Beginning Frontend Development with React

Lesson 2

Creating Components

Lesson Objectives

By the end of this lesson, you will be able to:

- Create basic React components
- Use JSX to define a component's markup
- Combine multiple React components in order to create complex UI elements
- Manage the internal state of React components

Topic A: Definition of a Component

```
import React from 'react';

class Catalog extends React.Component {
  render() {
    return <div><h2>Catalog</h2></div>;
  }
}

export default Catalog;
```

Topic A: Definition of a Component

The *`render()`* method must comply with a few constraints

- It is mandatory; that is, any React component must implement it
- It must return one React element; that is, a single markup item with any nested elements
- It should be a pure function; that is, it should not change the internal state of the component
- It should not directly interact with the browser; that is, it shouldn't contain statements that try to access the DOM

Topic A: Definition of a Component

```
import React, { Component } from 'react';
import logo from './logo.svg';
import './App.css';

class App extends Component {
  render() {
    return (
      <div className="App">
        <header className="App-header">
          <img src={logo} className="App-logo" alt="logo" />
          <h1 className="App-title">Welcome to React</h1>
        </header>
        <p className="App-intro">
          To get started, edit <code>src/App.js</code> and save to reload.
        </p>
      </div>
    );
  }
}

export default App;
```

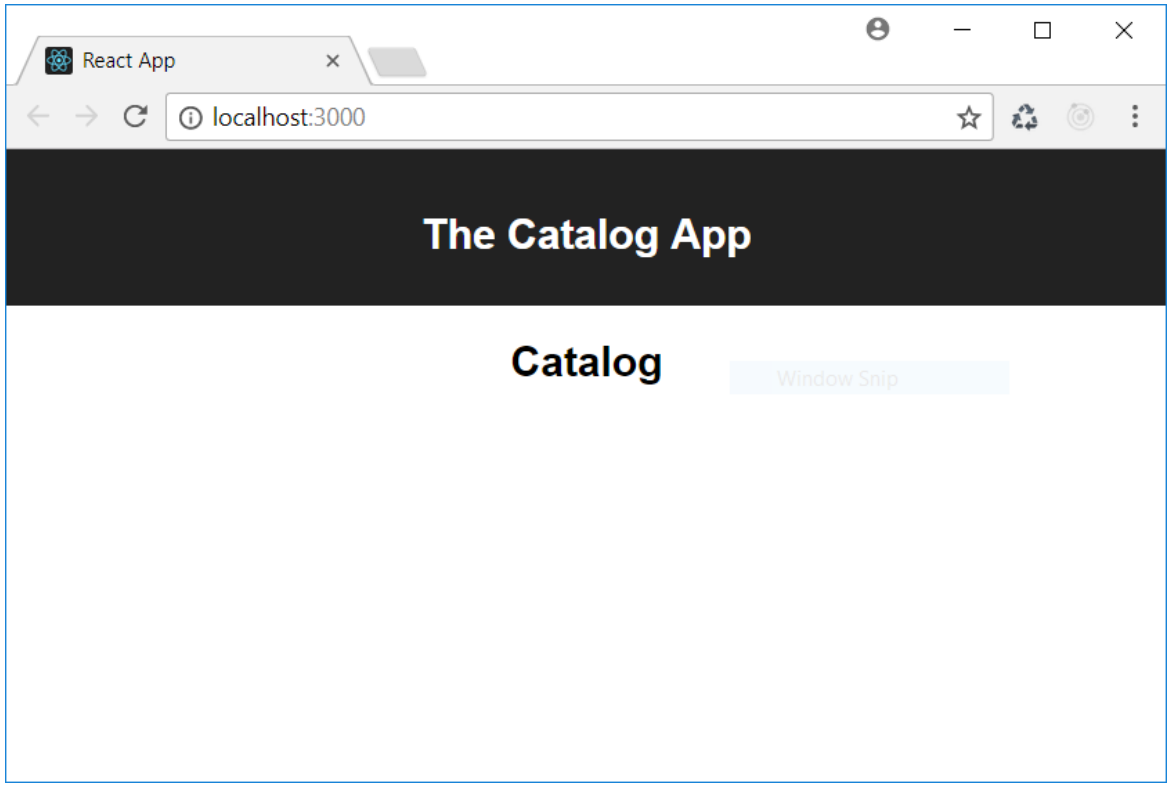
Topic A: Definition of a Component

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

class App extends Component {
  render() {
    return (
      <div className="App">
        <header className="App-header">
          <h1 className="App-title">The Catalog App</h1>
        </header>
        <Catalog />
      </div>
    );
  }
}

export default App;
```

Topic A: Definition of a Component



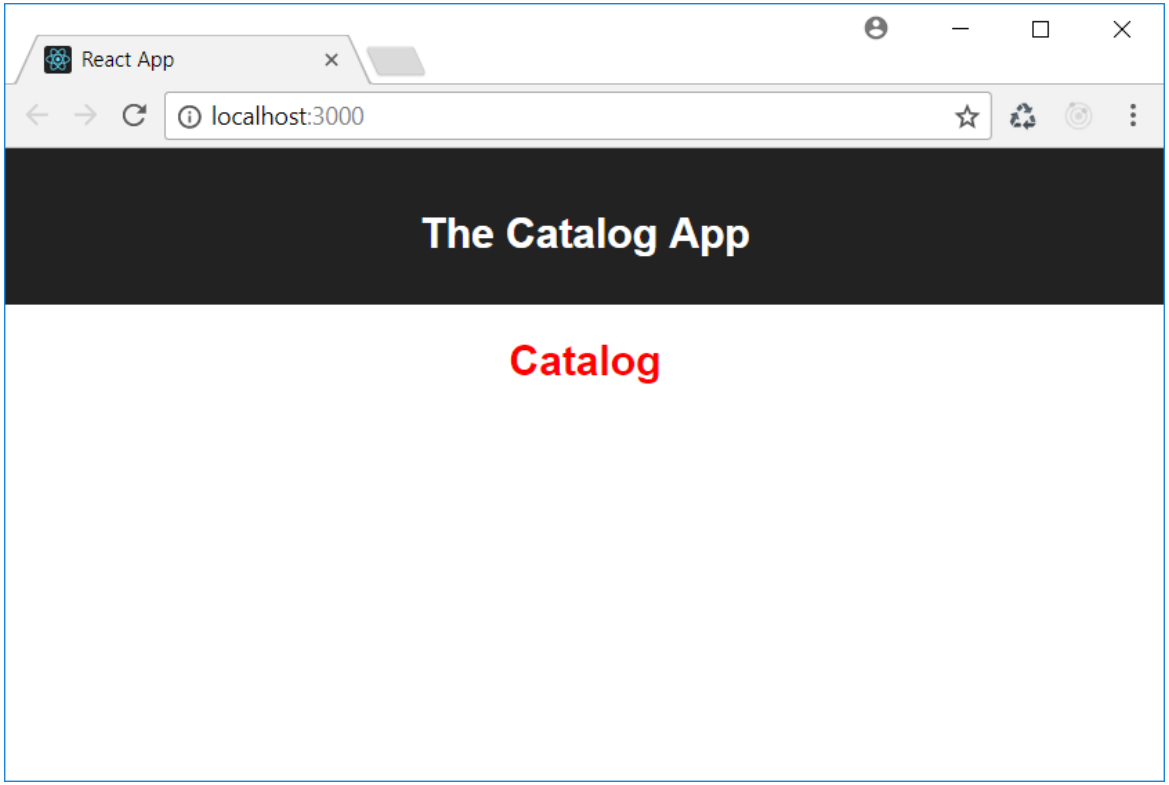
Managing Styles

We notice an **`import`** statement concerning a CSS file in the **`App`** component module.

This allows us to use inside our component the classes and other CSS definitions defined in **`App.css`**.

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

Managing Styles



Activity A: Defining a Shopping Cart

Aim

The aim of this activity is to start using React to define a component.

Scenario

We need a shopping cart for our e-shop.

Steps for Completion

1. We should define a React component that acts as the basis of a shopping cart.

2. It should be a component that simply shows the string **`Cart`**.

Topic B: Using JSX

```
import React from 'react';
import './Catalog.css';

class Catalog extends React.Component {
  render() {
    return <div><h2>Catalog</h2></div>;
  }
}

export default Catalog;
```

Topic B: Using JSX

JSX extends JavaScript with XML expressions in order to simplify the creation of HTML elements within JavaScript code:

```
//Valid

<div><h2>Catalog</h2></div>


//Not Valid

<div><h2>Catalog</h2></div>
<div><img src="image.png" /></div>
```

Topic B: Using JSX

JSX expressions must be compliant with XML syntax:

```
//Valid

<img src="image.png" />


//Not Valid

<img src="image.png">
```

Topic B: Using JSX

We can assign a JSX expression to a variable:

```
import React from 'react';
import './Catalog.css';

class Catalog extends React.Component {
  render() {
    let output = <div><h2>Catalog</h2></div>;

    return output;
  }
}

export default Catalog;
```

Topic B: Using JSX

We can embed any JavaScript expression inside a JSX expression:

```
import React from 'react';
import './Catalog.css';

class Catalog extends React.Component {
  render() {
    let title = "Catalog";

    return <div><h2>{title}</h2></div>;
  }
}

export default Catalog;
```

Topic B: Using JSX

The JavaScript expression can be as complex as we need:

```
import React from 'react';
import './Catalog.css';

class Catalog extends React.Component {
  render() {
    let title = "The Catalog of today " + new
Date().toDateString();

    return <div><h2>{title}</h2></div>;
  }
}
```

Topic B: Using JSX

A common usage of a combination of JavaScript and JSX expressions is called conditional rendering:

```
render() {
    let message;
    let today = new Date().getDay();

    if (today == 0) {
      message = <div className="sorry">We are closed on Sunday...</
div>;
    } else {
      message = <div className="happy">How can we help you?</div>
    }

    return message;
  }
```

Topic B: Using JSX

You can put a JSX expression in multiple lines:

```
import React from 'react';
import './Catalog.css';

class Catalog extends React.Component {
  render() {
    let title = "Catalog";

    return <div>
       <h2>{title}</h2>
           </div>;
  }
}
```

Topic B: Using JSX

You can put comments inside a JSX expression by using the JavaScript syntax wrapped in curly brackets:

```
<div>
      <h2>Catalog</h2>
      {//This is a comment}
      {/* This is a comment, too */}
</div>;
```

Topic B: Using JSX

JSX has a few restrictions:

- All HTML tags are in lowercase
- You need to use **`className`** instead of the **`class`** attribute
- You need to use **`htmlFor`** instead of the `for` attribute

Activity B: Translating HTML into JSX

Aim

The aim of this activity is to understand the difference between HTML and JSX.

Scenario

The *Graphics* department has provided you with an HTML snippet and you need to translate it into JSX in order to create a React component.

Steps for Completion

1. Open the **`Code02.txt`** file.
2. Transform the HTML code it contains into JSX.

Topic C: Composing Components

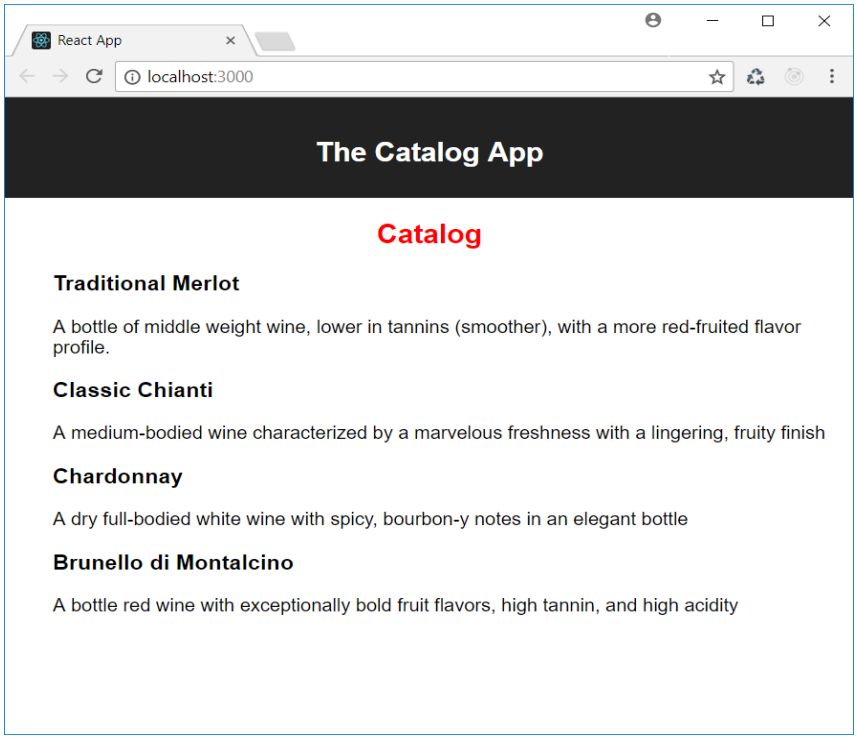
We can use the **ProductList** component as a tag in the JSX markup of the **Catalog** component:

```
import React from 'react';
import ProductList from './ProductList';

class Catalog extends React.Component {
  render() {
    return <div>
       <h2>Catalog</h2>
        <ProductList />
        </div>;
  }
}

export default Catalog;
```

Composing Components



Activity C: Defining a Composed Cart

Aim

The aim of this activity is to compose React components.

Scenario

We want to create some content for our shopping cart.

Step for Completion

Integrate the previously created **`Cart`** component in order to contain a **`CartList`** component showing two items.

Topic D: Data Propagation

Create a **`Product`** component that acts as a template for each list item:

```
import React from 'react';

class Product extends React.Component {
  render() {
    return <li>
             <h3>Product name</h3>
             <p>Product description</p>
           </li>;
  }
}

export default Product;
```

Topic D: Data Propagation

We can dynamically build our product list:

```
class ProductList extends React.Component {
  render() {
      let products = [{name: "Traditional Merlot", description:
"..."}, ...];
      let productComponents = [];

      for (let product of products) {
          productComponents.push(<Product/>);
      }

    return <ul>{productComponents}</ul>;
  }
}
```

Topic D: Data Propagation

We need a way to pass the data of each product to the component class: component attributes:

```
class ProductList extends React.Component {
  render() {
      let products = [{name: "Traditional Merlot", description:
"..."}, ...];
      let productComponents = [];

      for (let product of products) {
          productComponents.push(<Product item={product} />);
      }

    return <ul>{productComponents}</ul>;
  }
}
```

Topic D: Data Propagation

We need a way to receive and manage the passed data: component props:

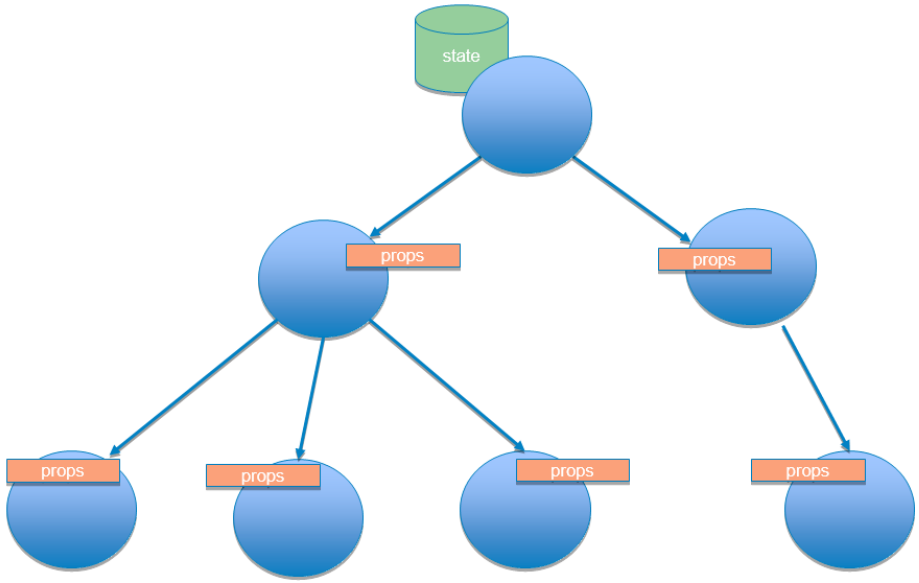
```
import React from 'react';

class Product extends React.Component {
  render() {
    return <li>
             <h3>{this.props.item.name}</h3>
             <p>{this.props.item.description}</p>
           </li>;
  }
}

export default Product;
```

Topic D: Data Propagation

Data passes from a component to another in a unidirectional data flow, from the parent component towards the children components:



Activity D: Creating a Cart Item Component

Aim

The aim of this activity is to compose React components and pass data between them.

Scenario

We want to make the **`CartList`** component a dynamic component, so that it can adapt its content to received data.

Steps for Completion

1. Create a **`CartItem`** component showing the name of an item.
2. Change the previously created **`CartList`** component so that it is dynamically composed of **`CartItem`** instances based on an `items` array.

Topic E: Managing the Internal State

- React **`state`** is a property that represents data that changes over time
- Components that store data that can change over time are called **stateful components**
- To inform a component that the state has changed, we use the **`setState()`** method

Topic E: Managing the Internal State

State initialization is the only case where you can assign a value to the `this.state` property without using **`setState()`**:

```
class ProductList extends React.Component {
  constructor() {
    super();
    this.state = { products: [] };
    fetch("products.json")
      .then(response => response.json())
      .then(json => {this.setState({products: json})})
      .catch(error => console.log(error));
  }
```

Topic E: Managing the Internal State

Changes to the state trigger the component's rendering, that is, the automatic execution of the **`render()`** method:

```
class ProductList extends React.Component {
...
  render() {
        let productComponents = [];

      for (let product of this.state.products) {
          productComponents.push(<Product item={product}/>);
      }

    return <ul>{productComponents}</ul>;
  }
}
```

Topic E: Managing the Internal State

- **`setState()`** merges new data with old data already contained in the state and overwrites the previous state
- **`setState()`** triggers the execution of the **`render()`** method, so you should never call **`render()`** explicitly

Topic E: Managing the Internal State

- State should contain the minimum data needed to represent data that can change over time in your UI; any information that can be derived from this minimal data should be computed inside the **`render()`** method
- State should be avoided as much as possible, since it adds complexity to a component
- Stateful components should be located high up in the component hierarchy of a UI

Topic E: Managing the Internal State

We move the logic of getting data inside the `Catalog` component:

```
class Catalog extends React.Component {
  constructor() {
    super();
    this.state = { products: [] };

    fetch("products.json")
      .then(response => response.json())
      .then(json => {this.setState({products: json})})
      .catch(error => console.log(error));
  }
  render() {
    return <div><h2>Wine Catalog</h2><ProductList
items={this.state.products}/></div>;
  }
}
```

Activity E: Adding State Management to the Cart Component

Aim

The aim of this activity is to become familiar with component state management.

Scenario

In order to make the **`Cart`** component production ready, we add state management and dynamic data loading.

Step for Completion

Change the previously created **`Cart`** component and add state management so that data is loaded via HTTP requests and the contents of the **`Cart`** are automatically updated.

Summary

In this lesson, we started to create React components and explored their basic features. In particular, we:

- Learned how to define a component as a class derived from **`React.Component`** and how to import specific CSS styles
- Explored the syntax of JSX, which allows us to quickly define the graphical aspect of a component and to use React components defined elsewhere
- Combined React components in order to build other components
- Used state management features so that React components automatically update their look when data changes