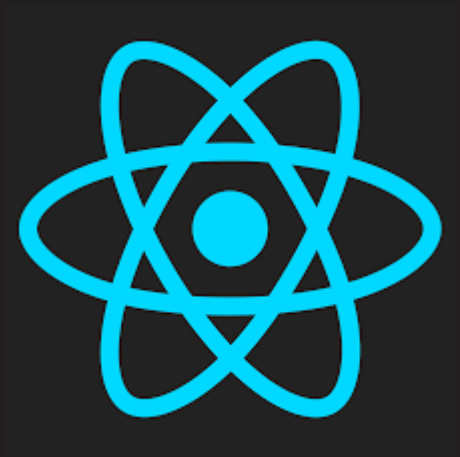
Render Props & Higher Order Components

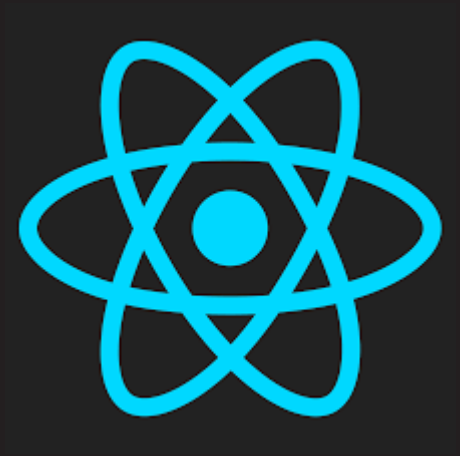


LEARNING OBJECTIVES

- Learn to implement shared logic using render props
- Learn to reuse component logic using the HOC pattern

Render Props & Higher Order Components

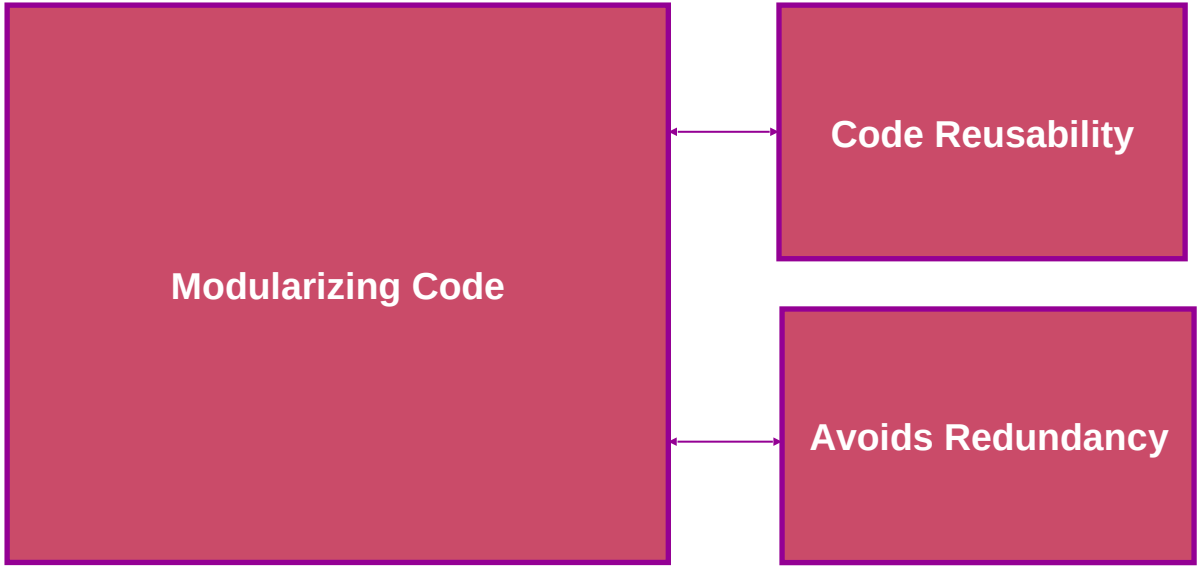
Render Props



What are Render Props?

Render Props are patterns for building reusable components that allows sharing of logic and state between components.

Modularizing Code



React:

- Is not opinionated
- Promotes time-tested software design patterns and techniques for efficient coding practices
- React apps and components are built using reusable code

Render Props

Can be used to write reusable & shared component logic

```
<MagicBox render={DATA => <Component
        DATA={DATA} />} />
```

Writing reusable & shared logic using Render Props

Create a component implementing render prop

```
<MagicBox render={DATA => <Component DATA={DATA} />} />
```

The render prop must implement a function, returning a React element

Writing reusable & shared logic using Render Props

The consumer is free to implement render logic

```
<MagicBox render={DATA => <Component DATA={DATA} />} />
```

The brain is encapsulated in a reusable component

Render Props: Example

Consumer provides this value

```
<FetchRates
        symbol={currencySymbol}
        render={rate => <DisplayRate value={rate} />}
/>
```

The fetched rate is provided to the render prop function as an argument

Our custom render logic consumes the rates as needed.

Render Props: Example

The database & operations are stored in the Storage component and can be reused.

```
< Storage
        client={(fetch, SAVE, UPDATE) =>
                <SomeApp onFetch={fetch} onSave={save}
onUpdate={update} />}
/>
```

<SomeApp /> uses the Storage through a render prop named client

Render Props: Hands On

Component Logic

```
<Weather
        location={location}
        render={
         ({error, isLoading, icon, place, temperature, conditions}) =>
          !error ? (
            isLoading ? (
              <div className="loading">Please wait...</div>
            ) : (
              <div className="result">
                <div className="place">{place}</div>
                <div className="temperature">{temperature}&deg;C</div>
                <div className="conditions">{conditions.join(",")}</div>
                <img src={icon} alt="Sunny" className="icon" />
              </div>
            )
          ) : (
            <div className="error">
              There was an error fetching the weather!
            </div>
```

Render Logic

Composing together components using render props

```
<SharedComponent
        render={(dataProps) =>
                <Enhance {...dataProps} render={
                        (resultProps) => <RenderComponent {...
resultProps} />
                        }
                }
/>
```

Compose it with other function to enhance/add abilities

Render Props

Render props makes it possible to declaratively compose together logic!

Use of ‘children’ prop instead to implement Render Prop:

- Provides React components with access to components that are enclosed within a pair of opening and closing component instances

```
const WEATHER = ({LOCATION, children}) => {
  const [ISLOADING, SETISLOADING] = USESTATE(true);
  const [error, setError] = USESTATE(FALSE);
  const [TEMPERATURE, SETTEMPERATURE] = USESTATE(0);
  const [conditions, setConditions] = USESTATE([]);
  const [icon, setIcon] = USESTATE("");
  const [PLACE, SETPLACE] = USESTATE("");
  const FETCHWEATHER = LOCATION => {…};


  useEffect(() => {
    if (LOCATION) {
      FETCHWEATHER(LOCATION);
    }
  }, [LOCATION]);
```

Use of ‘children’ prop instead to implement Render Prop:

```
<Weather location={location}>
  {(temperature, place, conditions, icon) => {
    // Render logic
  }}
</Weather>
```

Render prop pattern implemented using the children prop

Render Props pattern is used by various popular community packages

- React / Reach Router
- Downshift
- many more…

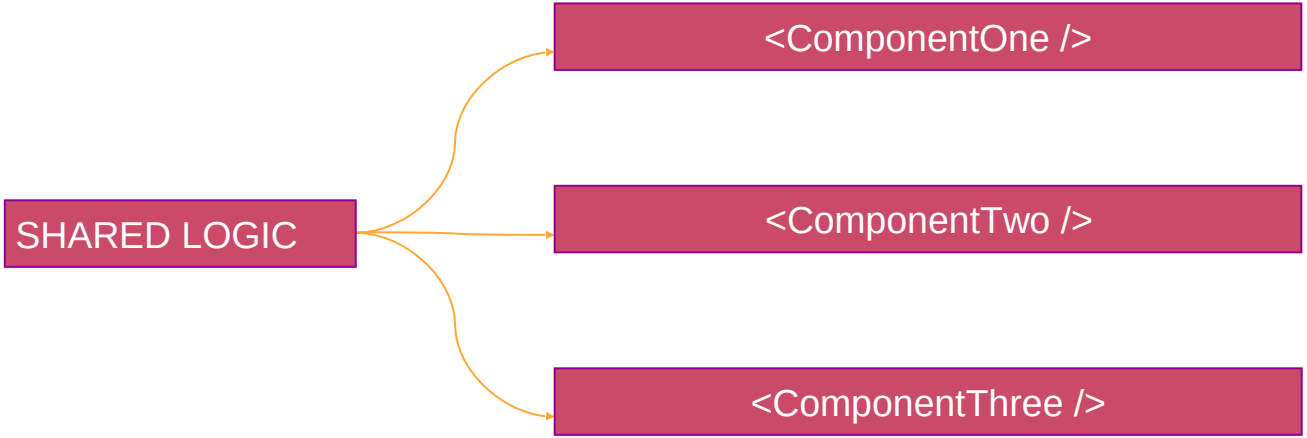
Render Props & Higher Order Components

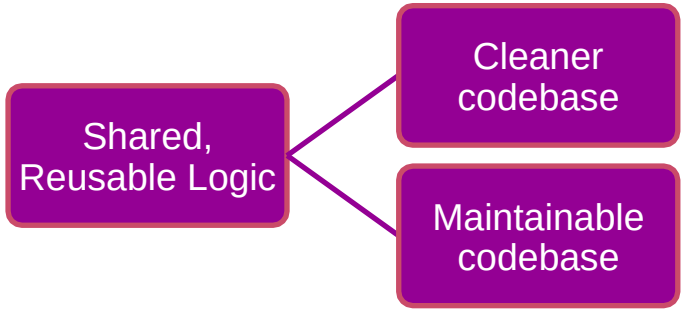
Higher Order Components

What are Higher Order Components?

Higher Order Components are patterns for abstracting & reusing component logic.

Higher Order Component





Higher Order Component

- Functions are first class objects in JavaScript
- They can be passed as arguments & returned from a function

```
const result = myArray.map(i => i * 2);
```

Higher Order Function

Accept a function as an argument

Return a function that invokes the received function

```
const hof = (x, fn) => (...ARGS) =>
            fn(x, ...ARGS);
```

Higher Order Function

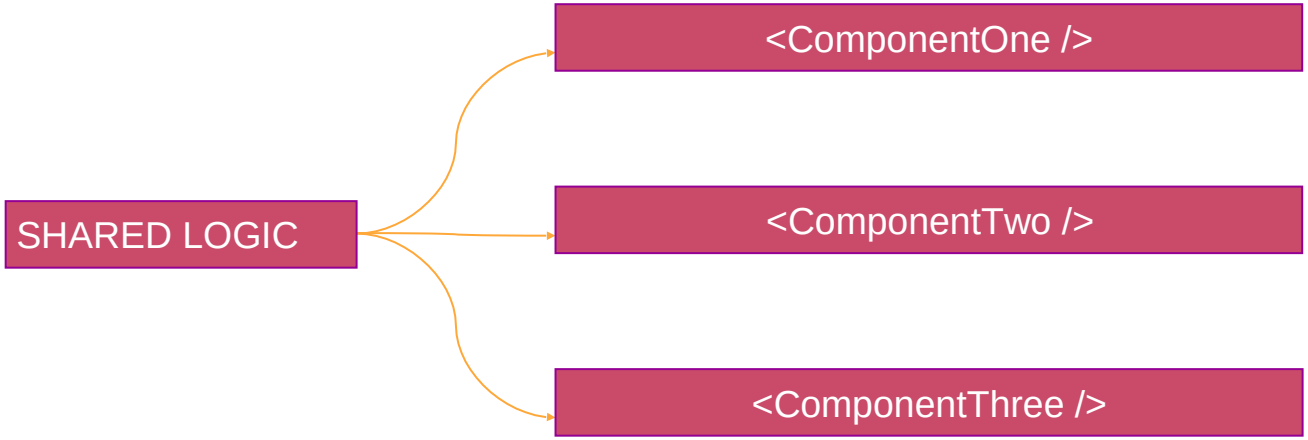
Accept a function as an argument

Return a function that invokes the received function

```
const hof = (x, fn) => (...ARGS) =>
            fn(x, ...ARGS);
```

Abstracts the inner workings and augmentations of the original function

Higher Order Component



Higher Order Component

To add extra, shared features

props

```
const HOC = Componen                                    =>
            <Componen {...props} />
```

Higher Order Component: Hands-on example

Higher Order Function

```
render() {
  const hoc = WITHPOWERS( RIGINAL);
  return (


  );
}
```

NEVER DO THIS

- Affects performance
- Destroys internal state in the Original component on every re-render

Higher Order Component

Render props may be a better pattern for cases with code reusability

Higher Order Component

HOOKS API

Even better??

Higher Order Component

Higher Order Components ae used extensively

- React / Reach Router
- Redux
- and many more

To sum it up...

- Render Props are patterns for building reusable components that allows sharing of logic and state between components.

- Higher Order Components is yet another pattern for abstracting & reusing component logic.

thank you!