upGrad

#RahoAmbitious

Full Stack Software Development

upGrad

Course: Advanced Frontend Development Using React

Lecture on: React Routing

Instructor: Mrigank Kaushik

In the last class, we discussed...

- Introduction to State
- Characteristics of State
- Setting the state using the *setState()* method
- Lifecycle of components and the processes of mounting, updating and unmounting

Poll 1

In which object should a component's properties be kept in React?

A. super
B. component
C. props
D. state

Poll 1 (Answer)

In which object should a component's properties be kept in React?

A. super

B. component

C. props

D. state

Poll 2

What will happen if you call the *setState()* method inside the *render()* method?

A. Output will be repeated on the screen

B. Stack overflow error

C. Code will work fine

D. Duplicate key error

Poll 2 (Answer)

What will happen if you call the *setState()* method inside the *render()* method?

A. Output will be repeated on the screen

B. Stack overflow error

C. Code will work fine

D. Duplicate key error

Poll 3

Which of the following is the most appropriate with respect to React lifecycle stages?

A. *render()* method is called in the mounting lifecycle stage

B. *render()* method is called in the updating lifecycle stage

C. *render()* method is called in both mounting and updating lifecycle stages

D. None of the above

Poll 3 (Answer)

Which of the following is the most appropriate with respect to React lifecycle stages?

A. *render()* method is called in the mounting lifecycle stage

B. *render()* method is called in the updating lifecycle stage

C. *render()* method is called in both mounting and updating lifecycle stages

D. None of the above

Today's Agenda

1. Single-Page and Multi-Page Applications and their advantages over each other
2. Types of components: Smart and Dumb
3. Difference between smart and dumb components
4. Routing in React
5. Implementing routing in the application using a node package called 'react-router-dom'
6. Developing a functionality for deleting a subscriber

upGrad

Single-Page and Multi-Page Applications

SPA vs MPA

upGrad

- Finally, it is time to study about Single-Page and Multi-Page Applications.
- Each architecture has its pros and cons and is well suited to a particular type of project and specific business goals. It is recommended that you choose the architecture according to your needs.

SPA vs MPA

upGrad

Multi-Page Application

- An MPA, which stands for Multi-Page Application, consists of several pages with embedded navigation to other pages.
- When you move from one page to another, the browser reloads the contents of the new page completely and downloads all the resources again even if the components are being repeated through all the pages.
- One such example can be the header of an application. In most applications that you see, the header remains constant across all the webpages.
    - If you build a multi-page application, this header would be re-rendered in all the pages, which, ideally, is not required.

SPA vs MPA

upGrad

Single-Page Application

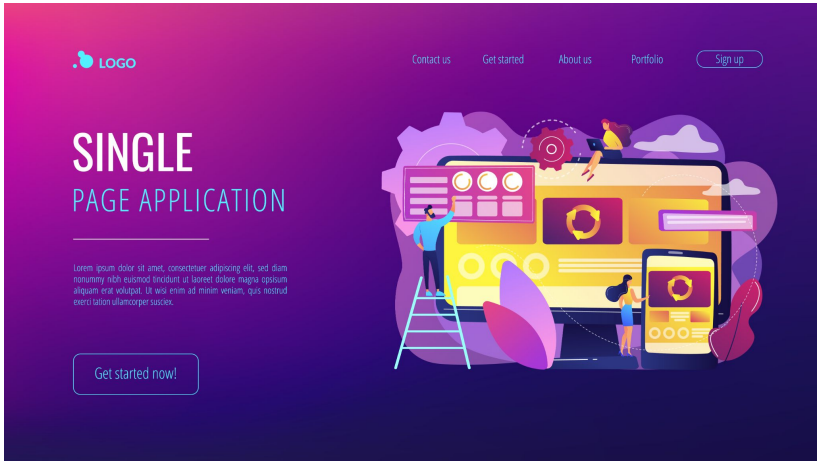
- An SPA or a single-page application works on the browser and does not require a page to be reloaded while being used. Some notable examples of an SPA are Gmail, Google Maps, Facebook, Github, etc.
- SPA is lighter and faster than an MPA because most of the resources are being loaded at once throughout the lifespan of the application. Only the data is transmitted back and forth.
- Single-page web applications fit perfectly for building dynamic platforms with small data volumes.
- An SPA is excellent for social networks and closed communities that require good application performance, dynamic nature and user experience.
- Coming back to React, it is a popular library for building SPAs and, in this course, you will be building one such application.

SPA vs MPA

upGrad

Advantages of SPAs over MPAs

- Faster loading of page; no need to download resources all over again
- Effective caching; easy local data storage
- Easy debugging; technologies provide their own debugging tools



SPA vs MPA

upGrad

Advantages of SPAs over MPAs

- Decoupling of front end and back end
- Simplified mobile development - Same back end can be used for web applications as well as native mobile applications
- Rich in responsiveness; better user experience
- JavaScript is mandatory in SPA.

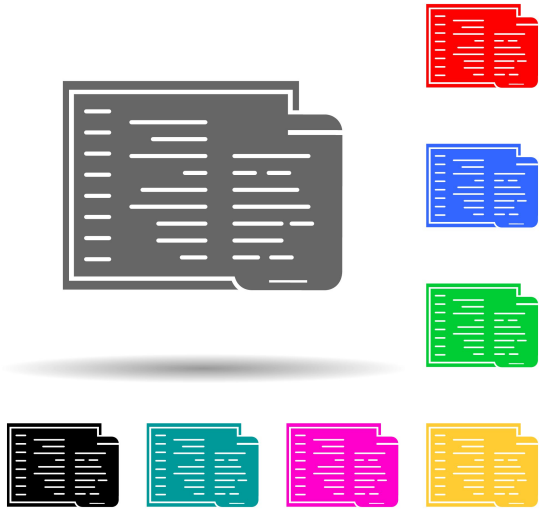
(**Note:** Theoretically, it is possible to have an SPA without having JavaScript enabled. If the app uses server-side or isomorphic rendering, the initial render on the server can be cached. Although this approach will come with some bottlenecks, you would have an SPA without JavaScript.)

SPA vs MPA

upGrad

Advantages of MPAs over SPAs

- Better Search Engine Optimization (SEO); architecture native to search engine crawlers; flexibility to add meta tags to each page
- Better in terms of analytics
- Unlimited scalability; new features can be added easily
- JavaScript not mandatory
- Better security; access control at a functional level



Poll 4

Which of the following is an advantage of MPA over SPA?

(Note: More than one option may be correct.)

A. Simplified mobile development

B. New features can be added easily

C. Faster loading of page

D. Better security

Poll 4 (Answer)

Which of the following is an advantage of MPA over SPA?

(Note: More than one option may be correct.)

A. Simplified mobile development

B. New features can be added easily

C. Faster loading of page

D. Better security

Smart vs Dumb Components

Smart vs Dumb Components

upGrad

- Components can be classified into two categories based on their roles in the application: **smart** and **dumb**. This is done to avoid any mismanagement in the state of the application
- A dumb (also known as presentational) component is a UI-based component that only presents data on the DOM. On the other hand, a smart (also known as container) component provides data and logic to dumb components
- Thus, dumb components describe the appearance of things, whereas smart components describe the working of things

Smart vs Dumb Components

upGrad

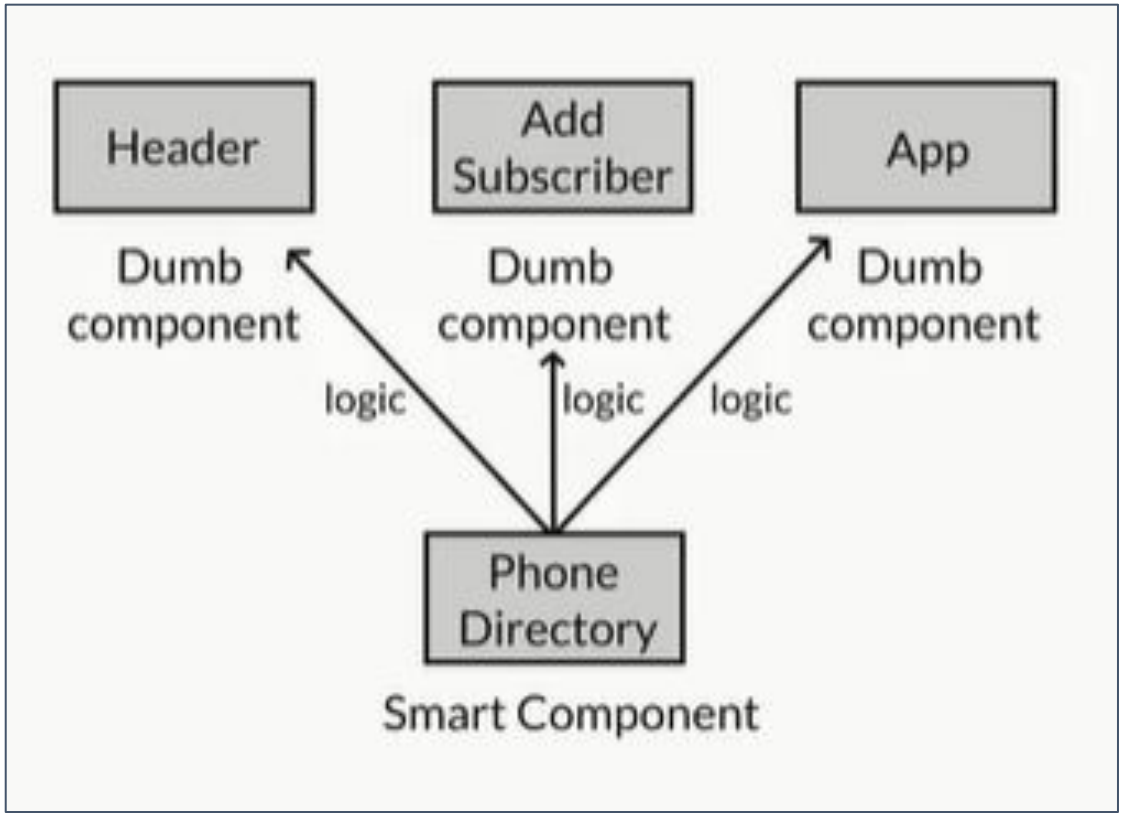
Phone Directory Application

- In context of our Phone Directory application. We have already made header as a dumb component
- The *AddSubscriber* component that we have made is also a dumb component. It is only managing its own state right now
- The *App* component, which does the work of showing all the subscribers, is also a dumb component
- Let's add one smart component as *PhoneDirectory* and pass all the logic from this smart component to our dumb components

Smart vs Dumb Components

upGrad

Phone Directory Application



Smart vs Dumb Components

upGrad

Phone Directory Application

Let's refactor our code a little bit and convert the *App* component into the *ShowSubscribers* component

- Rename *App.js* file to *ShowSubscribers.js* file. Also, rename *App.css* file to *ShowSubscribers.css* file
- Go to *ShowSubscribers.js* file and change the import statement of *App.css* to *ShowSubscribers.css*

```
import './ShowSubscribers.css';
```

- In the *App.js* file, make sure that you change the class name and export default statement

Code Reference

Smart vs Dumb Components

upGrad

Phone Directory Application

- Go to index.js file and change the import statement of *App.js* file to *ShowSubscribers*

```
import ShowSubscribers from './ShowSubscribers';
```

- Also, change the component name from *App* to *ShowSubscribers* inside *ReactDOM.render()* method

```
ReactDOM.render(<ShowSubscribers/>, document.getElementById('root'));
```

Code Reference

Smart vs Dumb Components

upGrad

Phone Directory Application

Let us create *PhoneDirectory* as a smart component inside the Phone Directory application

- Add *PhoneDirectory.js* file inside *src* folder. Make this as a class component which will be our smart component and which will provide logic to all other dumb components in our application

Smart vs Dumb Components

upGrad

Phone Directory Application

- Render *AddSubscriber* component inside this which is one of our dumb components

```
import React, { Component } from 'react';

class PhoneDirectory extends Component {
   render() {
       return (
           <AddSubscriber />
       )
   }
}

export default PhoneDirectory;
```

Code Reference

Smart vs Dumb Components

upGrad

Phone Directory Application

- Complete *PhoneDirectory.js* file by adding a constructor and adding state which maintains the list of subscribers

```
constructor() {
    super();

    this.state = {
       subscribersList: []
    };
}
```

Code Reference

Smart vs Dumb Components

upGrad

Phone Directory Application

- Now, add the *addSubscriberHandler* inside the *PhoneDirectory* component. This handler is a function which will add a subscriber into the list
- *console.log()* method will help one to know about the list of subscribers that are there in state. Every time a new subscriber is added to the list, you will come to know about it

Smart vs Dumb Components

upGrad

Phone Directory Application

```
addSubscriberHandler = (newSubscriber) => {
    let subscribersList = this.state.subscribersList;
    if (subscribersList.length > 0) {
        newSubscriber.id = subscribersList[subscribersList.length - 1].id + 1;
    } else {
      newSubscriber.id = 1;
    }
    subscribersList.push(newSubscriber);
    this.setState({ subscribersList: subscribersList });
    console.log(this.state.subscribersList);
}
```

Code Reference

Smart vs Dumb Components

upGrad

Phone Directory Application

- Now, pass this method as a prop inside the *AddSubscriber* component. Notice that here we have the logic of manipulating state in the smart component which is *PhoneDirectory* component here and from this component, we pass the method to the child component which can only call this method

```
render() {
    return (
        <AddSubscriber addSubscriberHandler={this.addSubscriberHandler.bind(this)} />
    )
}
```

Code Reference

Smart vs Dumb Components

upGrad

Phone Directory Application

- Now, go to the *AddSubscriber.js* file and add a form *onSubmit* event handler to the form. This will be called whenever the submit button, which is *Add* button over here, is called in order to submit the form

```
<form className="subscriber-form" onSubmit={this.onFormSubmitted.bind(this)}>
```

- Define this *onFormSubmitted* function which will call the *addSubscriberHandler* defined in the *PhoneDirectory* component and passed to this component as props

```
onFormSubmitted = (e) => {
    e.preventDefault();
    this.props.addSubscriberHandler(this.state);
}
```

Code Reference

Smart vs Dumb Components

upGrad

Phone Directory Application

- Go to *index.js* file. Import *PhoneDirectory* component and inside the return statement, render the *PhoneDirectory* component

```
import PhoneDirectory from './PhoneDirectory';

ReactDOM.render(<PhoneDirectory />,document.getElementById('root'));
```

- See the results in the browser. Open console tab in DevTools and see how adding a subscriber adds the details to the subscriber list which is inside the *PhoneDirectory* component
- Also, notice how the dumb component *AddSubscriber* only triggers a call to the method written inside the smart component *PhoneDirectory*

Note: The state manipulation is being done inside the smart component *PhoneDirectory*.

Code Reference

Smart vs Dumb Components

upGrad

Phone Directory Application

- You have successfully created *PhoneDirectory* as a smart component consisting of the list of subscribers' details
- You have also defined the logic of adding a new subscriber inside the *addSubscriberHandler()* method. If the length of *subscriberList* is greater than 0, then an *id* is allotted to the subscriber that is incrementing the list by 1; otherwise, an *id* of 1 is allotted

Smart vs Dumb Components

upGrad

Phone Directory Application

- Therefore, *addsubscriber* is not executing the application-level logic. The *PhoneDirectory* smart component is executing the logic. Hence, *addsubscriber* is the dumb component
- Similarly, *showsubscribers* can also be made a dumb component
- The *subscribersList* will be passed as a property to *showSubscribers*, and in *showSubscribers*, one can loop through the *subscribersList* using a map and render the list

Smart vs Dumb Components

upGrad

Phone Directory Application

Let us now make *ShowSubscribers* as a dumb component. We will pass the state containing list of all the subscribers and use this in order to show all the subscribers on the page.

- Go to *PhoneDirectory.js* file and add an import statement for *ShowSubscribers* component

```
import ShowSubscribers from './ShowSubscribers';
```

Code Reference

Smart vs Dumb Components

upGrad

Phone Directory Application

- Comment the code rendering AddSubscriber component

```
// <AddSubscriber addSubscriberHandler={this.addSubscriberHandler.bind(this)} />
```

- Render ShowSubscribers component inside the render method

```
<ShowSubscribers />
```

Code Reference

Smart vs Dumb Components

upGrad

Phone Directory Application

- Add a constant variable inside render method that will contain the subscribers list from the state

```
render() {
    const subscribersList = this.state.subscribersList;
}
```

- Pass this constant inside the *ShowSubscribers* component as props

```
<ShowSubscribers subscribersList={subscribersList} />
```

Code Reference

Smart vs Dumb Components

upGrad

Phone Directory Application

- Go to *ShowSubscribers.js* file. Remove all the commented code and the *constructor*
- Set the internal state of the page to the list of subscribers that was passed down as props
- Define the list of subscribers in the *PhoneDirectory.js* file

Code Reference

Poll 5

Which of the following are the only UI-based components?

A. Dumb components
B. Class components
C. Functional components
D. Smart components

Poll 5 (Answer)

Which of the following are the only UI-based components?

A. Dumb components

B. Class components

C. Functional components

D. Smart components

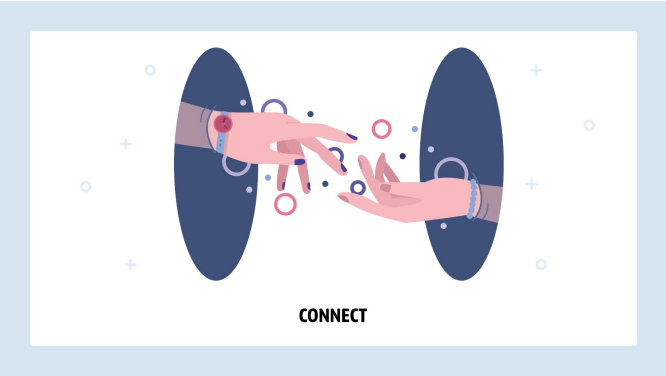
Routing

Routing

upGrad

Have you ever wondered how you navigate from one page to another in a single-page application?

This is possible by a process called **Routing**



upGrad

What is Routing?

- Routing is the process that helps in loading partial content, making it a dire need for building SPAs
- Based on the URL that a user visits, specific content is loaded on the page, which helps in displaying different content to users without any need for refreshing the page
- This is when the users get to view the entire application in the form of a single page even though it consists of multiple pages
- Dividing it in components and using routing to load different components, helps in logically bifurcating the app and making it more manageable

upGrad

React Router

- React Router is a library used to handle routing within a React application
- Its primary components include:
  - Routers: <BrowserRouter>, <HashRouter>
  - Route Matchers: <Route>, <Switch>
  - Navigation: <Link>, <NavLink>, <Redirect>

upGrad

react-router-dom

- In order to implement Routing in your application, we will need to use a node package called ‘react-router-dom’. This package provides React components to simulate server-side router handling.
- Though there are many other types of Routers that are available, we will be using a *BrowserRouter* component in our application
- It is a component that watches the URL and more or less passes the current path down to its children, while a *Route* component will render some other component based on the URL information passed to it by its parent *BrowserRouter*

Implement Routing in the Phone Directory Application

- To install the 'react-router-dom' package inside an application, stop the development server and go to application's root folder. Type the command given below inside the Command Prompt (Windows) or Terminal (macOS)

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

  Start the development server again.

upGrad

Implement Routing in the Phone Directory Application

- Go to *PhoneDirectory.js* file and add an import statement for ‘react-router-dom’

```
import { BrowserRouter as Router, Route } from 'react-router-dom';
```

upGrad

Implement Routing in the Phone Directory Application

- Change the *render()* method. Routing for an 'Add Subscriber' page and 'Show Subscribers' page is done in the following way inside the render method using props. Multiple route elements are enclosed in a single tag

```
<Router>
    <div className="main-container">
        <Route exact path='/' render={(props) => <ShowSubscribers
{...props} subscribersList={subscribersList} /> }  />

        <Route path='/add' render={(props) => <AddSubscriber {...props}
onAddSubscriber={this.addSubscriberHandler.bind(this)} /> } />
    </div>
</Router>
```

Code Reference

Route Component

Route component is used to establish the link between component's UI and the URL. Following are the props associated with the Route component:

- **path -** to specify the pathname assigned to component
- **exact** - to match the exact value with the URL
- **component** - to specify the component to be rendered on matching the path

upGrad

Implement Routing in the Phone Directory Application

- Go to *ShowSubscriber.js* file and add an import statement for *Link*. Provide link to the *add* button

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

<Link to="/add"><button className="custom-btn
add-btn">Add</button></Link>
```

- Go to *AddSubscriber.js* file and add an import statement for *Link*. Add link to the *back* button

```
<Link to="/"><button className="custom-btn">Back</button></Link>
```

Code Reference

upGrad

Implement Routing in the Phone Directory Application

- Go to the browser and see how *Add* and *Back* buttons work perfectly and how the routing happens from *ShowSubscribers* page to *AddSubscriber* page and back to the *ShowSubscribers* page
- This is routing which makes a single page application display multiple pages
- Click on *Add* button inside *AddSubscriber* page. It doesn't work. Let's make this work

Making *Add* Button Work on *AddSubscriber* Page

We need to redirect to the Home page when the *Add* button is clicked on *AddSubscriber* page.

- Go to *PhoneDirectory.js* file. Pass history too while passing props to the *AddSubscriber* component. History keeps track of the URL path

```
<Route path='/add' render={({history}, props) => <AddSubscriber {...props}
history={history} addSubscriberHandler={this.addSubscriberHandler.bind(this)}
/> } />
```

Making *Add* Button Work on *AddSubscriber* Page

- Go to *AddSubscriber.js* file and modify the *onFormSubmitted()* method and redirect to homepage while pushing the *homepage* url path in the history

```
onFormSubmitted = (e) => {
    e.preventDefault();
    this.props.addSubscriberHandler(this.state);
    this.props.history.push("/");
}
```

- You can remove the *console.log()* method written inside *addSubscriberHandler()* method written inside *PhoneDirectory.js* file
- See the changes in the browser. Add a subscriber inside *AddSubscriber* page and see how it is added on the home page

Code Reference

Poll 6

Which of the following is NOT true about Routing in React?

A. React router is a framework used for routing in React applications.

B. We need to use a node package called ‘react-router-dom’ to implement routing.

C. Link component is used to navigate around in your application.

D. Routing helps in loading partial content.

Poll 6 (Answer)

Which of the following is NOT true about Routing in React?

A. React router is a framework used for routing in React applications.

B. We need to use a node package called ‘react-router-dom’ to implement routing.

C. Link component is used to navigate around in your application.

D. Routing helps in loading partial content.

Poll 7

Which of the following is NOT a component of React Router?

A. <Link>

B. <BrowserRouter>

C. <Switch>

D. <SwitchRoute>

Poll 7 (Answer)

Which of the following is NOT a component of React Router?

A. <Link>

B. <BrowserRouter>

C. <Switch>

D. <SwitchRoute>

upGrad

Poll 8

Which of the following props of Route component is to specify the pathname assigned to component?

A. path

B. pathTo

C. exact

D. component

Poll 8 (Answer)

Which of the following props of Route component is to specify the pathname assigned to component?

A. path

B. pathTo

C. exact

D. component

Poll 9

Which of the following is true about the <Link> component?

(**Note:** More than one option may be correct.)

A. It is used to render the route that matches the location.

B. It is used to create links to different routes.

C. It is used to implement navigation across the application.

D. It is used to establish the link between component's UI and the URL.

Poll 9 (Answer)

Which of the following is true about the <Link> component?

(**Note:** More than one option may be correct.)

A. It is used to render the route that matches the location.

B. It is used to create links to different routes.

C. It is used to implement navigation across the application.

D. It is used to establish the link between component's UI and the URL.

Deleting a Subscriber

Deleting a Subscriber

upGrad

Let's add one more functionality to the app, which will delete a subscriber's details from the phone directory.

Deleting a Subscriber

upGrad

Phone Directory Application

Let's now see how to delete a subscriber.

- Write a *deleteSubscriberHandler()* method inside the *PhoneDirectory* smart component

```
deleteSubscriberHandler = (subscriberId) => {
    let subscribersList = this.state.subscribersList;
    let subscriberIndex = 0;
    subscribersList.forEach(function (subscriber, index) {
      if (subscriber.id === subscriberId) {
        subscriberIndex = index;
      }
    }, this);
    let newSubscribers = subscribersList;
    newSubscribers.splice(subscriberIndex, 1);
    this.setState({subscribers: newSubscribers});
}
```

Code Reference

Deleting a Subscriber

upGrad

Phone Directory Application

- This method will check for each subscriber in the list of all the subscribers inside its state and delete the one whose *id* matches the *id* received in the function parameter
- The deletion is done using the *splice* method in JavaScript. After deleting the required subscriber, the state will be set again to the updated list

Deleting a Subscriber

upGrad

Phone Directory Application

- Pass the *deleteSubscriberHandler()* method down to *ShowSubscribers* component as props inside the *render* method

```
<Route exact path='/' render={(props) => <ShowSubscribers {...props}
subscribersList={this.state.subscribersList}
deleteSubscriberHandler={this.deleteSubscriberHandler.bind(this)} /> }
/>
```

Code Reference

Deleting a Subscriber

upGrad

Phone Directory Application

- The delete button inside this dumb component will call a local event handler and pass the id of the subscriber to be deleted to this handler
- Go to *ShowSubscriber.js* file and bind a delete event listener to the *Delete* button

```
<button className="custom-btn delete-btn"
onClick={this.onDeleteClicked.bind(this, sub.id)}>Delete</button>
```

- Define this *onDeleteClicked()* method which should call the *deleteSubscriberHandler()* function inside the parent component which is *PhoneDirectory*

```
onDeleteClicked() = (subscriberId) => {
    this.props.deleteSubscriberHandler(subscriberId);
}
```

Code Reference

App Completion

upGrad

Phone Directory Application

Now, a beautiful single-page Phone Directory application has been successfully built. It has the functionalities of adding and deleting a subscriber's details

All the code used in today's session can be found in the link provided below (branch session6-demo):

https://github.com/upgrad-edu/react-class-components/tree/session6-demo

Doubt Clearance (5 minutes)

Important Questions

Important Questions

upGrad

1. What is React Router? Why do we need it?
2. How is React routing different from conventional routing?
3. What is the use of **Switch** component?
4. Why you get the "Router may have only one child element" warning?
5. How to implement the default or NotFound page?

Key Takeaways

- A Single-Page Application (SPA) works on the browser and does not require a page to be reloaded completely. React helps you build dynamic SPAs.
- A Multi-Page Applications (MPA) is an application that consists of multiple pages while embedding links to other pages in it. When moving from one page to another, the browser reloads the contents of the page completely and downloads all the resources again.
- A dumb (also known as presentational) component only presents data on the DOM. On the other hand, a smart (also known as container) component provides data and logic to the dumb components
- Dumb components describe the appearance of things, whereas smart components describe the working of things.

Key Takeaways

- Routing is the process that helps in loading partial content, making it a dire need for building SPAs.
- Based on the URL that a user visits, specific content is loaded on the page, which helps in displaying different content to users without any need for refreshing the page. This is when users can view the entire application in the form of a single page although it consists of multiple pages

These tasks are to be completed after today's session:

MCQs
Coding Questions
Course Project (Part A) - Checkpoint 5

In the next class, we will discuss...

1. Introduction to React Hooks
    - Class-based vs functional components
    - Why Hooks?
2. Properties in Hooks
    - Using state in functional components
3. Routing in Hooks
    - Hooks in the React Router library

upGrad

#RahoAmbitious

Thank You!