5 - React (events & conditionals)

React education, 2024.

MITHRIL

Overview

- Events
- Conditional rendering
- Lists & keys
- Forms

Events

Events

- Handling events with React elements is very similar to handling events on DOM elements.
- There are some differences:
  - React events are named using camelCase, rather than lowercase.
  - With JSX you pass a function as the event handler, rather than a string.

```
<button onclick="activateLasers()">
  Activate Lasers
</button>
```

```
<button onClick={activateLasers}>
  Activate Lasers
</button>
```

MITHRIL

Events

- Another difference between classic HTML and React is that you can't *return false* to prevent default behavior in React
- You must call *preventDefault* explicitly.
- In the function handleClick() you can see e.preventDefault(), e stands for synthetic event which is defined according to W3C spec.
- Read more about SyntheticEvent.

```
<a href="#" onclick="action(); return false">
Click me
</a>
```

```
function ActionLink() {
  function handleClick(e) {
    e.preventDefault();
    console.log('The link was clicked.');
  }

  return (
    <a href="#" onClick={handleClick}>
      Click me
    </a>
  );
}
```

MITHRIL

Events

- On the previous slides, we have learned how App can communicate between components in the Top-Down order using props.
- What if we need to communicate from child to parent component (Down-Top)?
  - Then parent component passes callback to the child component as on of its props.
  - Then when the child needs to communicate information upwards to the parent, it calls the callback.
- In simple terms, we need to pass a reference to handler as a prop to child component.

Events

- Top component has some initial state, and handler that can change initial state value.
  - Inside Top component we are passing prop *onIncrementClick* which passes *handleIncrement* callback to Down component.
- Down component just need to call that callback to change the state of Top component.

```jsx
import React, { useState } from 'react';

import Down from '../Down/Down';

const Top = () => {
    const [increment, setIncrement] = useState(0);

    const handleIncrement = () => {
        setIncrement(increment + 1);
    }

    return (
        <div>
            <p>Result of increment: <strong>{increment}</strong></p>
            <Down onIncrementClick={handleIncrement} />
        </div>
    );
}

export default Top;
```

```jsx
import React from 'react';

const Down = (props) => {
    return (
        <button onClick={() => props.onIncrementClick()}>
        Increment</button>
    );
}

export default Down;
```

MITHRIL

Conditional rendering

Conditional rendering

- In React you can create distinct components that encapsulate behavior you need. Then you can render only some of them, depending on the state of your application.
- Conditional rendering works the same way conditions work in JavaScript.
- In React we can use following:
  - Element variables
  - if, if-else
  - Inline if with logical && operator
  - Inline if-else with conditional operator

MITHRIL

Conditional rendering

- In this example, you can see use of element variable, if, inline if with && operator and inline if-else with conditional operator.
- You can use any of these, be careful not to overuse it in return method.
- You can use user-defined methods to conditionally return some elements and then call that method in JSX.

```
const displaySubheader = (props) => {
    let content = '';

    if (!props.typeSpeakers) {
        content = (
            <div className="MyComponent-Subheader">
                My subheader
            </div>
        );
    }

    return content;
}

const MyComponent = (props) => {
    return (
        <div className="MyComponent">
            <div className="MyComponent-Header">
                {props.name.length > 0 && <h2>{props.name}</h2>}
            </div>
            {displaySubheader(props)}
            <p className="MyComponent-About">
                {props.about}
            </p>
            <span>The user is {props.isLoggedIn ? 'currently' : 'not'}
            logged in.</span>
        </div>
    );
}

export default MyComponent;
```

MITHRIL

Lists & keys

Lists & keys

- React allows us transforming arrays into lists of elements and include them in JSX using curly braces.
- For iteration through arrays we are using map() function which creates new array populated with the results of calling a provided function on every element in the calling array.

```
const Fruits = () => {

    const fruitItems = fruits.map((fruit, index) => (
        <Fruit
            key={index}
            title={fruit.title}
            about={fruit.about}
        />
    ));


    return (
        <>
            <PageTitle>Fruits</PageTitle>
            <Section>
                {fruits ? fruitItems : <SomeOtherComponent />}
            </Section>
        </>
    );
}

export default Fruits;
```

MITHRIL

Lists & keys

- When creating list of elements in array you need to include special string attribute *key* which helps React identify which items have changed, are added, or are removed.
- The best way to pick a *key* is to use **string that uniquely identifies** list item, most often these will be ID of item in array.
- Use index only in case when you don’t have IDs attached to your elements in array.

```
const Fruits = () => {

    const fruitItems = fruits.map((fruit, index) => (
        <Fruit
            key={index}
            title={fruit.title}
            about={fruit.about}
        />
    ));


    return (
        <>
            <PageTitle>Fruits</PageTitle>
            <Section>
                {fruits ? fruitItems : <SomeOtherComponent />}
            </Section>
        </>
    );
}

export default Fruits;
```

Lists & keys

- Keys used within array should be unique among their siblings.
- Keys serve as a hint to React but they don’t get passed to your components. If you need the same value in your component, pass it explicitly as a prop with a different name.
- You can also embed *map()* in JSX

```
const Fruits = () => {
    return (
        <>
            <PageTitle>Fruits</PageTitle>
            <Section>
                {fruits ? (
                    fruits.map((fruit, index) => (
                        <Fruit
                            key={index}
                            title={fruit.title}
                            about={fruit.about}
                        />
                    ))
                ) : (<SomeOtherComponent />)}
            </Section>
        </>
    );
}

export default Fruits;
```

MITHRIL

Forms

Forms

- HTML form elements work a little bit differently from other DOM elements in React, because form elements naturally keep some internal state.
- Default HTML form behaviour is to send submitted data to another or the same script, but in most cases we don’t want this behaviour.
- In most cases we want to have JS function that handles the submission of the form and has access to the data that user entered into the form.
- In React we can handle form by use of Controlled and Uncontrolled components.

Controlled and Uncontrolled components

- In a controlled component, form data is handled by a React component. The alternative is uncontrolled components, where form data is handled by the DOM itself.
- Read more about the differences here.
- Common libraries that allow us to implement complex forms easily - React Hook Form, Formik