Components

1. Overview
2. Using inheritance
3. Functional stateless comps
4. Using factories

1. Overview

- The story so far…
- A more modular approach…
- Defining components in React

The Story So Far…

- The examples so far have created a monolithic dollop of React elements in one giant block of code

```
let prodList = …
let shopList = …

let retailer = React.createElement('div', null,
  React.createElement('h1', null, 'Catalog'),
  prodList, …, shopList, …
)

ReactDOM.render(retailer, …)
```

A More Modular Approach…

- Divide-and-conquer
  - Partition the UI into a bunch of components
  - Each component is responsible for one part of the UI
- Benefits of the component approach
  - Each component is relatively small and focussed
  - Easier to develop
  - Potential reuse
  - Easier to test

How to Define Components in React

- There are several ways to develop components in React
  - Via ES6 inheritance - see Section 2
  - Via stateless functional components - see Section 3
  - Via factories - see Section 4

- In earlier versions or React, you could also create a component using `React.createClass()`
  - But this is deprecated nowadays

2. Using Inheritance

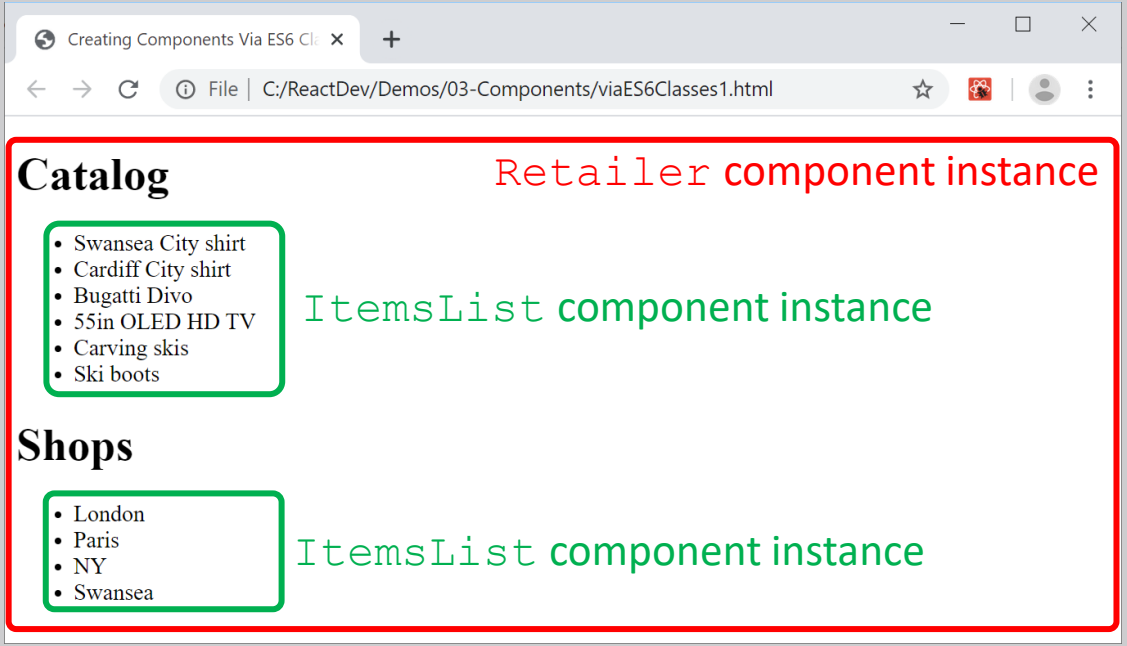
- Overview
- Example scenario
- Example data
- Component classes
- `ItemsList` component class
- `Retailer` component class
- Creating/rendering `Retailer`

Overview

- React has a class named `React.Component`
  - Has common capabilities needed by all components
  - E.g. render React elements for a component
- To define your own component class:
  - Define a class that inherits from `React.Component`
  - Override `render()` function, to render your elements
  - Access properties on the component, via `this.props`

Example Scenario

- `viaES6Classes1.html` has several components



Example Data

- Here's the familiar data for our components

```
const products = [
  'Swansea City shirt',
  'Cardiff City shirt',
  …
]

const shops = [
  'London',
  'Paris',
  …
]
```

Component Classes

- Here's how to define a component class:
  - Inherit from `React.Component`
  - Override `render()`, return React element(s) to render
  - Use `this.props.items` to access `items` property, passed into component by its parent component

ItemsList Component Class

```
class ItemsList extends React.Component {
  render() {
    return React.createElement(
      "ul", null,
      this.props.items.map((item, i) =>
                    React.createElement("li",
                    {key:i},
                    item))
    )
  }
}
```

Retailer Component Class (1 of 2)

- The Retailer component is similar

```
class Retailer extends React.Component {
  render() {
    // Create a <div> that contains:
    //   <h1>
    //   ItemsList component for products
    //   <h1>
    //   ItemsList component for shops
  }
}
```

Retailer Component Class (2 of 2)

- Retailer creates ItemsList for products like so:

```
React.createElement(ItemsList,
                    {items: products},
                    null)
```

- Retailer creates ItemsList for shops like so:

```
React.createElement(ItemsList,
                    {items: shops},
                    null)
```

Creating/Rendering `Retailer`

- We create/render a `Retailer` component as the root React element as follows:

```
const retailer =
  React.createElement(Retailer, null, null)

ReactDOM.render(
  retailer,
  document.getElementById('osl-container'))
```

- Also see `viaES6Classes2.html`
  - Encapsulates the creation of each `<li>`

3. Functional Stateless

- Overview
- How to do it
- `ItemsList` component class
- `Retailer` component class
- Creating/rendering `Retailer`

Overview

- Let's see another way to define components in React
  - As stateless functional components
  - This is the preferred way - cleaner than inheritance
  - See `viaStatelessFunctionalComps.html`

- A stateless functional component:
  - Is just a function (i.e. not a class)
  - Doesn't remember any data (it's stateless, i.e. no `this`)
  - Receives inputs via function parameters

How to do it

- To define a component as a stateless functional component:
  - Define an arrow function, a.k.a. lambda
  - Receive an object parameter (from parent component)
  - Destructure the object, to access its property
  - Create and return a React element - React automatically renders this element

ItemsList Component

- Here's `ItemsList` as a stateless functional comp
  - Receives an object with `items` property, from parent

```
const ItemsList = ({items}) =>
  React.createElement(
    "ul",
    null,
    items.map((item, i) =>
      React.createElement("li",
                          {key:i},
                          item))
  )
```

`Retailer` Component (1 of 2)

- Here's `Retailer` as a stateless functional comp
  - Receives an object with `products`, `shops` properties

```
const Retailer =({products,shops}) =>
  // Create a <div> that contains:
  //   <h1>
  //   ItemsList component for products
  //   <h1>
  //   ItemsList component for shops
```

`Retailer` Component (2 of 2)

- `Retailer` creates `ItemsList` for products and shops the same way as before

```
React.createElement(ItemsList,
                    {items: products},
                    null)
```

```
React.createElement(ItemsList,
                    {items: shops},
                    null)
```

Creating/Rendering `Retailer`

- We create/render a `Retailer` component like so
  - Note that we now pass an object into the `Retailer` component, with `products` and `shops` properties

```
const retailer = React.createElement(
  Retailer,
  {products,shops},
  null)

ReactDOM.render(
  retailer,
  document.getElementById('osl-container'))
```

Summary

- Overview
- Using inheritance
- Functional stateless comps
- Using factories