Beginning Frontend Development with React

Lesson 3

Managing User Interactivity

Lesson Objectives

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

- Handle events generated by user interaction
- Change a component's state on event triggering
- Use the component's lifecycle events for a better user experience
- Configure routing to allow navigation through components

Topic A: Managing User Interaction

```
[
  {"code":"P01",
    "name": "Traditional Merlot",
    "description": "A bottle of middle weight wine...",
    "price": 4.5},
  {"code":"P02",
    "name": "Classic Chianti",
    "description": "A medium-bodied wine...",
    "price": 5.3},
  {"code":"P03",
    "name": "Chardonnay",
    "description": "A dry full-bodied white wine...",
    "price": 4.0},
  {"code":"P04",
    "name": "Brunello di Montalcino",
    "description": "A bottle of red wine...",
    "price": 7.5}
]
```

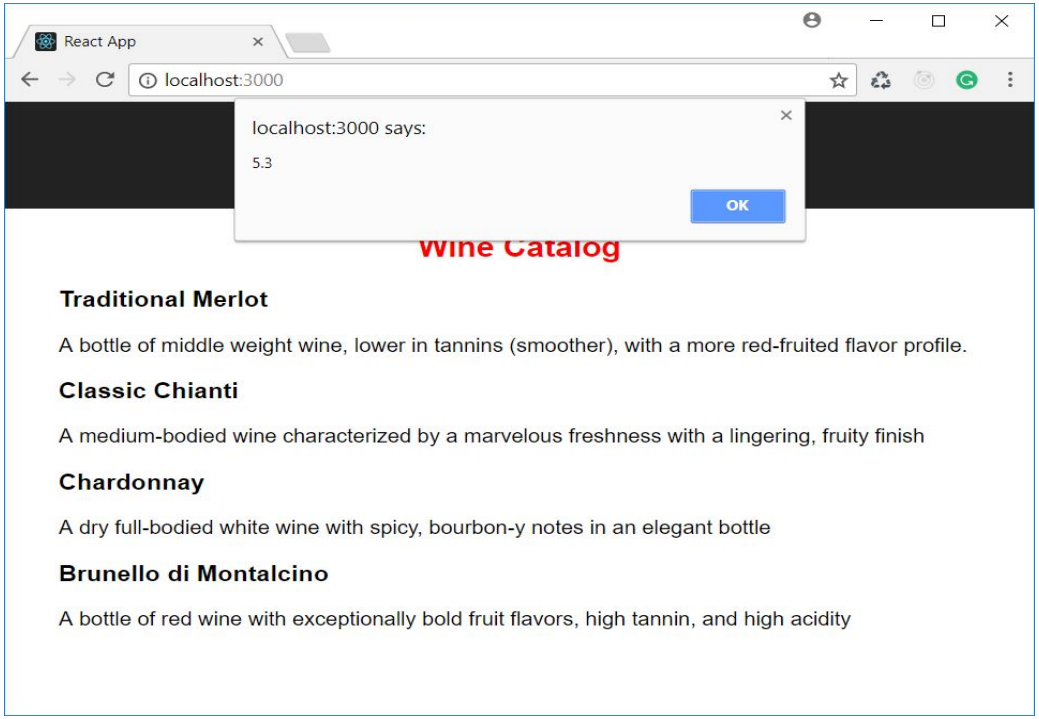
Topic A: Managing User Interaction

Showing price on click event

```
import React from 'react';

class Product extends React.Component {
  showPrice() {
    alert(this.props.item.price);
  }

  render() {
    return <li onClick={() => this.showPrice()}>
             <h3>{this.props.item.name}</h3>
             <p>{this.props.item.description}</p>
           </li>;
  }
}

export default Product;
```

Topic A: Managing User Interaction

Showing price on click event



HTML Events versus React Events

HTML	React
`<li onclick="...">...</li>`	`<li onClick=...>...</li>`
`<li onclick="showPrice()">...</li>`	`<li onClick={showPrice}>...</li>` `<li onClick={() => this.showPrice()}>`

`<a href="#" onclick={(e) =>`
`{ console.log("Clicked");`
`return false;}}>Click</a>`
	`<a href="#" onClick={(e) =>`
	`{ e.preventDefault();`
	`console.log("Clicked");}}>Click</a>`

Event Handlers and the this Keyword

```
<li onClick={() => this.showPrice()}>
<li onClick={this.showPrice.bind(this)}>
constructor() {
  this.showPrice = this.showPrice.bind(this);
}
...
<li onClick={this.showPrice}>
```

Changing the State

```
[
  {"code":"P01",
    "name": "Traditional Merlot",
    "description": "A bottle of middle weight wine...",
    "price": 4.5,
    "selected": false},
  {"code":"P02",
    "name": "Classic Chianti",
    "description": "A medium-bodied wine...",
    "price": 5.3,
    "selected": false},
  {"code":"P03",
    "name": "Chardonnay",
    "description": "A dry full-bodied white wine...",
    "price": 4.0,
    "selected": false},
  {"code":"P04",
    "name": "Brunello di Montalcino",
    "description": "A bottle of red wine...",
    "price": 7.5,
    "selected": false}
]
```

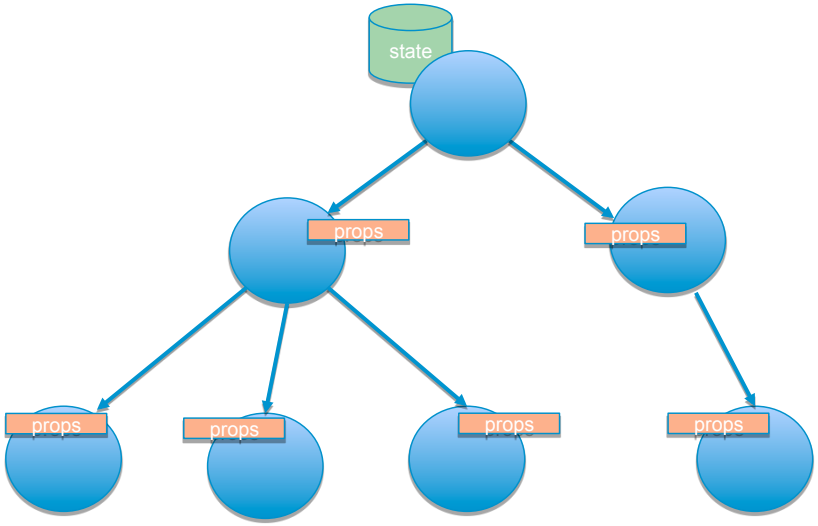
Changing the State

Change background color on click

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

class Product extends React.Component {
  select() { this.props.item.selected = !this.props.item.selected; }

  render() {
    let classToApply = this.props.item.selected? "selected": "";

    return <li onClick={() => this.select()} className={classToApply}>
            <h3>{this.props.item.name}</h3>
            <p>{this.props.item.description}</p>
          </li>;
  }
}

export default Product;
```

Changing the State

In a React component hierarchy, data flows in a unidirectional way



Changing the State

The parent component must provide methods to its children via **props** property

```
class Catalog extends React.Component {
  constructor() { ... }

  select(productCode) {
    let productList = this.state.products.map(function(p) {
                if (p.code === productCode) {
                        p.selected = (!p.selected);
                }
                return p;
        });
    this.setState({products: productList});
  }

  render() {
    return <div><h2>Wine Catalog</h2><ProductList items={this.state.products}
selectHandler={this.select}/></div>;
  }
}
```

Changing the State

The method is propagate to children

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

      for (let product of this.props.items) {
          products.push(<Product item={product}
selectHandler={this.props.selectHandler}/>);
      }

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

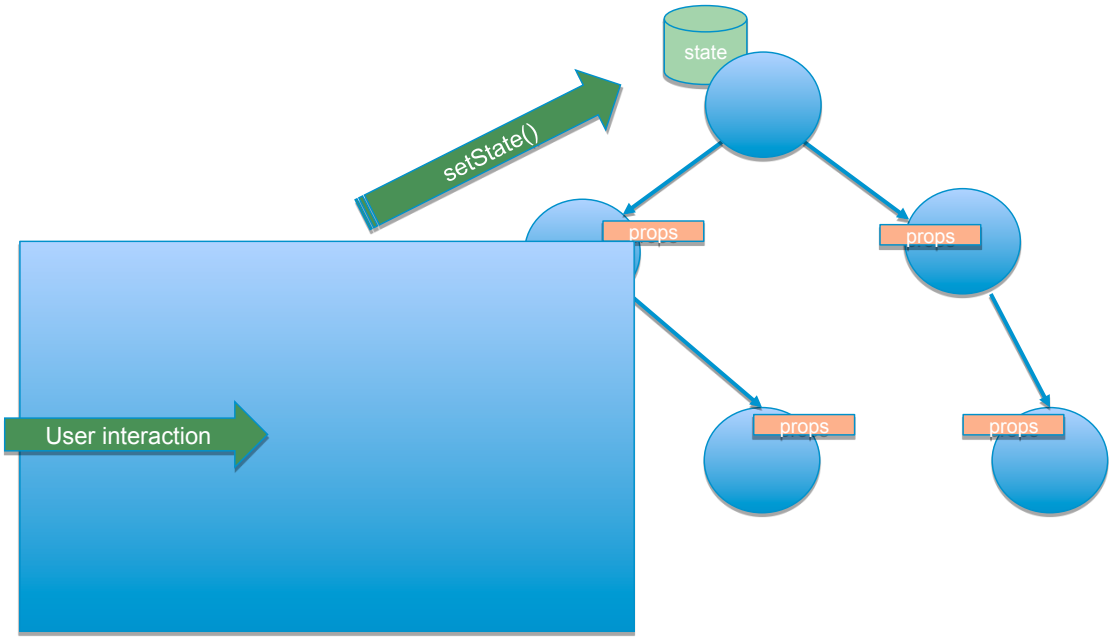
Changing the State

The target component handles the event by calling the passed method

```
class Product extends React.Component {
  render() {
    let classToApply = this.props.item.selected? "selected": "";

    return <li onClick={() => this.props.selectHandler (this.props.item.code)}
className={classToApply}>
            <h3>{this.props.item.name}</h3>
            <p>{this.props.item.description}</p>
          </li>;
  }
}
```

Changing the State

An event on a child component triggers the execution of a parent component method passed via props



Activity A: Adding Items to the Shopping Cart

Aim

The aim of this activity is to become familiar with event management in React.

Scenario

We want to allow the user to add items to the shopping cart by picking them from the product catalog.

Steps for Completion

1. Consider the existing project in **`my-cart-01`** folder.
2. Handle the click event of the **Add** to cart button of the **`Product`** component in order to add that product to the cart.

Topic B: Component Lifecycle Events

Component's constructor execution

- DOM is not available
- It is not possible to access any child component
- It is the right time to perform those initializations not involving graphic rendering or child manipulation

Topic B: Component Lifecycle Events

Lifecycle events

- componentWillMount
- componentWillReceiveProps
- shouldComponentUpdate
- componentWillUpdate
- componentDidUpdate
- componentDidMount
- componentWillUnmount

Topic B: Component Lifecycle Events

Grouping lifecycle events

- **mounting**

  componentWillMount, componentDidMount, componentWillUnmount

- **updating via props**

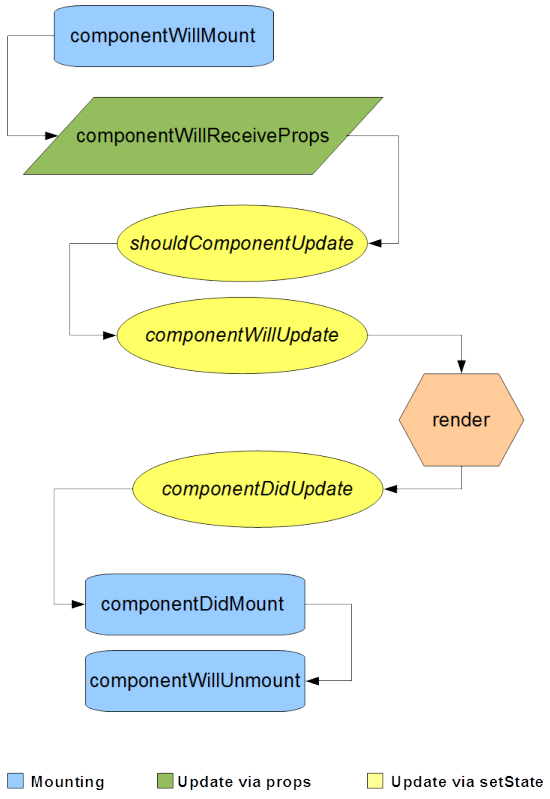
  componentWillReceiveProps, shouldComponentUpdate, componentWillUpdate, componentDidUpdate

- **updating via setState()**

  shouldComponentUpdate, componentWillUpdate, componentDidUpdate

Topic B: Component Lifecycle Events

Events flow



Activity B: Showing the Quantity of Items Added to the Cart

Aim

The aim of this activity is to exploit the lifecycle events of React components

Scenario

We want to avoid multiple occurrences of the same product appearing in the cart. So we want the cart showing a single occurrence of the product and its quantity.

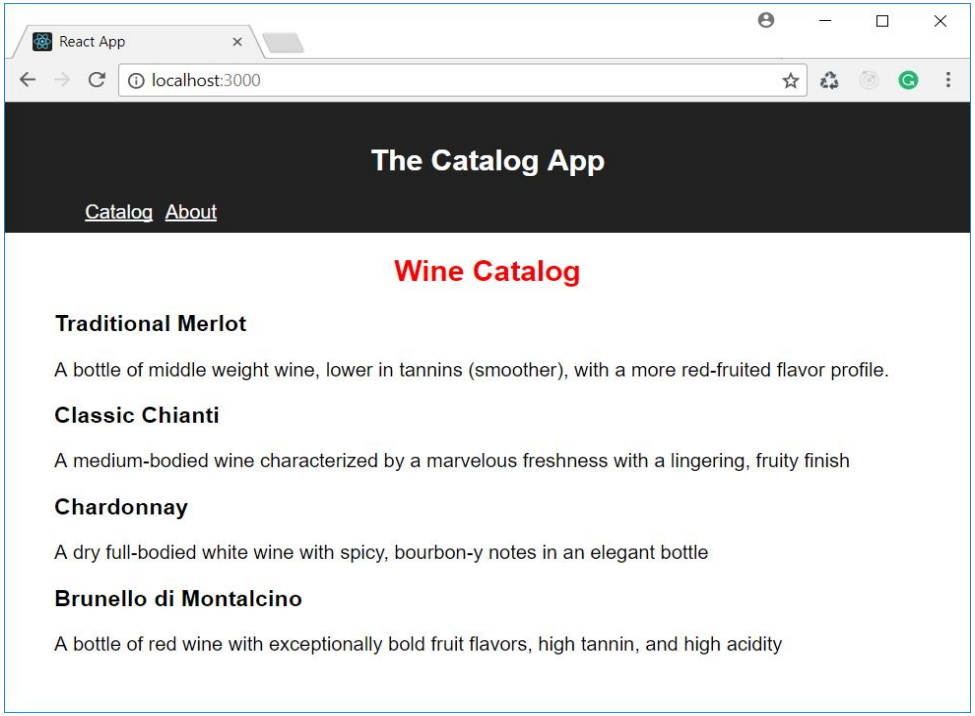
Steps for Completion

1. Use the project changed in the previous activity (or the existing project in **`my-cart-02`** folder).
2. Change the **`Cart`** component in order to show a list of non-duplicated products and their related number of occurrences.

Tip: Handle the **`componentWillReceiveProps`** event to prepare data for the internal state of the **`Cart`** component.

Topic C: Managing Routing

A view is a placeholder in the UI where we can dynamically render one component or another in an exclusive way



Installing React Router

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

- react-router
- react-router-dom
- react-router-native

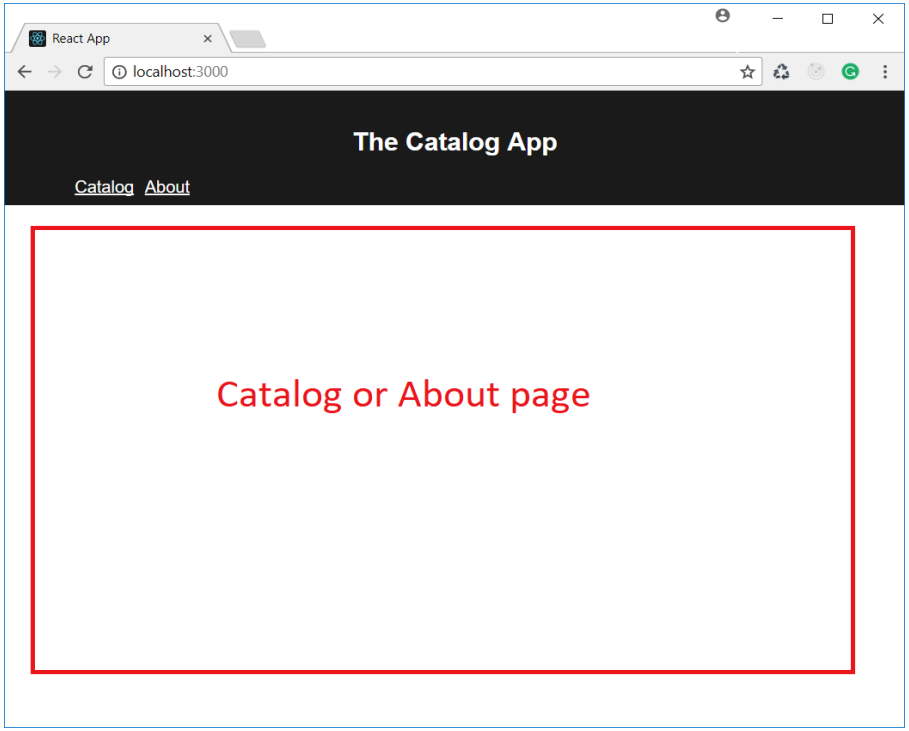
Using the Router

Adding routing capabilities to our application

```
import React from 'react';
…
import { BrowserRouter } from 'react-router-dom'

ReactDOM.render(
  <BrowserRouter>
    <App />
  </BrowserRouter>
  , document.getElementById('root'));
registerServiceWorker();
```

Defining Views

Adding routing capabilities to our application



Defining Views

Creating a view to display components

```
...
import { Switch, Route } from 'react-router-dom'

class App extends Component {
  render() {
    return (
      <div className="App">
        <header className="App-header">...</header>
            <Switch>
              <Route exact path='/' component={Catalog}/>
              <Route path='/about' component={About}/>
            </Switch>
      </div>
    );
  }
}
```

Defining Views

Defining the navigation bar

```
...
import { Switch, Route, Link } from 'react-router-dom'

class App extends Component {
  render() {
    return (
      <div className="App">
        <header className="App-header">…
          <nav>
            <ul>
              <li><Link to='/'>Catalog</Link></li>
              <li><Link to='/about'>About</Link></li>
            </ul>
          </nav>
        </header>
                  <Switch>...</Switch>
      </div>
    );
  }
}
```

Some Notes about the Route Component

The **path** attribute

```
<Switch>
  <Route path='/' component={Catalog}/>
  <Route path='/about' component={About}/>
</Switch>
```

- The path value that matches the starting part of the URL

Some Notes about the Route Component

The **path** attribute

```
<Switch>
  <Route exact path='/' component={Catalog}/>
  <Route path='/about' component={About}/>
</Switch>
```

- The exact attribute forces a strict comparison between the path attribute's value and the URL

Some Notes about the Route Component

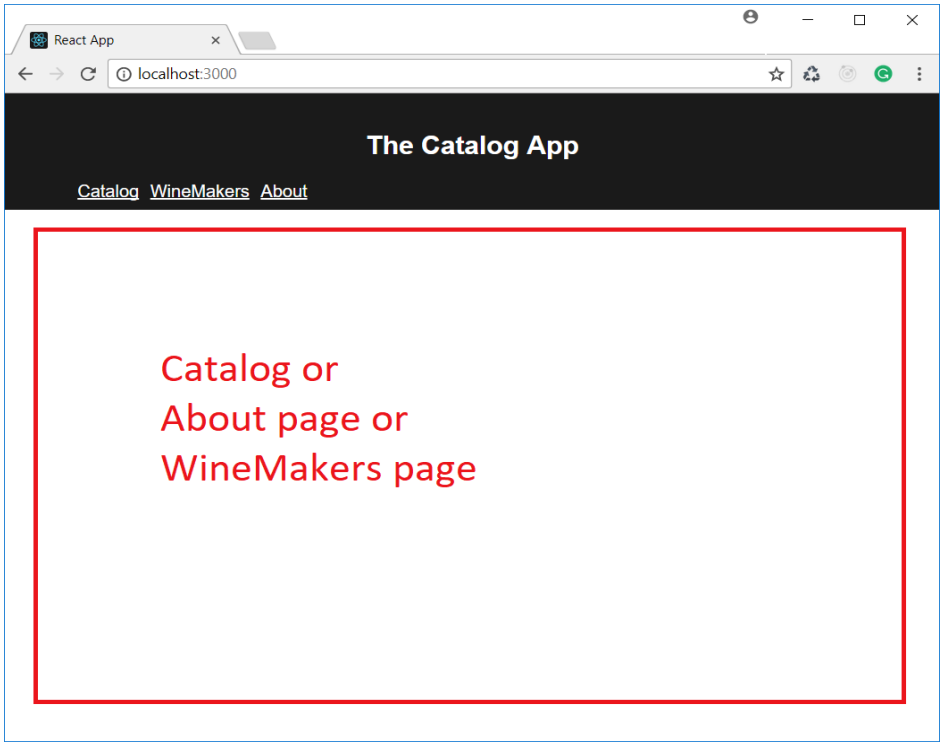
The **path** attribute

```
<Switch>
  <Route exact path='/' component={Catalog}/>
  <Route path='/about' render={() => (<About data={someData}/>)}/>
  <Route path='/footer' children={() => (<Footer />)}/>
</Switch>
```

- The **component** attribute maps a route to a component
- The **render** attribute maps a route to a function returning a React element
- The **children** attribute always renders a function returning a React element

Some Notes about the Route Component

Nested views: the navigation bar



Some Notes about the Route Component

Nested views: the navigation bar

```
...
import WineMakers from './WineMakers';

class App extends Component {
  render() {
    return (
      <div className="App">
        <header className="App-header">...
          <nav>
            <ul> ...
              <li><Link to='/winemakers'>WineMakers</Link></li>
            </ul>
          </nav>
        </header>
                  <Switch> ...
           <Route path='/winemakers' component={WineMakers}/>
                  </Switch>
      </div>
    );
  }
}
```

Some Notes about the Route Component

Nested views: the nested routes

```
...
import WineMaker from './WineMaker';

class WineMakers extends React.Component {
  renderWineMakersList() {
    return <ul>
             <li><Link to="/winemakers/WM1">Wine & Wine</Link></li>
             <li><Link to="/winemakers/WM2">Wine & Co</Link></li>
           </ul>;
  }

  render() {
    return <Switch>
             <Route exact path='/winemakers' render={this.renderWineMakersList}/>
             <Route path='/winemakers/WM1' render={() => (<WineMaker code='WM1' />}/>
             <Route path='/winemakers/WM2' render={() => (<WineMaker code='WM2' />}/>
           </Switch>;
  }
}
```

Path Parameters

Defining parameters

```
render() {
  return <Switch>
           <Route exact path='/winemakers' render={this.renderWineMakersList}/>
           <Route path='/winemakers/:code' component={WineMaker}/>
         </Switch>;
}
```

Path Parameters

Using parameters

```
class WineMaker extends React.Component {
  constructor() {
    super();
    this.wineMakers = [...];
  }


  render() {
    let wineMaker = this.wineMakers.find(wm => wm.code === this.props.match.params.code);

    return <div>
            <h2>{wineMaker.name}</h2>
            <h3>{wineMaker.country}</h3>
            <p>{wineMaker.description}</p>
           </div>;
  }
}
```

Path Parameters

Properties of the **`this.props.match`** object:

- **`params`**: this is an object whose properties match the parameters in the path; that is, the dynamic parts preceded by colons
- **`isExact`**: this is a Boolean indicating that the URL matches the path
- Path: this is the string assigned to the path attribute of the selected Route
- **`url`**: this is the URL that matched the Route's path

Activity C: Adding a View About Shipping Methods

Aim

The aim of this activity is to explore the components provided by React Routing.

Scenario

We want to add a section to our catalog app containing information about the available shipping methods.

Steps for Completion

1. Use the project changed in previous activity (or the existing project in **`my-cart-03`** folder).
2. Create a **`ShippingMethods`** component, showing the list of available shipping methods, and a **`ShippingMethod`** component, showing the details of each shipping method according to the code passed via props (available shipping methods: ECO - Economic delivery, STD - Standard delivery, EXP - Express delivery).
3. Create a navigation bar and a routing configuration that allows us to navigate through the **`Catalog`** and the **`Shipping`** method views.

Summary

In this lesson, we learned how to manage user interaction. In particular, we:

- We managed events that don't involve changes to a component's state
- We handled events that involve changes to a component's state
- We explored the component lifecycle and learned how to customize each phase
- We used React Router's components to configure navigation between components