8

JS

Higher Order Components

Aravind G

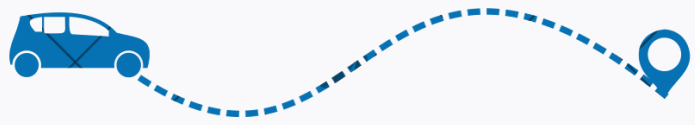
Simple Analogy

Imagine you have a basic car that gets you from point A to point B.It works fine on its own.

Now, you bring this car to a customization shop (the HOC function).

The shop doesn't change your car's engine or core functionality, but it adds enhancements:

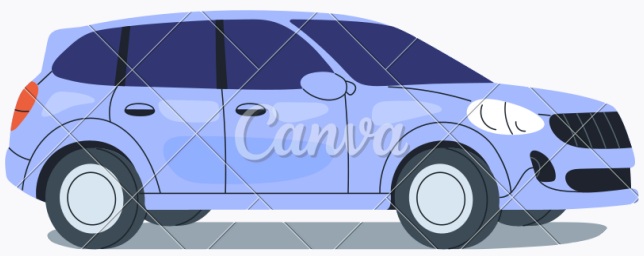
Maybe it adds a better **navigation system**
Or installs advanced **security features**
Or applies a premium **paint job**



Simple Analogy

When the shop is done, you drive away with the same car at its core, but now it has new capabilities that weren't built into it originally.

The important part: the customization shop doesn't rebuild your car. Instead, it takes your complete car, adds features around it, and returns an enhanced version while leaving the original design intact.



Higher Order Component

A **Higher Order Component** is a pattern in React where a function takes a component and returns a **new enhanced component**.

It's essentially a function that accepts a component as an argument and returns a new component with additional props, state, or behavior.

HOC.jsx

```jsx
function withExtraProps(WrappedComponent) {
  return function(props) {
    const extraProps = { extraData: 'Some extra data' };
    return <WrappedComponent {...props} {...extraProps} />;
  };
}

// Usage
const EnhancedComponent = withExtraProps(SimpleComponent);
```

Practical Use Cases

Authentication: Protect routes by checking if a user is logged in

HOC.jsx

```jsx
function withAuth(Component) {
  return function(props) {
    if (!isAuthenticated()) {
      return <Navigate to="/login" />;
    }
    return <Component {...props} />;
  };
}

const ProtectedDashboard = withAuth(Dashboard);
```

Practical Use Cases

Loading States: Add loading indicators to components that fetch data

HOC.jsx

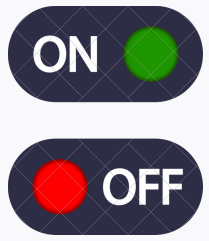
```jsx
function withLoader(Component) {
  return function(props) {
    const [loading, setLoading] = useState(true);
    const [data, setData] = useState(null);

    useEffect(() => {
      fetchData()
        .then(result => {
          setData(result);
          setLoading(false);
        });
    }, []);

    if (loading) return <LoadingSpinner />;
    return <Component {...props} data={data} />;
  };
}

const ProductListWithLoader = withLoader(ProductList);
```

Practical Use Cases



Feature Toggling: Show or hide components based on feature flags

HOC.jsx

```jsx
function withFeatureFlag(Component, featureName) {
  return function(props) {
    const features = useFeatures();
    if (!features[featureName]) {
      return null;
    }
    return <Component {...props} />;
  };
}

const NewFeature = withFeatureFlag(BetaComponent, 'enableBetaFeatures');
```

Practical Use Cases

Error Boundaries: Wrap components with error handling

HOC.jsx

```jsx
function withErrorHandling(Component) {
  return class ErrorBoundary extends React.Component {
    state = { hasError: false };

    static getDerivedStateFromError() {
      return { hasError: true };
    }

    render() {
      if (this.state.hasError) {
        return <ErrorMessage />;
      }
      return <Component {...this.props} />;
    }
  };
}

const SafeComponent = withErrorHandling(RiskyComponent);
```

Save Post

https://github.com/aravindFrontEnd

Aravind G