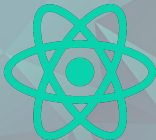


REACT ADVANCED (PT. 2)

Bernardo Cuteri



REACT LIFECYCLE

Each React Component has a lifecycle split in three phases

- Mounting
- Updating
- Unmounting

MOUNTING

React has four built-in methods that gets called in order during mounting:

- ❶ `constructor()`
- ❷ `getDerivedStateFromProps()`
- ❸ `render()`
- ❹ `componentDidMount()`

The `render()` is mandatory, while the others are optional

CONSTRUCTOR

- Invoked when a new component of the class defining it is instantiated

GETDERIVEDSTATEFROMPROPS

- Invoked right before the render() method
- Static method
- Takes **props** and **state** as parameters
- Natural place to set the state object based on props

RENDER

- Required method
- The method that actually outputs HTML to the DOM

COMPONENTDIDMOUNT

- Executed after the component is rendered
- For statements that require that the component is already placed in the DOM

UPDATING

A component is update whenever there is a change in the component **state**. React has 5 built-in methods that gets called, in orderd, when a component is updated

- ❶ `getDerivedStateFromProps()`
- ❷ `shouldComponentUpdate()`
- ❸ `render()`
- ❹ `getSnapshotBeforeUpdate()`
- ❺ `componentDidUpdate()`

The `render()` is again the only mandatory method

GETDERIVEDSTATEFROMPROPS

Discussed previously. It is also executed when a component gets updated

SHOULDCOMPONENTUPDATE

- Boolean method
- Tells React whether it should continue with the rendering or not
- Default returned value is true

RENDER

Discussed previously. It also gets executed on component update.

- Only if `shouldComponentUpdate` returned true

GETSNAPSHOTBEFOREUPDATE

- Provides access to **props** and **state** of the component before the update
- Takes previous props and state as arguments
- If `getSnapshotBeforeUpdate` is implemented, then also `componentDidUpdate` has to be implemented

- Last method executed in the updating phase

The unmounting phase happens when a component is removed from the DOM. React has a single built-in method related to the unmounting phase:

- ❶ `componentWillUnmount()`

REACT EVENTS

React allows to handle user events (e.g. click, mouseover, change)

- Events use camel-case syntax (e.g. `onClick` instead of `onclick`)
- Event handlers are written in curly brackets (e.g. `onClick={handleOnClick}`)
- Good practice is that event-handlers are methods in the component class
- Arrow functions are the best suited option for implementing events
- Events methods can take argument and an event object (that stores event info) can be passed over

React has several integration points with HTML forms

- Forms data is usually handled by the components
- Changes in forms data can be handled by adding an `onChange` event handler
- Forms data is typically intended to be stored in the component state

FORMS SUBMIT

- Place a submit button in the form
- Implement an onSubmit event handler
- Invoke `event.preventDefault()` to prevent default form submit behaviour

MULTIPLE INPUT FIELDS

Implementing more onChangeEvents at once can be made compact:

- Add a **name** attribute to inputs
- Implement a single change event handler with square bracket notation around the property name

INPUTS VALIDATION

Inputs are typically validated on input change or on submit

SOME REACT FORM INPUTS

- Input:
 - Used for short text, numbers, e-mails, checkboxes, and many more
- Textarea:
 - Used for long text
 - In React, the value of a textarea is placed inside a **value** attribute
- Select:
 - Used for selections
 - In react, the selected value is defined with a **value** attribute in the **select** tag

INLINE STYLING

Styling can be done inline:

```
render() {  
  return (  
    <div>  
      <h1 style={{color: "red"}}> Ciccio pasticcio </h1>  
    </div>  
  )  
}
```

JAVASCRIPT OBJECT STYLING

Styling can be placed in a JavaScript object:

```
const mystyle = {  
  color: "red"  
};  
  
class Person extends React.Component {  
  
  render() {  
    return (  
      <div>  
        <h1 style={mystyle}> Ciccio pasticcio </h1>  
      </div>  
    )  
  }  
}
```

STYLESHEET FILE STYLING

Styling can be declared in a css file:

- Place css in a separate .css file
- Import the css file

Best practice: use css modules

- Use extension .module.css
- css rules will be applied only to components importing the module
- No worries about name clashes

Note: remember that class names are defined with the className attribute

HANDLE AJAX REQUESTS IN REACT

By default, React does not include dedicated classes for performing server requests. Some options are:

- Use plain JavaScript
- Use a library

In the tutorial we will consider the Axios React library.

Note: `componentDidMount()` is a good place for doing sever calls

AXIOS

- Promise-based API (for asynchronous calls)
- Supports all HTTP verbs (get, post, delete, ..)
- Is simple and compact

Get started: install axios dependency in your project

```
$ npm install axios --save
```

AXIOS GET EXAMPLE

An example of GET request with axios on a sample API

```
import React from 'react';
import axios from 'axios';
export default class PersonList extends React.Component {
  state = {
    persons: []
  }
  componentDidMount() {
    axios.get('https://jsonplaceholder.typicode.com/users')
      .then(res => {
        const persons = res.data;
        this.setState({ persons: persons });
      })
  }
  render() {
    return (
      <ul>
        { this.state.persons.map(person => <li>{person.name}</li>)}
      </ul>
    )
  }
}
```

Errors can be caught by using **.catch()** (can be placed after **.then()**)

AXIOS POST EXAMPLE

An example of POST request with axios on a sample API

```
import React from 'react';
import axios from 'axios';
export default class PersonList extends React.Component {
  state = {
    name: '',
  }
  handleChange = event => {
    this.setState({ name: event.target.value });
  }
  handleSubmit = event => {
    event.preventDefault();
    const user = {
      name: this.state.name
    };
    axios.post(`https://jsonplaceholder.typicode.com/users`, { user: user })
      .then(res => {
        console.log(res);
        console.log(res.data);
      })
  }
  render() {
    /* */
  }
}
```

AXIOS BASE INSTANCE

Axios allow for the creation of a base instance where we put configuration elements:

```
import axios from 'axios';

export default axios.create({
  |   baseUrl: `http://jsonplaceholder.typicode.com/`
});
```

It can then be used wherever needed:

```
import React from 'react';
import API from './API.js';
export default class PersonList extends React.Component {
  state = {
    |   persons: []
  }
  componentDidMount() {
    |   API.get(`/users`)
    |   .then(res => {
    |     |   const persons = res.data;
    |     |   this.setState({ persons: persons });
    |   })
  }
  /*...*/
```

MAKE YOUR APPLICATION LOOK GOOD

Use a UI library

- for example Material-UI: <https://material-ui.com>

UI libraries contains reusable good-looking standard components (buttons, inputs, menus, etc.)



QUESTIONS??