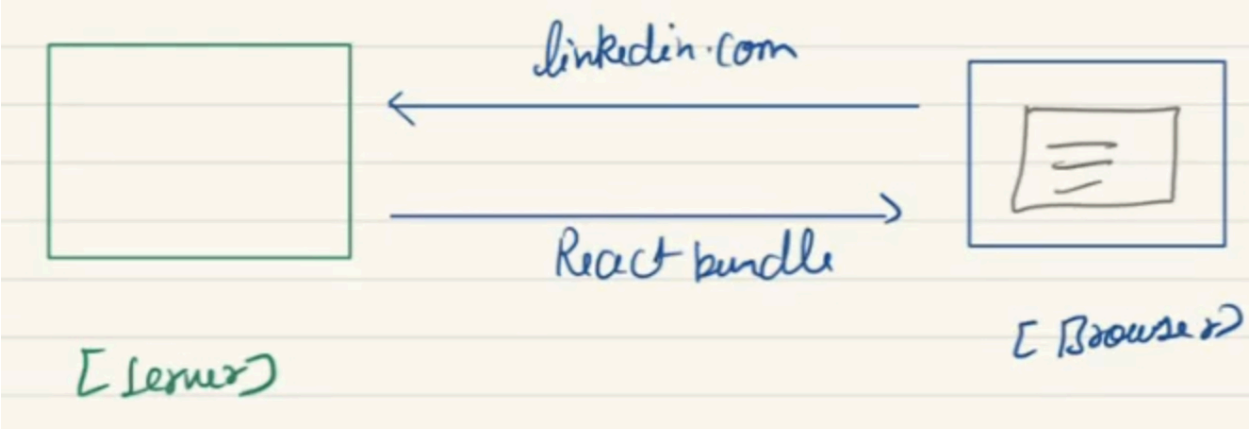
Agenda

title: Intution for Frontent Routing

Req res cycle for a react app

Let us discuss the client server once again

The browser sends the request for the page linkedin.com. The server returns a React Bundled file.



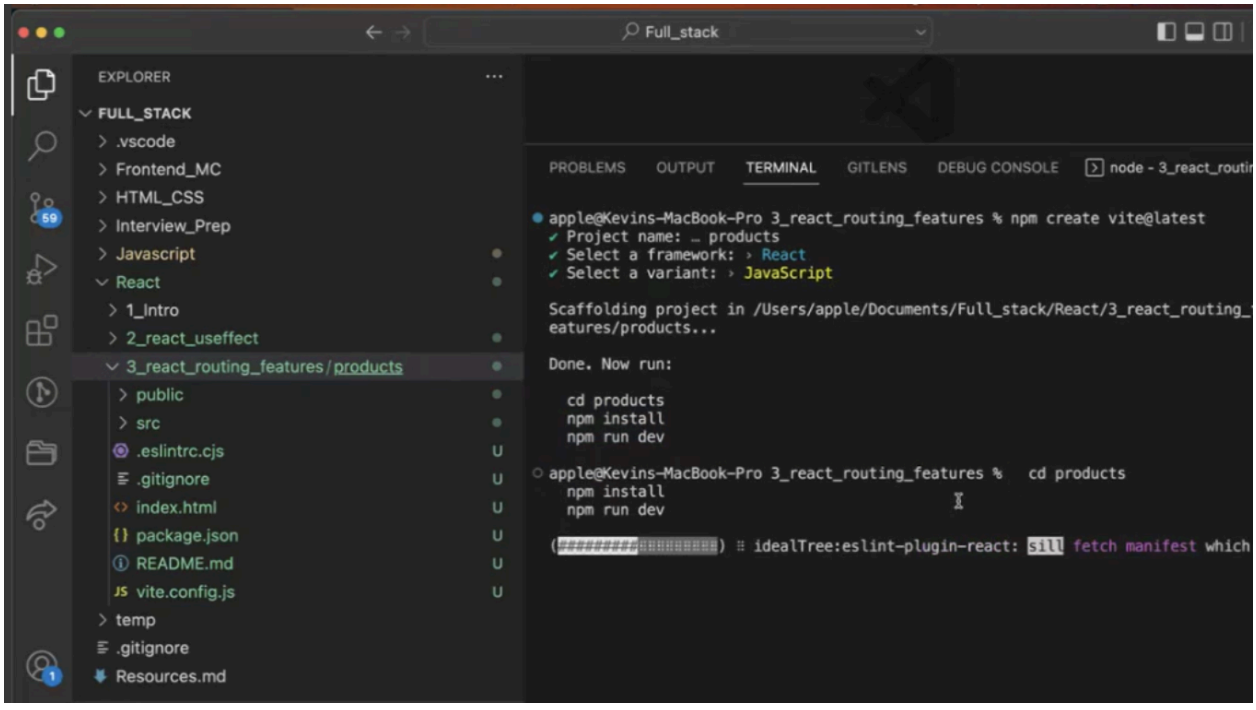
behaviour of SPA

1. For this kind of web apps
    a. The URL changes but the app doesn't reload
    b. You need to have different routes for different page
2. For the end user
    a. It should look like an app
    b. load time should be small
3. Let us say the browser is sending a request for the Job so it fetches the latest React bundle. But here exists a situation that we must remember
4. When we request data from the backend there are always two components UI and data.
5. Initial Request for UI and Bundle: When the user accesses the application, the browser sends a request for the initial UI and a minimal bundle of the application.
6. Initial UI and Placeholder/Loader: The application loads an initial UI, which may include placeholders or loaders for various components that are not immediately visible. These placeholders help create a better user experience by giving the impression that the application is responsive.
7. User Interaction: When the user interacts with the application, such as clicking a button or navigating to a specific page, the application checks whether the necessary UI components for that action are already loaded.
8. Conditional Data Request: If the UI components for the requested action are already loaded, the application only makes a request to the backend for the data needed for that action,

rather than fetching the entire UI and bundle again. Initial few pages are loaded to avoid unnessary loading again and again on the user side.

9. Initially we will fetch a few UIs of the pages and when the user clicks on any buttons, it will simply request data if the page UI already exists.
10. If you don't optimize it up to a level, the bundle size can be very big.

title: React Router DOM Setup

1. Create a new project and navigate inside it.
2. Npm create vite@latest
3. 

4. install module react-router-dom using npm i react-router-dom

5. import BrowserRouter from 'react-router-dom' in the main.jsx or the page to the lowermost component.
6. Wrap <App /> inside <BrowserRouter></BrowserRouter>

```
import React from "react";
import ReactDOM from "react-dom/client";
import App from "./App.jsx";
import "./index.css";
import { BrowserRouter } from "react-router-dom";

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

7. Create a folder named poc in src and create a file Routing.jsx inside it.
8. Inside Routing.jsx create a navbar with home, about, and listing.
9. Create Routes for different pages in the same file.
10. import {Routes, Route} from 'react-router-dom'

```
function About() {
 return <h2>About Page</h2>;
}
function Home() {
 return <h3>I am Home Page</h3>;
}
function Listing() {
 return <h3>I am Listing Page</h3>;
}
```

11. Use Route to specify paths. It takes two props path to define and whenever it has that path it renders the items mentioned inside the element prop.
12. Routes is used to combine multiple Route. And Inside Routes only Route can be called.

```
<Routes>
 <Route path="/" element={<Home></Home>}></Route>
 <Route path="/about/*" element={<About></About>}>
   {" "}
 </Route>
 <Route path="/listing" element={<Listing></Listing>}></Route>
 <Route path="*" element={<PageNotFound></PageNotFound>}>
   {" "}
 </Route>
 {/* path -> /* -. wild card  */}
</Routes>;
```

13. There is a wildcard matching if the path is given as path = "*" which matches everything. The order of placing wildcard won't affect its working. It will always try to match the specific path first.
14. We are adding Page Not Found using the wildcard.

```
function PageNotFound() {
   return <h3>Page Not found</h3>;
}
```

title: Link

1. There is another tag called Link in "react-router-dom". It takes a prop to="/Home" where we can give some path and it will change the URL and the Page accordingly.

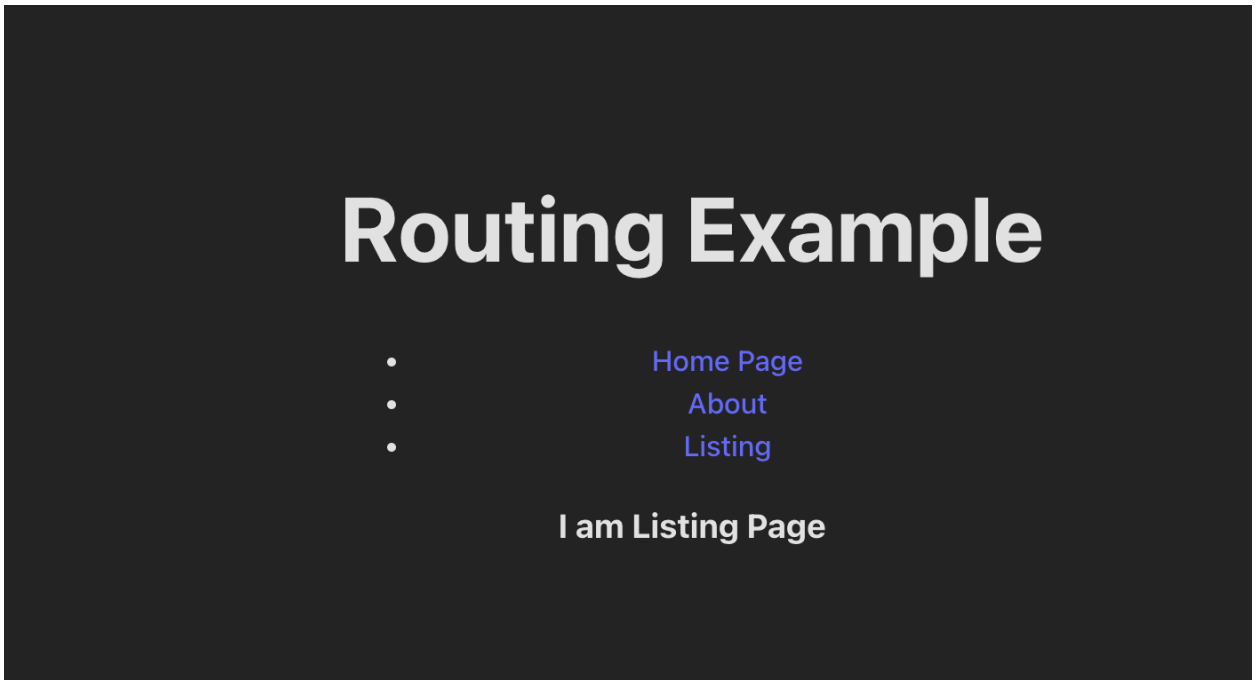
2. On clicking these buttons, the page won't reload. The content of the page changes but the page doesn't reload.

Complete code

```
import { Routes, Route, Link } from "react-router-dom";
function About() {
 return <h2>About Page</h2>;
}
function Home() {
 return <h3>I am Home Page</h3>;
}
function Listing() {
 return <h3>I am Listing Page</h3>;
}
function PageNotFound() {
 return <h3>Page Not found</h3>;
}
function Routing() {
 return (
   <>
       <h1>Routing Example</h1>
     <nav>
       <ul>
         <li>
           <Link to="/">Home Page </Link>
         </li>
         <li>
           <Link to="/about">About</Link>
         </li>
         <li>
           <Link to="/listing">Listing</Link>
         </li>
       </ul>
     </nav>

     <Routes>
       <Route path="/" element={<Home></Home>}></Route>
       <Route path="/about/*" element={<About></About>}>
```

```
          {" "}
        </Route>
        <Route path="/listing" element={<Listing></Listing>}></Route>
        <Route path="*" element={<PageNotFound></PageNotFound>}>
          {" "}
        </Route>
        {/* path -> /* -. wild card  */}
      </Routes>
    </>
  );
}
export default Routing;
```



Why Use Link Over <a> Tag

1. Better User experience

    a. Clicking on <a> tags causes browser to reload the entire page. This resets the application state and can disrupt user experience
    b. Link uses client side routing to update the URL without reloading the entire page. This keeps the application state intact and help transition from one view to another seamlessly
2. Performance
    a. Every page load requires browsers to fetch the resources (HTML, CSS, JS) again which is inefficient especially over slower network connections
    b. Link only helps update the parts that are needed to change, hence speeding up the navigation
3. Integration with React Router
    a. Anchor tags usage will require to manually update the url and browser history and prevent the page reload (using event.preventDefault)
    b. Automatically handles navigation using React router;s navigation management system. Additional benefits like dynamic route matching and nested routes are available. It integrates seamlessly with the React Router setup, ensuring that route changes are in sync with the application's state and the URL

title: Template Routes

Template Routes/ Dynamic Routes

Let's say we are rendering user routes
So based on the id of the user the path as well as the page is defined.

```
<Route path="/listing" element={<Listing></Listing>}></Route>
       <Route path = "/users/:id" element = {<Users isAdmin =
{true}></Users>}> </Route>
      <Route path="*" element={<PageNotFound></PageNotFound>}>
```

The hook called usePrams() provided by React Router DOM returns an object whatever template route you have given.
This will return the path given after ../users/.

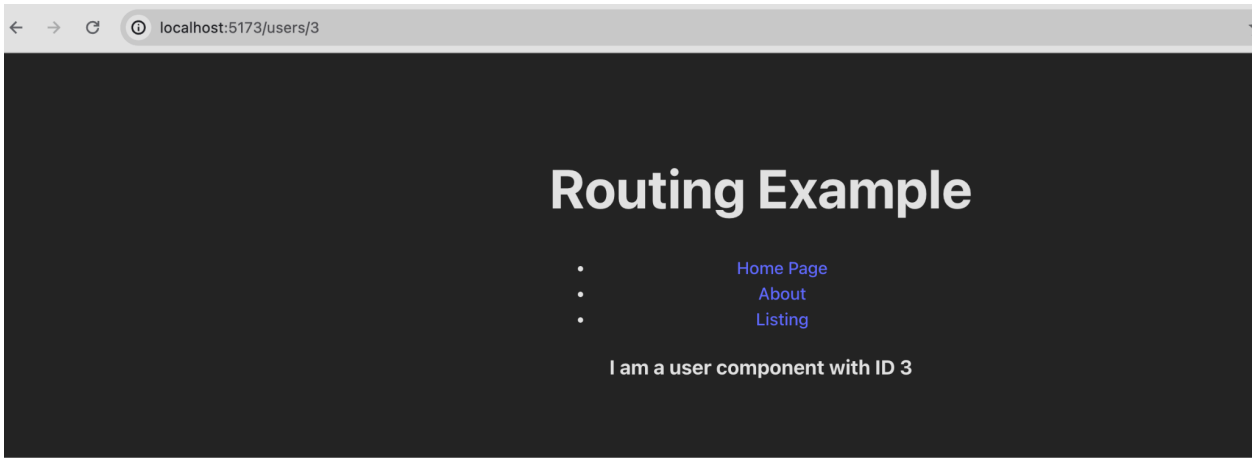
(We are using props to define if the user isAdmin)

```
import { Routes, Route, Link, useParams } from "react-router-dom";
```

```
function Users(props) {
 console.log(props.isAdmin);
 let params = useParams();
 const userID = params.id;
 console.log("param", params);

 return <h3>I am a user component with user ID {userID}</h3>;
}
```

Change the URL to users/3



Application of Template route

Fake Store API - we want to make a simple get request for the users. recall demo how users and id returns the data in Fake Store API - https://fakestoreapi.com/users/1

Before rendering we will just check if the user data is not null then we will print the user data else we will provide some placeholder like loading....

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

```
function Users(props) {
 // console.log(props.isAdmin);
 let params = useParams();
 let [user, setUser] = useState(null);
 console.log(params);
 // https://fakestoreapi.com/users/2
 useEffect(() => {
   async function fetchData() {
     const resp = await fetch(`https://fakestoreapi.com/users/${params.id}`);
```

```
    const userData = await resp.json();
    console.log(userData);
    setUser(userData);
  }
  fetchData();
}, []);
return (
  <>
    {user == null ? (
      <h3>...loading</h3>
    ) : (
      <>
        <h4>User Name: {user.username}</h4>
        <h3> Name: {user.name.firstname + " " + user.name.lastname}</h3>
        <h4> Phone: {user.phone}</h4>
      </>
    )}
  </>
);
}
```

These are called template routes or dynamic routes. Everything written like :abc can be derived with the help of params.abc, where let params = useParams().

title: Redirecting Routes

Let's say we want to redirect the user to another path when some specific path is given as input.
Navigate component inside React Router DOM helps us achieve this.

```
<Route path = "/abc" element = {<Navigate to = "/"></Navigate>}></Route>
      <Route path="*" element={<PageNotFound></PageNotFound>}>
```

```
        {" "}
      </Route>
```

All the component should have their own file. It's just for explanatory purposes that we have defined everything under the same file.

title: IMDB Project

Project Description

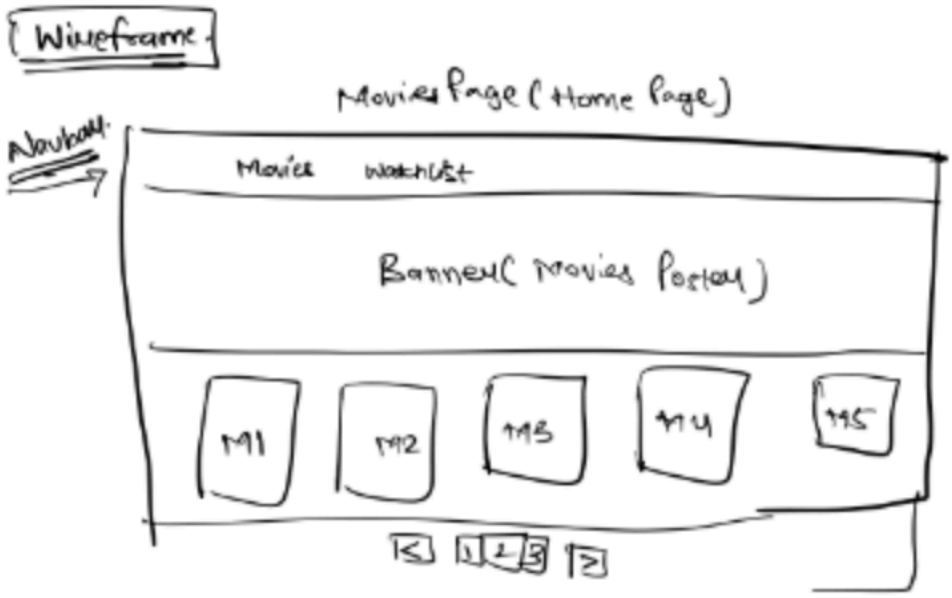
We will be creating an IMDB clone, where we will fetch Real-time trending movies data and will show it in grid format, we will be designing the whole application by using Tailwind CSS

Features of the project

The following will be the features of our IMDB clone:

1. The user will be able to view all the latest & trneding movies (IMDB API)
2. User can create his own separate watchlist
3. User can filter movies according to genre
4. User can sort movies according to ratings
5. Pagination will be implemented to move from one page to another to get updated data of other movies
6. Search feature will be there for movies.
7. We will be deploying this to netlify

Wireframe



What is tailwind css?

Tailwind CSS is a highly popular and innovative utility-first CSS framework for web development. Unlike traditional CSS frameworks that provide pre-designed components, Tailwind CSS focuses on providing a comprehensive set of utility classes that make it easier to create custom and responsive designs without the need for writing custom CSS.

Following are features of Tailwind CSS:

1. Utility-First Approach: Tailwind CSS adopts a utility-first approach, offering a vast collection of small, single-purpose utility classes that can be combined to create complex designs and layouts. This approach promotes code reusability and rapid development.
2. Customizable and Configurable: Tailwind is highly customizable through a configuration file, allowing developers to tailor the framework to match their project's specific design requirements. You can customize everything from colors and spacing to typography and breakpoints.
3. Responsive Design Made Easy: Creating responsive web designs is simplified with Tailwind CSS. It provides responsive variants of utility classes, enabling developers to adapt the layout and styling of their websites for various screen sizes and devices effortlessly.
4. Performance Optimization: Tailwind CSS is designed with performance in mind. It generates optimized CSS files by purging unused classes, resulting in smaller file sizes and faster loading times for web pages.
5. Active Community and Ecosystem: Tailwind CSS has a thriving community of developers who contribute to its growth and share resources. Additionally, there are numerous plugins and extensions available that enhance Tailwind's capabilities, making it suitable for a wide range of web development projects.
6. You can learn more about tailwind and how to set it up here with this link - https://tailwindcss.com/

Let's now Setup Tailwind in our React app

Step 1 – Create Your Project Folder

Open your terminal, and navigate to the folder where you want to build your project – for example Desktop. Input the command below in the terminal and click enter:

npm create vite@latest your-project-name -- --template react

"your-project-name" shoud be replaced with your project name. movies-app for example

The command above will create your project folder.

My project name is "movies-app", the movies-app folder will be created in the Programming folder on my Desktop

Step 2 – Navigate to Your Project Folder

Input the command below in your terminal and click enter:

cd movies-app

This command will navigate to your project folder. You should have this:

Inputing "cd movies-app" in terminal to navigate to the "movies-app" folder

Step 3 – Install Tailwind CSS and Other Dependencies

Input the command below in your terminal and click enter:

npm install -D tailwindcss postcss autoprefixer

Input this command to install the tailwindcss, postcss and autoprefixer dependencies

This command will install the following:

1. The Tailwind CSS framework
2. Post CSS - It's a powerful tool that allows for the transformation of CSS with JavaScript. This means you can write future CSS and it compiles into CSS understood by most browsers today.
3. Autoprefixer, which is a PostCSS plugin to parse CSS and add vendor prefixes to CSS rules.

Ensure the dependencies installed in package. Json

```
"devDependencies": {
  "@types/react": "^18.2.66",
  "@types/react-dom": "^18.2.22",
  "@vitejs/plugin-react": "^4.2.1",
  "autoprefixer": "^10.4.19",
  "eslint": "^8.57.0",
  "eslint-plugin-react": "^7.34.1",
  "eslint-plugin-react-hooks": "^4.6.0",
  "eslint-plugin-react-refresh": "^0.4.6",
  "postcss": "^8.4.38",
  "tailwindcss": "^3.4.3",
  "vite": "^5.2.0"
```

The version number might change when you read this.

Step 4 – Generate the Configuration Files

Input the command below in your terminal and click enter:

npx tailwindcss init -p

This command generates tailwind.config.js and postcss.config.js configuration files, also known as config files. They help you interact with your project and customize everything.

Step 5 – Configure Source Paths

Add the paths to all of your template files in your tailwind.config.cjs file. Template files include HTML templates, JavaScript components, and other source files that contain Tailwind class names. This is to make sure that vanilla CSS is generated for the corresponding elements.

Add this in your content section.
"./index.html",
"./src//*.{js,ts,jsx,tsx}",**

```
/** @type {import('tailwindcss').Config} */
export default {
 content: [
   "./index.html",
"./src/**/*.{js,ts,jsx,tsx}",

 ],
 theme: {
   extend: {},
 },
 plugins: [],
}
```

Step 6 – Add Tailwind Directives to Your CSS

Tailwind directives are custom Tailwind-specific statements that instruct CSS how to behave.

You will use these directives to insert different sets of Tailwind's styles into your CSS.

1. @tailwind base
    a. This directive adds Tailwind's basic foundational styles to your project. These are essential styles that provide a consistent styling baseline across all browsers, similar to what a CSS reset does.
2. @tailwind components
    a. This injects any predefined component styles from Tailwind. These can include styles defined by Tailwind and any additional styles from plugins you might be using.
3. @tailwind utilities
    a. This directive adds utility classes that Tailwind provides. Utility classes are the core of Tailwind's design system, allowing you to style elements directly in your HTML by applying utility classes that represent specific CSS properties.

Add the statements below to your ./src/index.css file:

```
@tailwind base;
@tailwind components;
@tailwind utilities;
```

Your index.css file contains some default styling. You can clear all that and paste the three lines of directives above.

Step 7 – Start/ Restart Your Vite Server - npm run dev

Step 8

Testing tailwind

```
function App() {
 return (
   <h1 className="text-3xl font-bold underline text-center">Hello world!</h1>
 );
}
```

Cheatsheet - https://nerdcave.com/tailwind-cheat-sheet

Homework

Build a navigation bar and have two LINKS on it, one by the name Home and one by the name Watchlist

Create two Components by the name Home and Watchlist

When you click on Home on the Navbar Home component should get rendered and when you click on Watchlist so Watchlist should get Rendered

Use React-Router-DOM to achieve this

Extra Notes - why tailwind is installed as a dev dependency

1. Build-time Usage: Tailwind CSS operates primarily at build time. When you use Tailwind, you aren't typically deploying the Tailwind library itself as you would with a library like jQuery or Lodash. Instead, you use Tailwind's utility classes in your markup, but the actual CSS that Tailwind generates based on your configuration and usage needs to be processed (compiled and purged) during the build step before it goes to production.
2. CSS Generation and Purging: Tailwind processes your CSS using your build configuration to generate the necessary styles. During this process, it also purges unused styles. This minimizes the CSS bundle size by including only the styles you actually use in your production CSS. This step is crucial given the vast number of utility classes Tailwind provides.
3. Development Tool: Given that all of Tailwind's processing occurs during development—when you're writing and building your project—it is categorized as a tool that aids development. The final output is the minimized, production-ready CSS file, and not the Tailwind framework or its build tools themselves.
4. Deployment Considerations: When your application is deployed, it doesn't need Tailwind itself; it only needs the final CSS produced by your build process. Therefore, Tailwind and associated tools like PostCSS and Autoprefixer are not required to be present in the production environment.