1. How do you create Nested Routes react-router-dom configuration?

* We use nested routing in our application so that a parent component has control over its child component at the route level.
* Nested routes enables you to have multiple components render on the same page with route parity.
* This is useful for app experiences where you want the user to be able to "drill down" into content.
* export default function App() {
* return (
* <Router>
* <Routes>
* <Route path='/' element={<Home />} />
* <Route path='about' element={<About />} />
* <Route path='posts' element={<Posts />}>
* <Route path='new' element={<NewPost />} />
* <Route path=':postId' element={<Post />}>
* <Route index element={<PostIndex />} />
* <Route path='comments' element={<Comments />} />
* </Route>
* </Route>
* </Routes>
* </Router>
* )
* }

1. What is the order of life cycle method calls in Class Based Components?

* Each component in React has a lifecycle which you can monitor and manipulate during its three main phases.
* The three phases are: **Mounting**, **Updating**, and **Unmounting**.
* Mounting –
  + Mounting means putting elements into the DOM.
  + React has four built-in methods that gets called, in this order, when mounting a component:
    - constructor()
    - getDerivedStateFromProps()
    - render()
    - componentDidMount()
  + The render() method is required and will always be called, the others are optional and will be called if you define them.
* Updating –
  + The next phase in the lifecycle is when a component is updated.
  + A component is updated whenever there is a change in the component's state or props.
  + React has five built-in methods that gets called, in this order, when a component is updated:
    - getDerivedStateFromProps()
    - shouldComponentUpdate()
    - render()
    - getSnapshotBeforeUpdate()
    - componentDidUpdate()
  + The render() method is required and will always be called, the others are optional and will be called if you define them.
* Unmounting –
  + The next phase in the lifecycle is when a component is removed from the DOM, or unmounting as React likes to call it.
  + React has only one built-in method that gets called when a component is unmounted:

componentWillUnmount()

* https://www.w3schools.com/react/react\_lifecycle.asp

1. Why do we use componentDidMount?

* componentDidMount() is invoked immediately after a component is mounted (inserted into the tree).
* Initialization that requires DOM nodes should go here. If you need to load data from a remote endpoint, this is a good place to instantiate the network request.
* This method is a good place to set up any subscriptions. If you do that, don’t forget to unsubscribe in componentWillUnmount().
* You **may call**setState()**immediately** in componentDidMount(). It will trigger an extra rendering, but it will happen before the browser updates the screen.

1. Why do we use componentWillUnmount? Show with example.

* This method is called when a component is being removed from the DOM.
* componentWillUnmount() is invoked immediately before a component is unmounted and destroyed.
* Perform any necessary cleanup in this method, such as invalidating timers, canceling network requests, or cleaning up any subscriptions that were created in componentDidMount().
* You **should not call**setState() in componentWillUnmount() because the component will never be re-rendered. Once a component instance is unmounted, it will never be mounted again.

import React, { Component } from 'react';

export default class SideMenu extends Component {

constructor(props) {

super(props);

this.state = {

};

this.openMenu = this.openMenu.bind(this);

this.closeMenu = this.closeMenu.bind(this);

}

componentDidMount() {

document.addEventListener("click", this.closeMenu);

}

componentWillUnmount() {

document.removeEventListener("click", this.closeMenu);

}

openMenu() {

}

closeMenu() {

}

render() {

return (

<div>

<a

href = "javascript:void(0)"

className = "closebtn"

onClick = {this.closeMenu}

>

×

</a>

<div>

Some other structure

</div>

</div>

);

}

}

1. Why do we use super(props) in constructor?

* Props are used to pass data from parent components to child components.
* These props can be updated only by the parent component. It is read-only for child components.
* We might require props inside the child component constructor with this keyword.
* **Super()** function calls the constructor of the parent class. Using super constructor with props arguments basically allows accessing **this.props** in a Constructor function.
* Super() function is to call the constructor of the parent class. It is used when we need to access a few variables in the parent class.

1. Why can’t we have the callback function of useEffect async?

* You cannot directly make the callback function supplied to the useEffect hook async because: async functions implicitly return a promise, and; useEffect expects its callback to either return nothing or a clean-up function.
* Because React’s useEffect hook expects a cleanup function returned from it which is called when the component unmounts.
* Using an async function here will cause a bug as the cleanup function will never get called.