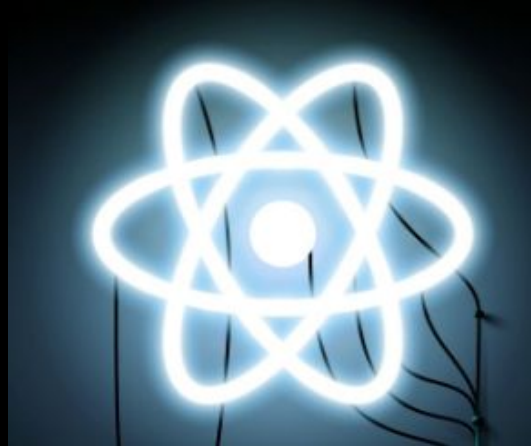


FS IV- Lecture

Intro to React Hooks



Topics

- Component Review
- Disadvantages of Components
- React Hooks!

Components Review

Functional vs Class Component Recap

Functional Component

```
import React from 'react';

const Greeter = React.createClass({
  render: function() {
    return <div>Hello {this.props.firstName}</div>;
  }
});

export default Greeter;
```

Class Component

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

class Greeter extends Component {
  render() {
    return <div>Hello {this.props.firstName}</div>;
  }
}

export default Greeter;
```

Class Component - State and Life Cycle

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

class Greeter extends Component {
  state = {
    loaded: false
  };

  componentDidMount() {
    this.setState({ loaded: true });
  }

  render() {
    return <div>Hello {this.props.firstName}</div>;
  }
}

export default Greeter;
```

Class Components - Disadvantages

Class Component Disadvantage #1 - No Autobinding

```
1 class Counter extends Component {  
2   state = { count: 0 };  
3  
4   onClick() {  
5     this.setState({ count: this.state.count + 1 });  
6   }  
7 }  
8  
9 <button onClick={this.onClick}>  
10   Click Me  
11 </button>  
12  
13 );  
14 }  
15 }
```

```
Uncaught TypeError: Cannot read property 'setState' of undefined  
    at onClick (Counter.js:79)  
    at HTMLUnknownElement.callCallback (react-dom.development.js:147)  
    at Object.invokeGuardedCallbackDev (react-dom.development.js:196)  
    at invokeGuardedCallback (react-dom.development.js:250)  
    at invokeGuardedCallbackAndCatchFirstError (react-dom.development.js:265)  
    at executeDispatch (react-dom.development.js:622)  
    at executeDispatchesInOrder (react-dom.development.js:647)
```

- React components using ES6 classes **no longer** autobind **this** to non React methods.

- In your constructor, you need to add:

```
this.onChange =  
  this.onChange.bind(this)
```

Class Component Disadvantage #2 - Single Responsibility

```
class Clock extends Component {
  state = { now: new Date().toLocaleTimeString() };

  componentDidMount() {
    this.handle = setInterval(
      () =>
        this.setState({
          now: new Date().toLocaleTimeString()
        }),
      1000
    );
  }

  componentWillUnmount() {
    clearInterval(this.handle);
  }

  render() {
    return <div>{this.state.now}</div>;
  }
}
```

- Single responsibility in multiple functions
- Unrelated mixed in logic in lifecycle methods can get repetitive, and cause unnecessary side effects

Class Component Disadvantage #3 - Deeply Nested Components

```
▼ <ul style={} className="ant-menu ant-menu-inline ant-menu-sub" role="menu"
  ▼ <Connect(OnError) key=".71" ref=chainedFunction() connectionId="c183:
    ▼ <OnError connectionId="c18323cd-c841-430f-90a2-d777b6a9d273" mode="j
      ▼ <MenuItem key="71" connectionId="c18323cd-c841-430f-90a2-d777b6a9d27
        ▼ <Tooltip title="" placement="right" overlayClassName="ant-menu-inli
          ▼ <Tooltip ref=ref() title="" placement="right" overlayClassName="ar
            ▼ <Trigger ref=fn() popupClassName="ant-menu-inline-collapsed-tool
              ▼ <MenuItem connectionId="c18323cd-c841-430f-90a2-d777b6a9d273"
                ▼ <li style={paddingLeft: 48} className="ant-menu-item-selectec
                  ▶ <Icon type="frown-o">_</Icon>
                    "PCS_MISSING"
                  ▼ <Connect(ImportPlugin) url="shader">
                    ▼ <ImportPlugin url="shader" dispatch=fn()>
                      ▼ <Connect(Plugin) id="76de5df9-ee25-4404-bf37-3c5e9d54e9d
                        ▼ <Plugin id="76de5df9-ee25-4404-bf37-3c5e9d54e9da" item=
                          ▼ <div style={color: "rgba(0, 0, 0, 0.65)", fontSize: "
                            ▼ <Connect(Element) key="ce6785d9-790a-4e90-83da-3d81:
                              ▼ <Element id="ce6785d9-790a-4e90-83da-3d81e0510dee"
                                ▼ <Connect(GroupElement) handleEvent=fn() style={d:
                                  ▼ <GroupElement handleEvent=fn() style={display:
                                    ▼ <Fields handleEvent=fn() style={display: "blo
                                      ▼ <div style={width: "100%", display: "flex",
                                        ▶ <Connect(Row) key="5c36ce5a-a69d-4aa3-9018:
                                        ▶ <Connect(Row) key="30fe4229-5544-44d5-81bd:
                                        ▶ <Connect(Row) key="66ba662d-03b0-4057-9c60:
                                        ▶ <Connect(Row) key="7281bab2-f3d4-4832-9116:
                                        ▼ <Connect(Row) key="ccc62da8-6826-456c-8ed1:
                                          ▼ <Row id="ccc62da8-6826-456c-8ed1-2d5e94fc
                                            ▼ <div style={marginBottom: 2, display: "
```

- Managing State: Reusing logic between multiple components can lead to wrapper hell or deeply nested components.

Functional Components - Disadvantages

Functional Component Disadvantage #1 - No State & Life Cycle!

```
import React from 'react';

const Greeter = props => {
  return <div>Hello {props.firstName}</div>;
};

export default Greeter;
```

Functional Component Disadvantage #2 - Deep Nesting

```
▼ <PresTrackedContainer className="billboard-presentation-tracking">
  ▼ <div className="ptrack-container billboard-presentation-tracking">
    ▼ <getTrackingInfoFromContext(createTrackingComponent(billboard)) className="billboard-presentation-trac
      " videoId={80190859}>
      ▼ <createTrackingComponent(billboard) className="billboard-presentation-tracking" imageKey="BILLBOARD
        ▼ <div className="billboard-presentation-tracking ptrack-content">
          ▼ <getTrackingInfoFromContext(createTrackingComponent(boxArt)) className="billboard-presentation
            BILLBOARD|6d853480-ce72-11e8-b627-0e319b527290|en" videoId={80190859}>
            ▼ <createTrackingComponent(boxArt) className="billboard-presentation-tracking" imageKey="BILL
              >
            ▼ <div className="billboard-presentation-tracking ptrack-content">
              ▼ <logPresentationManually(getTrackingInfoFromContext(windowVisibility(inViewport(Connec
                BILLBOARD|6d853480-ce72-11e8-b627-0e319b527290|en" videoId={80190859} backgroundImageS
                useAvailablePhase={true}>
              ▼ <getTrackingInfoFromContext(windowVisibility(inViewport(ConnectToApps(e)))) isMotion
                BILLBOARD|6d853480-ce72-11e8-b627-0e319b527290|en" videoId={80190859} backgroundIma
                useAvailablePhase={true}>
              ▼ <windowVisibility(inViewport(ConnectToApps(e))) isMotionEnabled={true} imageKey="
                80190859) backgroundImageStartsPlay={false} trackId={254015180} hasScrolled={true}
              ▼ <inViewport(ConnectToApps(e)) isMotionEnabled={true} imageKey="BILLBOARD|6d853
                backgroundImageStartsPlay={false} trackId={254015180} hasScrolled={true} useAv
                ={false} ignoreElementWithNoDimensions={false}>
              ▼ <ConnectToApps(e) isMotionEnabled={true} imageKey="BILLBOARD|6d853480-ce72-1
                backgroundImageStartsPlay={false} trackId={254015180} hasScrolled={true} use
                defaultInViewportState={false} ignoreElementWithNoDimensions={false} inViewp
              ▼ <e muted={true} isMotionEnabled={true} imageKey="BILLBOARD|6d853480-ce72-
                backgroundImageStartsPlay={false} trackId={254015180} hasScrolled={true}
```

- Deeply Nested Components
- Many are wrappers to add props

WHAT IF I TOLD YOU

FUNCTIONS CAN HAVE STATE

React Hooks!

React Hooks!

- Basic hooks
 - `useState()`
 - `useEffect()`
 - `useContext()`
- Additional Hooks
 - `useReducer()`
 - `useMutationEffect()`
 - `useLayoutEffect ()`
 - `useCallback ()`
 - `useMemo()`
 - `useRef()`
 - `useImperativeMethods()`
- Custom hooks

- Hooks are a new addition in React 16.8.
- They let you use state and other React features without writing a class.

useState() Hook

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

const Counter = () => {
  const [count, setCount] = useState(1);

  return (
    <div>
      <p>The counter is: {count}</p>
      <button onClick={
        () => setCount(count + 1)}>
        Increment
      </button>
    </div>
  );
};

export default Counter;
```

- The *State Hook* lets you have state in functional components without using class

Class Component Example

```
class Example extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = {  
      count: 0  
    };  
  }  
  
  render() {  
    return (  
      <div>  
        <p>You clicked {this.state.count} times</p>  
        <button onClick={() => this.setState({ count: this.state.count + 1 })}>  
          Click me  
        </button>  
      </div>  
    );  
  }  
}
```

Equivalent Function Component - `useState` Example

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

function Example() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

useEffect() Hook

```
import React, { useEffect, useState } from 'react';
import AnalogClock from './AnalogClock';

const Clock = ({ interval }) => {
  const [time, setTime] = useState(new Date());

  useEffect(
    () => {
      const handle = setInterval(
        () => setTime(new Date()), interval);

      return () => clearInterval(handle);
    },
    [interval]
  );

  return <AnalogClock time={time} />;
};

export default Clock;
```

- the *Effect Hook* lets you perform side effects in function components

```
class Example extends React.Component {  
  constructor(props) {  
    super(props);  
    this.state = {  
      count: 0  
    };  
  }  
}
```

```
  componentDidMount() {  
    document.title = `You clicked ${this.state.count} times`;  
  }  
  
  componentDidUpdate() {  
    document.title = `You clicked ${this.state.count} times`;  
  }  
}
```

```
  render() {  
    return (  
      <div>  
        <p>You clicked {this.state.count} times</p>  
        <button onClick={() => this.setState({ count: this.state.count + 1 })}>  
          Click me  
        </button>  
      </div>  
    );  
  }  
}
```

Equivalent Function Component - `useEffect` Example

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

function Example() {
  const [count, setCount] = useState(0);

  // Similar to componentDidMount and componentDidUpdate:
  useEffect(() => {
    // Update the document title using the browser API
    document.title = `You clicked ${count} times`;
  });

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```



Context API and Hooks

Context API

- Context is designed to share data that can be considered “global” for a tree of React components, such as the current authenticated user, theme, or preferred language

```
import { createContext } from 'react';  
  
const TimeContext = createContext(new Date());  
  
export default TimeContext;
```


Context API

```
import React, { Component } from 'react';
import TimeContext from './TimeContext';
import ThemeContext from './ThemeContext';
import AnalogClock from './AnalogClock';

class Clock extends Component {
  render() {
    return (
      <TimeContext.Consumer>
        ({ { time } }) => (
          <ThemeContext.Consumer>
            ({ { theme } }) => (
              <AnalogClock time={time} theme={theme} />
            )
          </ThemeContext.Consumer>
        )
      </TimeContext.Consumer>
    );
  }
}

export default Clock;
```

- Context provides a way to pass data through the component tree without having to pass props down manually at every level.

useContext() Hook

```
import React, { useContext } from 'react';
import TimeContext from './TimeContext';
import ThemeContext from './ThemeContext';
import AnalogClock from './AnalogClock';

const Clock = () => {
  const time = useContext(TimeContext);
  const theme = useContext(ThemeContext);

  return <AnalogClock time={time} theme={theme} />;
};

export default Clock;
```

- *Context Hook*, gives functional components easy access to your context

Rules with Hooks

- Hooks can only be used in functional components
 - Or in other hooks
 - Not in class based components
- Hooks must be created in the same order
 - Must be used outside loops, conditions or nest functions
- Hooks names must be prefixed with 'use'