# HOC

JURUSAN TEKNOLOGI INFORMASI
POLITEKNIK NEGERI MALANG

Higher-Order Components (HOCs) are an interesting technique in React used to refactor similar components that share almost the same logic. I know that it sounds abstract and advanced. However, it is an architectural pattern that isn't specific to React, and hence you can use the approach to do lots of things.

**Higher-Order Functions**

What is a higher-order function? Wikipedia has a straightforward definition:

*In mathematics and computer science, a higher-order function (also functional, functional form or functor) is a function that either takes one or more functions as arguments or returns a function as its result or both.*

## Higher-Order Components

A higher-order component is a function that accepts a component as an argument and returns an extended version of that component.

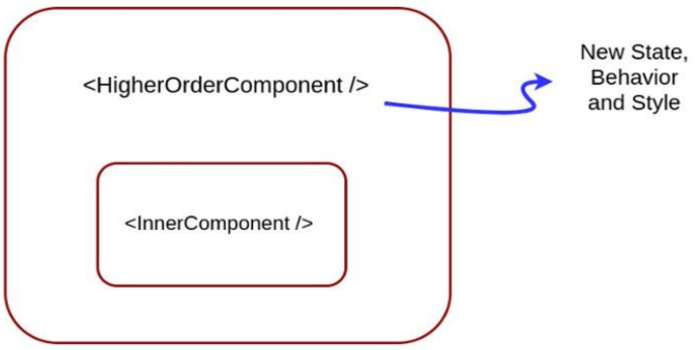
```
(InputComponent) => {
    return ExtendedComponent
    }

// or alternatively

InputComponent => ExtendedComponent
```

The `ExtendedComponent` *composes* the `InputComponent`. The `ExtendedComponent` is like a container. It renders the `InputComponent`, but because we're returning a new component, it adds an extra layer of abstraction. You can use this layer to add state, behavior, or even style. You can even decide not to render the `InputComponent` at all if you desire—HOCs are capable of doing that and more.

The image below should clear the air of confusion if any.



Enough with the theory—let's get to the code. Here is an example of a very simple HOC that wraps the input component around a `<div>` tag. From here on, I will be referring to the

`InputComponent` as `WrappedComponent` because that's the convention. However, you can call it anything you want.

HOCs are popular techniques for building reusable components. We started with a discussion of basic ES6 syntax so that it would be easier for you to get used to arrow functions and write modern-day JavaScript code.

**Practical Higher-Order Components**

Since HOCs create a new abstract container component, here is the list of things that you can normally do with them:

- Wrap an element or component around a component.
- State abstraction.
- Manipulate props, e.g. adding new props and modifying or removing existing props.
- Props validation to create.
- Use refs to access instance methods.

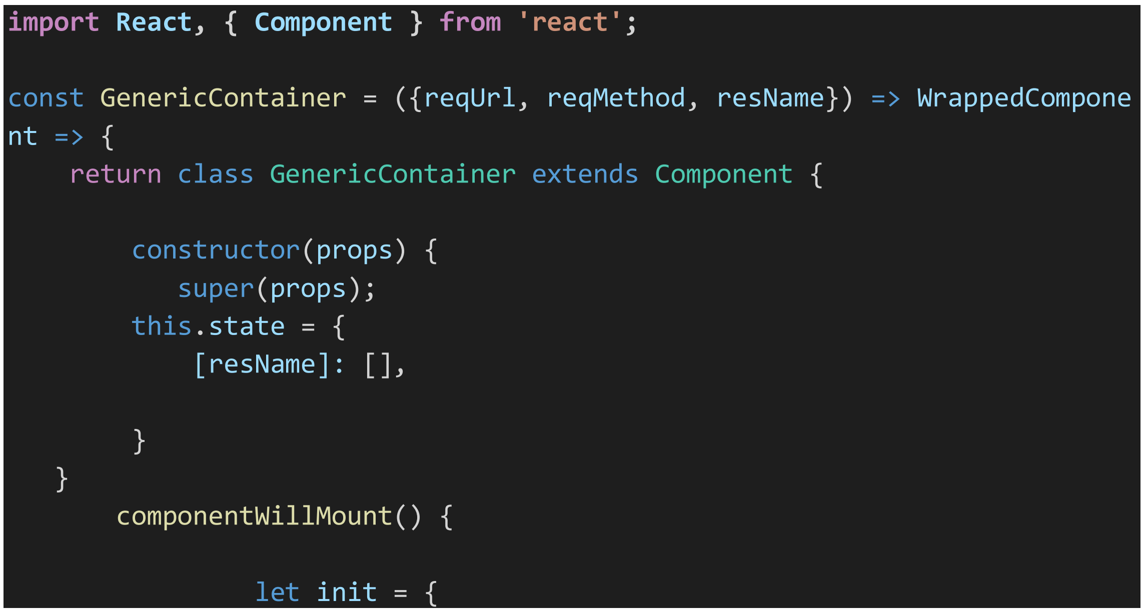
**Praktikum :**

To install the dependencies and run the project, just run the following commands from the project folder.

```
npm install
npm install --save react-router-dom
npm start
```

A. A Higher-Order Generic Container

1. If you noticed, the loader example above had a component that made a GET request using the fetch API. After retrieving the data, it was stored in the state. Making an API request when a component mounts is a common scenario, and we could make a HOC that perfectly fits into this role.
2. GenericContainer/GenericContainerHOC.jsx



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

const GenericContainer = ({reqUrl, reqMethod, resName}) => WrappedCompone
nt => {
    return class GenericContainer extends Component {

        constructor(props) {
           super(props);
        this.state = {
            [resName]: [],

        }
    }
       componentWillMount() {

               let init = {
```

```
                        method: reqMethod,
                        headers: new Headers(),
                        mode: 'cors',
                        cache: 'default'
                    };


                fetch(reqUrl, init)
                    .then( (response) => (response.json()))
                    .then(
                        (data) =>  {this.setState(
                    prevState => ({
                        [resName]: [...data.contacts]
                    })
                )}
                    ).then(console.log(this.state))



            }

        render() {
            return(
                <WrappedComponent {...this.props} {...this.state} />)
        }

    }
}

export default GenericContainer;
```

3. GenericContainer/GenericContainerDemo.jsx

/* A presentational component */

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

import GenericContainer from './GenericContainerHOC.jsx';
class GenericContainerDemo extends Component {
  render() {
    const ContactListWithGenericContainer = GenericContainer({reqUrl: 'http
s://demo1443058.mockable.io/users/', reqMethod:'GET', resName:'contacts'})(
ContactList);
    return(<div className="contactApp">

        <ContactListWithGenericContainer  />
         </div>
        )
  }
}
const ContactList = ({contacts}) => {
  return(
    <div>

    <ul>
      {contacts.map(
        (contact) => <li key={contact.email}>
                   <img src={contact.photo} width="100px" height="100px"  a
lt="presentation" />
          <div className="contactData">
          <h4>{contact.name}</h4>
           <small>{contact.email}</small>  <br/><small> {contact.phone}</sm
all>
          </div>

        </li>
      )}
    </ul>
    </div>
    )
}
export default GenericContainerDemo;
```

B. A Higher-Order Form
Here is another example that uses the state abstraction to create a useful higher-order form component.

CustomForm/CustomFormDemo`.jsx`

```
      import React from 'react';

import CustomForm from './CustomFormHOC.jsx';

const Form = (props) => {

    const handleSubmit = (e) => {
        e.preventDefault();
        props.onSubmit();
    }

    const handleChange = (e) => {
        const inputName = e.target.name;
        const inputValue = e.target.value;

        props.onChange(inputName,inputValue);
    }

    return(
        <div>
          <form onSubmit  = {handleSubmit} onChange= {handleChange} >
            <input name = "name" type= "text" /><br />
            <input name ="email" type="text"/><br/>
            <button type="submit"> Submit </button>
          </form>
        </div>

        )
}

const CustomFormDemo = (props) => {
    const FormWithCustom = CustomForm({ contact: {name: '', email: ''}})({prop
Name:'contact', propListName: 'contactList'})(Form);
    return(
        <div>
            <FormWithCustom {...props} />
        </div>
        );
}

export default CustomFormDemo;
```

CustomForm/CustomFormHOC.jsx

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


const CustomForm = (propState) => ({propName, propListName}) => WrappedCompone
nt => {
    return class CustomForm extends Component {


    constructor(props) {
        super(props);
        propState[propListName] = [];
        this.state = propState;

        this.handleSubmit = this.handleSubmit.bind(this);
        this.handleChange = this.handleChange.bind(this);
    }

    handleSubmit() {
      this.setState( prevState => {
        return ({
        [propListName]: [...prevState[propListName], this.state[propName] ]
      })}, () => console.log(this.state[propListName]) )}
```

```
  handleChange(name, value) {

      this.setState( prevState => (
        {[propName]: {...prevState[propName], [name]:value} }) )
      }




        render() {
            return(
                <div>
                    <WrappedComponent {...this.props} {...this.state} onChange
 = {this.handleChange} onSubmit = {this.handleSubmit} />
                    The values are { JSON.stringify(this.state[propListName],
null,2)}
                </div>

                )
        }
    }
}

export default CustomForm;
```

C. A Loading Indicator HOC

The first example is a loading indicator built using HOC. It checks whether a particular prop is empty, and the loading indicator is displayed until the data is fetched and returned.

LoadDemo/LoaderHOC.jsx

```
import React, { Component } from 'react';
import './LoadIndicator.css';

const isEmpty = (prop) => (
  prop === null ||
  prop === undefined ||
  (prop.hasOwnProperty('length') && prop.length === 0) ||
  (prop.constructor === Object && Object.keys(prop).length === 0)
);

const LoadIndicator = (loadingProp) => (WrappedComponent) => {
  return class LoadIndicator extends Component {

    render() {
      return isEmpty(this.props[loadingProp]) ? <div className="loader" /> : <
WrappedComponent {...this.props}/>;
    }
  }
}


export default LoadIndicator;
```

LoadDemo/LoaderDemo.jsx

```
import React, { Component } from 'react';
import LoadIndicator from './LoaderHOC.jsx';
import './ContactsApp.css';


class LoaderDemo extends Component {

    constructor(props) {
        super(props);
        this.state = {
            searchText : '',
            searchResults: [],
            contactList: []
        }
        this.handleUserInput = this.handleUserInput.bind(this);

    }

    handleUserInput(inputText) {
        this.setState({
            searchText: inputText,
        }, () => console.log(this.state))
    }

    componentWillMount() {
        let init = {
               method: 'GET',
               headers: new Headers(),
               mode: 'cors',
               cache: 'default'
           };


        fetch('https://demo1443058.mockable.io/users/', init)
            .then( (response) => (response.json()))
            .then(
                (data) => {console.log(data);this.setState(
                    prevState => ({
                        contactList: [...data.contacts]
                    })
                )}
        )
    }


    render() {

        const ContactListWithLoadIndicator = LoadIndicator('contacts')(ContactList);
        return(<div className="contactApp">

                <ContactListWithLoadIndicator contacts = {this.state.contactList} />
               </div>
          )
    }
}


const ContactList = ({contacts}) => {

    return(
        <div>

    <ul>
      {contacts.map(
        (contact) => <li key={contact.email}>

          <img src={contact.photo} width="100px" height="100px" alt="presentation" />
          <div className="contactData">
          <h4>{contact.name}</h4>
           <small>{contact.email}</small>  <br/><small> {contact.phone}</small>
>
          </div>
          {console.log(contact)}
        </li>
      )}
    </ul>
        </div>
        )
}


export default LoaderDemo;
```

LoadDemo/LoaderIndicator.css

```css
.loader {
  position: absolute;
  top: 50%;
  left: 50%;
  -webkit-transform: translateX(-50%) translateY(-50%);
  -ms-transform: translateX(-50%) translateY(-50%);
  transform: translateX(-50%) translateY(-50%);
}

/* Static Shape */

.loader:before {
  position: absolute;
  content: '';
  top: 0%;
  left: 50%;
  width: 100%;
  height: 100%;
  border-radius: 500rem;
  border: 0.2em solid rgba(0, 0, 0, 0.1);
}

/* Active Shape */

.loader:after {
  position: absolute;
  content: '';
  top: 0%;
  left: 50%;
  width: 100%;
  height: 100%;
  animation: loader 0.6s linear;
  animation-iteration-count: infinite;
  border-radius: 500rem;
  border-color: #767676 transparent transparent;
  border-style: solid;
  border-width: 0.2em;
  box-shadow: 0px 0px 0px 1px transparent;
}

/* Active Animation */

@-webkit-keyframes loader {
  from {
    -webkit-transform: rotate(0deg);
    transform: rotate(0deg);
  }
  to {
    -webkit-transform: rotate(360deg);
    transform: rotate(360deg);
  }
}

@keyframes loader {
  from {
    -webkit-transform: rotate(0deg);
    transform: rotate(0deg);
  }

  to {
    -webkit-transform: rotate(360deg);
    transform: rotate(360deg);
  }
}

.loader:before,
.loader:after {
  width: 2.28571429rem;
  height: 2.28571429rem;
  margin: 0em;
}
```

D. A Higher-Order Component for Protecting Routes

Here is an example of protecting routes by wrapping the relevant component with a withAuth higher-order component.

ProtectedRoutes/RequireAuthHOC.jsx

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


const RequireAuth = ComposedComponent =>  {
    return class Authentication extends Component {


        componentWillMount() {
            console.log(this.props);
            if(!this.props.authenticated) {
                this.props.history.push('/login');
            }
        }

        render() {

            return(
                <div>
                    <ComposedComponent {...this.props} />
                </div>
            )
        }
    }
}

export default RequireAuth;
```

ProtectedRoutes/RequireAuthDemo.jsx

```
import React, { Component } from 'react';
import {} from "react-router-dom";
import { Link, Route, Switch, withRouter } from 'react-router-dom';
import RequireAuth from './RequireAuthHOC.jsx';


class RequireAuthDemo extends Component {

  constructor(props) {
    super(props);
    /* Initialize state to false */
    this.state = {
      authenticated : false,
    }
  }
  render() {
    const AuthContacts = withRouter(RequireAuth(Contacts));
    const {match} = this.props;
    console.log(match);
    return (

      <div>

          <ul className="nav navbar-nav">
            <li><Link to={`${match.url}/home/`}>Home</Link></li>
             <li><Link to={`${match.url}/contacts` }>Contacts(Protected Route)
</Link></li>
          </ul>


        <Switch>
          <Route exact path={`${match.path}/home/`} component={Home}/>
          <Route path={`${match.path}/contacts`} render = { () =>  <AuthContac
ts authenticated = {this.state.authenticated} {...this.props} /> }/>
        </Switch>

      </div>
    );
  }
}

const Home = () => {
    return(<div> Navigating to the protected route gets redirected to /login <
/div>);
}

const Contacts = () => {
    return(<div> Contacts </div>);

}


export default RequireAuthDemo;
```

withAuth checks if the user is authenticated, and if not, redirects the user to /login. We've used withRouter, which is a react-router entity. Interestingly, withRouter is also a higher-order component that is used to pass the updated match, location, and history props to the wrapped component every time it renders.

RefsDemo/RefsHOC.jsx

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

  const RefsHOC = WrappedComponent => {
    return class Refs extends Component {

      constructor(props) {
        super(props);
        this.state =  {
            value: ''
        }
        this.setStateFromInstance = this.setStateFromInstance.bind(this);
      }

    setStateFromInstance() {
            this.setState({
                value: this.instance.getCurrentState()
          })

     }

      render() {
        return(
            <div>
            <WrappedComponent {...this.props} ref= { (instance) => this.instan
ce = instance } />

            <button onClick = {this.setStateFromInstance }> Submit </button>

            <h3> The value is {this.state.value} </h3>

            </div>
        );
      }
    }
}

export default RefsHOC;
```

RefsDemo/RefsDemo.jsx

```
import React, { Component } from 'react';
import RefsHOC from './RefsHOC.jsx';



class RefsDemo extends Component {

    render() {

        const RefsComponent = RefsHOC(SampleComponent);
        return(<div className="contactApp">

                <RefsComponent />
               </div>
          )
    }
}


class SampleComponent extends Component {

    constructor(props) {
        super(props);
        this.state = {
            value: ''
        }
        this.handleChange = this.handleChange.bind(this);

    }
    getCurrentState() {
        console.log(this.state.value)
        return this.state.value;
    }

    handleChange(e) {
        this.setState({
            value: e.target.value
        })

    }
    render() {
        return(
            <input type="text" onChange= {this.handleChange} />
        )
    }
}

export default RefsDemo;
```