upGrad

#RahoAmbitious

Full Stack Software Development

upGrad

Course: Advanced Frontend Development Using React

Lecture on: Props and Event Handling

In the last class, we discussed...

- Components, its types and conditions
- Inline styling in React
- External styling in React
- Dynamic rendering using the map() method

Poll 1

Which of the following is NOT a characteristic of inline styling?

A. All property-value pairs are separated using the comma operator.

B. Property names must be written in camelCase.

C. The *style* property is used with the element or component to be rendered into the DOM.

D. Three curly braces are used with the style property.

Poll 1 (Answer)

Which of the following is NOT a characteristic of inline styling?

A. All property-value pairs are separated using the comma operator.

B. Property names must be written in camelCase.

C. The *style* property is used with the element or component to be rendered into the DOM.

D. Three curly braces are used with the style property.

Poll 2

What is the use of the map() method?

A. It is used to map the objects of one array to another.

B. It is used to iterate over an array and inject data into the React elements dynamically.

C. It is used to render elements into DOM in React.

D. None of the above.

Poll 2 (Answer)

What is the use of the map() method?

A. It is used to map the objects of one array to another.

B. It is used to iterate over an array and inject data into the React elements dynamically.

C. It is used to render elements into DOM in React.

D. None of the above.

Today's Agenda

1. Props and how to use them inside a functional as well as a class component
2. Events and event handling
3. Developing a functionality for adding a subscriber

Props

Props

upGrad

Have you wondered if it is possible to pass information from one component to another?

Can we make the header display different content dynamically on different pages?

Props

upGrad

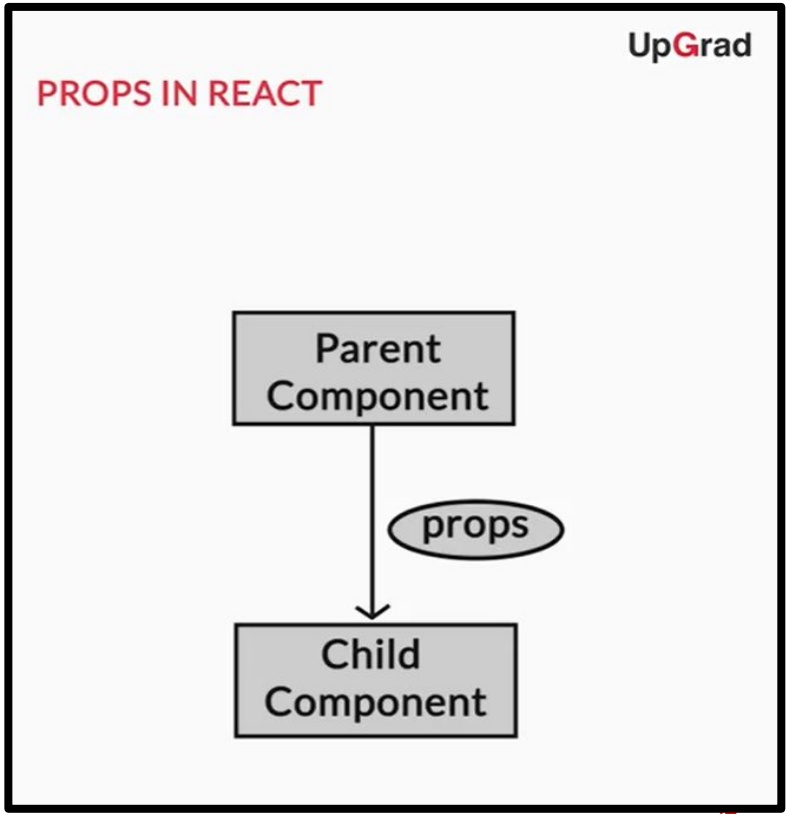
It is possible and can be done using **props**. When React sees an element representing a user-defined component, it passes JSX attributes to this component as a single object. We call this object as “props”.

We can see this by displaying the header text dynamically on different pages in our Phone Directory application using props.

Props

upGrad

- Props stands for properties
- Props help you to pass values from a parent component to a child component so that they can be accessed within the child component
- Also, the flow of props is unidirectional from parent to child and not from child to parent



upGrad

Props in a Functional Component

Let’s see how we can use props in a functional component:

- A functional component accepts a parameter called props from the parent component
- This parameter is an object that holds all the properties passed from the parent component to the child component.
- In place of props, you can use any other parameter name

upGrad

Props in a Functional Component

Here's an example of using props in a functional component:

```
const Organization = function(props) {
  return (
     <div>
         <h1>{props.name}</h1>
      <h3>{props.tagline}</h3>
     </div>
  )
}
<Organization name="UpGrad" tagline="Building Careers of Tomorrow"/>
```

upGrad

Props in a Class Component

Let’s see how we can use props in a class component:

- The properties passed from the parent component can be accessed using the *this.props* keyword
- Note that in class components, you need to use the keyword **props** only, whereas in functional components, any parameter name can be used to represent the props of the component

upGrad

Props in a Class Component

Here's an example of using props in a class component:

```
class Organization extends Component {
  render() {
    return (
      <div>
        <h1>{this.props.name}</h1>
        <h3>{this.props.tagline}</h3>
      </div>
    )
  }
}
<Organization name="UpGrad" tagline="Building Careers of Tomorrow"/>
```

Phone Directory application

Now let's see how we can use props to display the header text dynamically on different pages in our Phone Directory application

- Edit the *App.js* file to send the props to the *Header* component.
  - Pass the property-value pairs while rendering the component

```
<Header heading="Phone Directory"/>
```

Props

upGrad

Phone Directory application

- Edit the *Header.js* file to send heading as props to the *Header* component.
  - props as the component argument
  - access property using name

```
const Header = function(props) {
  return(
    <div className="header">
      {props.heading}
    </div>
  )
}
```

- Thus, using props, you can dynamically inject values in any component. This is one of the things that makes React dynamic in nature.

Code Reference

Poll 3

Which of the following is used to pass data from one component to another?

A. Using curly braces {}

B. render with arguments

C. props

D. map() method

Poll 3 (Answer)

Which of the following is used to pass data from one component to another?

A. Using curly braces {}

B. render with arguments

C. props

D. map() method

Poll 4

Which of the following statements are true?

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

A. A React component can pass back props to its parent component.

B. A React component can indirectly communicate with its parent by invoking a function passed down as a prop.

C. React supports unidirectional data flow.

D. If a value of a DOM element is bound to a prop, e.g.,
`<input type="text" value={this.prop.name}" />`
this creates a two-way binding in React.

Poll 4 (Answer)

Which of the following statements are true?

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

A. A React component can pass back props to its parent component.

B. A React component can indirectly communicate with its parent by invoking a function passed down as a prop.

C. React supports unidirectional data flow.

D. If a value of a DOM element is bound to a prop, e.g.,
`<input type="text" value={this.prop.name}" />`
this creates a two-way binding in React.

Event Handling

Events and Event Handling

What is an event?

- An event is an action to be taken as a result of user interactions
- An event handler is a method to be called when an event occurs
- A programmer can define a series of steps inside an event handler that can be followed when a specified event occurs

Notes:

- An event name should follow camelCase
- It is good practice to prefix event handlers with ‘on’ such as ‘onSubmitOrder’ or add a suffix with ‘Handler’ such as ‘submitOrderHandler’. This is done to clarify the functions of handlers and the events to which they are attached

To learn more about all the supported events in React, follow **this** link.

upGrad

Phone Directory Application

In the Phone Directory application, we have a Delete button. Let’s see how to bind event handlers in React and see it in action while adding event listeners to this button.

- In the *App.js* file, you can define the ‘deleteHandler’ function and pass it as an event handler in the Delete button.

```
deleteHandler() {
  alert("Delete Clicked");
}
```

- Add the event handler to the delete button as follows:

```
<button className="custom-btn delete-btn"
onClick={this.deleteHandler}> Delete </button>
```

Events and Event Handling

Now, what if you need to pass down some value, let's say you want to pass the name of the button you clicked and have only one event handler to do the job?

- For this purpose, you would use the **bind()** method
- 'this' - default argument to be always passed while binding the handler

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

- Add the clickHandler method now in the component

```
clickHandler(buttonName) {
  alert(buttonName + " clicked!");
}
```

Phone Directory Application

Later on, you'll learn how to add the delete functionality, which will be used to delete a subscriber from the Phone Directory application. This is why you do not need this *onClick* event handler right now.

So, delete the *onClick* event that you have bind with the *Delete* button. Also, delete the clickHandler function defined for the *onClick* event on the *Delete* button.

Poll 5

Choose the correct option of the on-click event handler to an element as per the best practices followed for the naming convention in React.

A. submitOrderHandler

B. submitorderhandler

C. submitOrderhandler

D. SubmitOrderHandler

Poll 5 (Answer)

Choose the correct option of the on-click event handler to an element as per the best practices followed for the naming convention in React.

A. submitOrderHandler

B. submitorderhandler

C. submitOrderhandler

D. SubmitOrderHandler

Poll 6

Choose the correct code snippet on how to call the *editHandler* function of the class component with an argument “Edit Button Clicked”.

A. `this.bind("Edit Button Clicked");`

B. `this.editHandler(this, "Edit Button Clicked");`

C. `this.editHandler.bind("Edit Button Clicked");`

D. `this.editHandler.bind(this, "Edit Button Clicked");`

Poll 6 (Answer)

Choose the correct code snippet on how to call the *editHandler* function of the class component with an argument “Edit Button Clicked”.

A. `this.bind("Edit Button Clicked");`

B. `this.editHandler(this, "Edit Button Clicked");`

C. `this.editHandler.bind("Edit Button Clicked");`

D. `this.editHandler.bind(this, "Edit Button Clicked");`

Poll 7

Which of the following is correct syntax for a button click event handler, onClickHandler?

A. <button onClick={onClickHandler()}>

B. <button onclick={onClickHandler}>

C. <button onClick={this.onClickHandler()}>

D. <button onClick={this.onClickHandler}>

Poll 7 (Answer)

Which of the following is correct syntax for a button click event handler, onClickHandler?

A. <button onClick={onClickHandler()}>

B. <button onclick={onClickHandler}>

C. <button onClick={this.onClickHandler()}>

D. <button onClick={this.onClickHandler}>

Adding Subscriber

Adding a Subscriber

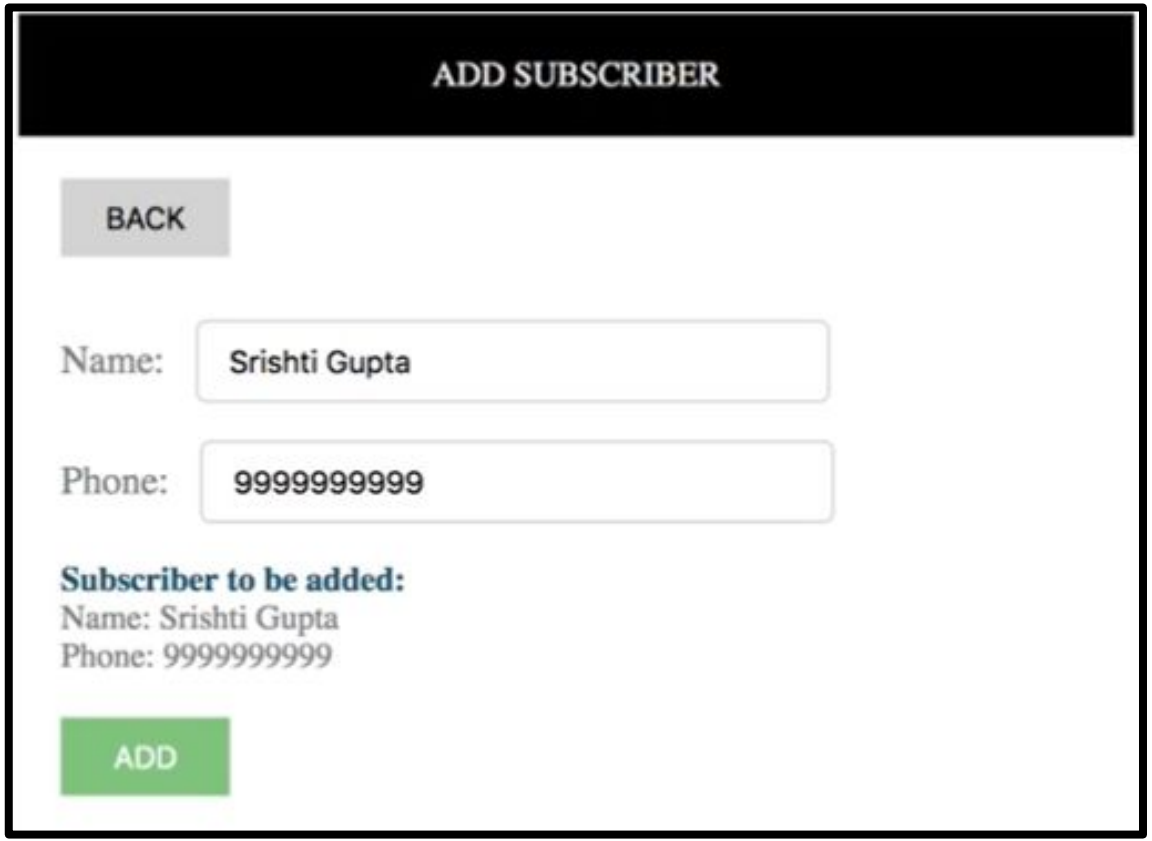
upGrad

Now, let’s add one more functionality to the application, which will add a subscriber to the phone directory.

Adding a Subscriber

upGrad

ADD SUBSCRIBER Page



The ADD SUBSCRIBER page will look like this.

Adding a Subscriber

upGrad

AddSubscriber as a Class Component

- Inside *src* folder, create a *AddSubscriber.js* file
- Make AddSubscriber as a class component

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

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

export default AddSubscriber;
```

Code Reference

Adding a Subscriber

upGrad

AddSubscriber as a Class Component

- Go to *index.js* file and render *AddSubscriber* in *ReactDOM.render()* method

```
import AddSubscriber from './AddSubscriber';
ReactDOM.render(<AddSubscriber/>, document.getElementById('root'));
```

Code Reference

Adding a Subscriber

upGrad

AddSubscriber as a Class Component

- Since header is a common component between homepage which is *App.js* and *AddSubscriber* page, create a new folder inside *src* folder named as ‘common’ and move ‘Header.js’ inside this folder. Also move *Header.css* file required by *Header.js* file inside the *common* folder
- Remember to change the import statement in *App.js* file because the relevant file path of Header file has changed

```
import Header from './common/Header';
```

Code Reference

Adding a Subscriber

upGrad

AddSubscriber as a Class Component

- Render header on the Add Subscriber page

```
import Header from './common/Header';

return (
   <div className="component-container">
      <Header heading="Add Subscriber"/>
   </div>
);
```

Code Reference

Adding a Subscriber

upGrad

AddSubscriber as a Class Component

- Render ‘Back’ button

```
<div className="component-container">
  <Header heading="Add Subscriber" />

  <div className="component-body-container">
    <button className="custom-btn">Back</button>
  </div>
</div>
```

Code Reference

Adding a Subscriber

upGrad

AddSubscriber as a Class Component

- Create a *common.css* file inside *common* folder and add all the common styles over there. *component-body-container* is already present in *App.js* file and is required by *AddSubscriber.js* too. Cut *component-body-container* class code from *App.css* and paste it in *common.css* file

```
.component-body-container {
  padding: 20px;
}
```

- Also, don't forget to include this *common.css* file in *App.js* and *AddSubscriber.js* files using an import statement

```
import './common/common.css';
```

Code Reference

Adding a Subscriber

upGrad

AddSubscriber as a Class Component

- So far, **`AddSubscriber`** is built as a class component
- Currently, it renders a ‘Back’ button. The ‘Back’ button is styled even though there is no css import because of webpack
- You have also defined the common styles that will be used to style elements that are spread over the entire application using *common.css* file

Adding a Subscriber

upGrad

AddSubscriber as a Class Component

- Add a form in the *AddSubscriber.js* file

```
<div className="component-body-container">
   <button className="custom-btn">Back</button>

   <form className="subscriber-form">
   </form>
</div>
```

Adding a Subscriber

upGrad

AddSubscriber as a Class Component

- In the *AddSubscriber.js* file, add a label and input textbox for name of subscriber in the form

```
<form className="subscriber-form">
  <label htmlFor="name" className="label-control">Name:</label><br />
  <input id="name" type="text" className="input-control" name="name" /><br /><br />
</form>
```

Code Reference

Adding a Subscriber

upGrad

AddSubscriber as a Class Component

- Create a new file *AddSubscriber.css* to contain CSS specific code for *AddSubscriber.js* file.
- Import this file inside *AddSubscriber.js f*ile.

```
import './AddSubscriber.css';
```

Adding a Subscriber

upGrad

AddSubscriber as a Class Component

- Write CSS for *subscriber-form, label-control* and *input-control* classes in the *AddSubscriber.css* file

```
.subscriber-form {
  margin-top: 30px;
}
```

```
.label-control {
  color: #808080;
  font-size: 1.15rem;
  margin-right: 15px;
}
```

```
.input-control {
  width: 300px;
  min-width: 100px;
  padding: 10px 15px;
  display: inline-block;
  border: 1px solid #ccc;
  border-radius: 5px;
  box-sizing: border-box;
  font-size: 0.9rem;
  outline: none;
}
```

Code Reference

Adding a Subscriber

upGrad

AddSubscriber as a Class Component

- Add one more label and input textbox for phone number of subscriber in the form

```
<label htmlFor="phone" className="label-control">Phone:</label><br />
<input id="phone" type="text" className="input-control" name="phone" /><br
/><br />
```

- Add more fields which will display the name and phone number of the subscriber to be added in the form

```
<div className="subscriber-info-container">
  <span className="subscriber-to-add-heading">Subscriber to be added:</span><br
   />
  <span className="subscriber-info">Name: </span><br />
  <span className="subscriber-info">Phone: </span>
</div>
```

Code Reference

Adding a Subscriber

upGrad

AddSubscriber as a Class Component

```
.subscriber-info-container {
  font-size: 1rem;
  margin-right: 10px;
  margin-bottom: 20px;
}

.subscriber-to-add-heading {
  color: #0a4f75;
  font-weight: 600;
}

.subscriber-info {
  color: #808080;
}
```

Write CSS for *subscriber-info-container*, *subscriber-to-add-heading* and *subscriber-info* classes in the *AddSubscriber.css* file

Code Reference

Adding a Subscriber

upGrad

AddSubscriber as a Class Component

- Add an ‘Add’ button inside this form

```
<button type="submit" className="custom-btn add-btn">Add</button>
```

- ‘Add’ Button is already present in the home page. Let’s cut *custom-btn* and *add-btn* CSS classes from *App.css* file and paste it into *common.css* file

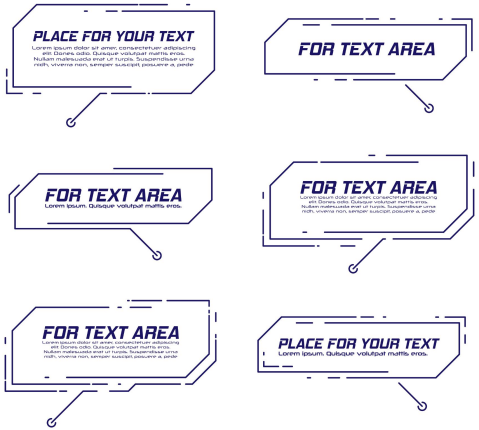
Code Reference

Adding a Subscriber

upGrad

AddSubscriber as a Class Component

- You have completed the **`AddSubscriber`** component
- It renders a textbox for the name and the phone number of a subscriber
- You have also rendered the skeleton of the section that will display the name and the phone number



Poll 8

Which is the purpose of the common folder created in the Phone Directory application?

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

A. To store the details that are common across different pages of the application

B. To avoid redundancy of code

C. To store any components of the application

D. To add unique styling to some components of the application

Poll 8 (Answer)

Which is the purpose of the common folder created in the Phone Directory application?

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

A. To store the details that are common across different pages of the application

B. To avoid redundancy of code

C. To store any components of the application

D. To add unique styling to some components of the application

All the code used in today's session can be found in the link provided below (branch session4-demo):

https://github.com/upgrad-edu/react-class-components/tree/session4-demo

Important Questions

Important Questions

upGrad

1. What is the difference between super() and super(props) in React using ES6 classes?
2. Why can't you update props in React? How do you say that props are read-only?
3. What is the difference between HTML and React event handling?
4. How to bind methods or event handlers in JSX callbacks?
5. How do you pass an event handler to a component?

Key Takeaways

- Props help you to pass values from a parent component to a child component so that they can be accessed within the child component
- An event handler is a method to be called when an event occurs
- An event name should follow camelCase
- It is good practice to prefix event handlers with ‘handle’ such as ‘handleSubmitOrder’ or suffix them with ‘handler’ such as ‘submitOrderHandler’. This is done to clarify the functions of handlers and the events to which they are attached

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

MCQs
Coding Questions
Course Project (Part A) - Checkpoint 3

In the next class, we will discuss...

1. Introduction to State
2. Characteristics of State
3. Setting the state using the *setState()* method
4. Lifecycle of components and the processes of mounting, updating and unmounting

upGrad

#RahoAmbitious

Thank You!