# HIGHER-ORDER COMPONENTS.

# STATE HOISTING.

# CONTROLLED INPUT.

by Anastasiia Derkach

softserve

# HIGHER-ORDER COMPONENTS

## What does it mean?

Higher-order component is a function that takes a component and returns a new component. A higher-order component (HOC) is the advanced technique in React.js for reusing a component logic. Higher-Order Components are not part of the React API. HOCs are common in third-party React libs, such as Redux or React Router.

## Why it's important?

Higher-order functions allow us to abstract over actions, not just values. HOCs are common in third-party React libs, such as Redux or React Router. The main purpose of a higher-order component in React is to share common functionality between components without repeating code.

We have greeting component.

```
const Greeting = ({ name }) => {
    if (!name) { return <div>Connecting...</div> }
    return <div>Hi {name}!</div>
}
```

If it gets props.name, it's gonna render that data. Otherwise it'll say that it's "Connecting…". Now for the the higher-order

```
const Connect = ComposedComponent =>
  class extends React.Component {
    constructor() {
      super()
      this.state = { name: "" }
    }

    componentDidMount() {
      this.setState({ name: "Michael" })
    }

    render() {
      return (
        <ComposedComponent
          {...this.props}
          name={this.state.name}
        />
    )}
}
```

softserve

This is just a function that returns component that renders the component we passed as an argument.

```
const ConnectedMyComponent = Connect(Greeting)
```

This is a powerful pattern for providing fetching and providing data to any number of stateless function components.

softserve

# STATE HOISTING

## Why it's important?

As the application grows you will realise that some components will need common data or actions in one component may need to cause another component to re-render as well.

1.Most child components won't need state vis a vis not need lifecycle callbacks, making it possible to use pure functions as components these making them easier to test and maintain.

2. Single source of truth. Child components will have one unit to find information from. What this gives you is a more stable application and one that is easier to debug.

3. Separation of concerns when we dedicate some components to rendering and others to data fetching and manipulation we have a nice abstraction between our data and user interfaces this gives a nice stable application to work with and debug.

You are encouraged to lift state up, if two components need to act on the same data or need to use the same callback. It mean that you should create a common ancestor in this common ancestor and then use the state to manage all the data and callbacks that children will use in rendering.

```
class NameContainer extends React.Component {
    constructor() {
      super();
      this.state = { name: "" };
    }

    render() {
      return <Name onChange={newName => this.setState({ name: newName })} />;
    }
  }
```

# CONTROLLED INPUT

## What does it mean?

In HTML, form elements such as ⟨input⟩, and typically maintain their own state and update it based on user input. In React, mutable state is typically kept in the state property of components, and only updated with setState().

The React component that renders a form also controls what happens in that form on subsequent user input. An input form element whose value is controlled by React in this way is called a "controlled input".

## Why it's important?

It only updates the DOM when state has changed. This is very important when creating large UIs.

In React has two different approaches to dealing with form inputs.

An input form element whose value is controlled by React is called a controlled component. When a user enters data into a controlled component a change event handler is triggered and your code decides whether the input is valid (by re-rendering with the updated value). If you do not re-render then the form element will remain unchanged.

An uncontrolled component works like form elements do outside of React. When a user inputs data into a form field (an input box, dropdown, etc) the updated information is reflected without React needing to do anything. However, this also means that you can't force the field to have a certain value.

```
<input type="text" />
```

```
<input type="text" value="This won't change. Try it." />
```

```
class ControlledNameInput extends React.Component {
    constructor() {
      super()
      this.state = {name: ""}
    }

    render() {
      return <input type="text" value={this.state.name} />
    }
}
```

softserve

This is a controlled input. It only updates the DOM when state has changed in our component. This is invaluable when creating consistent UIs.

```
return (
    <input
      value={this.state.name}
      onChange={e => this.setState({ name: e.target.value })}
    />
  )
```

# RESOURCES USED

- https://reactjs.org/docs/higher-order-components.html
- https://medium.com/quick-bytes/hoisting-state-in-react-and-why-7dc5c832e35d
- https://medium.com/@leonardobrunolima/react-tips-state-hoisting-c77c3cc78719