**Section 1: Getting Started**

****

**Section 3: Understanding the Base Features & Syntax**

**Using create React App**

* npm i create-react-app –g [create-react-app on Github](https://github.com/facebook/create-react-app)
* create-react-app react-complete-guide --scripts-version 1.1.5

**Understanding the Folder Structures**

* **public folder:** root folder sẽ được served bởi web server in the end. Các js file được inject vào file **index.html** nằm trong folder này. Chúng ta sẽ không thêm bất cứ code HTML nào vào đây.

  <body>

    <noscript>

      You need to enable JavaScript to run this app.

    </noscript>

    <div id="root"></div>

*<!--*

*This HTML file is a template.*

*If you open it directly in the browser, you will see an empty page.*

*You can add webfonts, meta tags, or analytics to this file.*

*The build step will place the bundled scripts into the <body> tag.*

*To begin the development, run `npm start` or `yarn start`.*

*To create a production bundle, use `npm run build` or `yarn build`.*

*-->*

  </body>

* Chúng ta sẽ mount app vào div có id là root kia.
* **public\manifest.json:** nơi define some metadata cho ứng dụng

**src\index.js:** rất quen thuộc, chúng ta sẽ inject component **App** vào thẻ div có id là **root** ở file **index.html** nằm trong thư mục public

import React from 'react';

import ReactDOM from 'react-dom';

import './index.css';

import App from './App';

import registerServiceWorker from './registerServiceWorker';

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

registerServiceWorker();

**src\App.js**

import React, { Component } from 'react';

import logo from './logo.svg';

import './App.css';

class App extends Component {

  render() {

    return (

      <div className="App">

        <h1>Hello</h1>

      </div>

    );

  }

}

export default App;

**Understanding Component Basics**

* React is all about creating **components**
  + Ở mỗi component, chúng ta luôn luôn phả return ra một HTML template để render ra screen
  + Các component sẽ kế thừa **Components** từ library của React
  + Đôi khi chúng ta sẽ thấy các file component sẽ có extension là .jsx, do method **render()** sẽ return ra code nhìn như HTMl nhưng **không phải HTML.** Đây là một syntactical sugar, là JS nhưng looking a bit different. Đoạn code ở bên trong return function trong render method sẽ được transpile thành valid javascript

**src\App.js**

**class** App **extends** Component {

  render() {

    return (

      <div className="App">

        <h1>Hello Xuân Hiếu</h1>

      </div>

    );

  }

}

**Understanding JSX**

* Chúng ta sẽ thử thay thế đoạn code bên trong render method của component để có thể render ra kết quản tương tự
* Đây là lý do vì sao chúng ta import **React** tuy nhiên không hề sử dụng nó như cách chúng ta làm ban đầu do module này sẽ được dùng khi code chúng ta được compile
* Chúng ta sẽ không sử dụng cách render này nếu như template của quá phức tạp, chỉ biết để hiểu bản chất.

**src\App.js**

**class** App **extends** Component {

  render() {

*// return (*

*//   <div className="App">*

*//     <h1>Hello Xuân Hiếu</h1>*

*//   </div>*

*// );*

    return React.createElement('div', null,

      React.createElement('h1', {

        className: 'App'

      }, 'Hello Xuân Hiếu'));

  }

}

* Các restrictions của JSX:
  + Không thể sử dụng class attribute để khai báo class cho element như bình thường, thay vào đó chúng ta sử dụng **className**
  + Chỉ được có duy nhất **một** root element.

**Creating a Functional Component**

* Đầu tiên chúng ta sẽ tạo ra một component

**src\Person\Person.js**

import React, { Component } from 'react';

**const** person = () **=>** {

*// đây là jsx syntax, chúng ta cần import React module để sử dụng*

  return <p>I'm a person</p>

}

export default person;

* Sử dụng component này trong **App** component như sau:

**src\App.js**

import React, { Component } from 'react';

import './App.css';

import Person from './Person/Person';

**class** App **extends** Component {

  render() {

    return (

      <div className="App">

        <h1>Hello Xuân Hiếu</h1>

        <Person />

        <Person></Person>

      </div>

    );

  }

}

export default App;

**kết quả**



**Working with components and re-use them**

**-** Sử dụng components giúp chúng ta focus vào một file và dễ dàng hơn trong việc maintain

- Không cần đưa tất cả mọi thứ vào **app.js** file

- Có thể dễ dàng re-use và configurable

* Dynamic content trong React components:
  + Chúng ta có thể sử dụng { } để bọc những đoạn code js của chúng ta trong JSX, khi đó React app sẽ hiểu đây không phải là text mà là những đoạn code JS.
  + Đây là có thể là những single line of code hoặc những function call

**src\Person\Person.js**

**const** person = () **=>** {

*// đây là jsx syntax, chúng ta cần import React module để sử dụng*

  return <p>I'm a Xuân Hiếu and I'm {(new Date()).getFullYear() - 1994} years old</p>

}

export default person;

* Working with Props

**src\Person\Person.js**

import React, { Component } from 'react';

**const** person = (props) **=>** { *// tên param ở đây là up to you, tuy nhiên nên để là props, đây chính là object có các attribute tương ứng là name của các attribute chúng ta truyền vào lúc gọi components*

*// đây là jsx syntax, chúng ta cần import React module để sử dụng*

  return <p>I'm a {props.name} and I'm {props.age} years old</p>

*// đối với class-based components, chúng ta sẽ dùng this.props trong JSX code để get value của các attribute này.*

}

export default person;

**src\App.js**

**class** App **extends** Component {

  render() {

    return (

      <div className="App">

        <h1>Hello Xuân Hiếu</h1>

        <Person name="Xuân Hiếu" age="26"/>

        <Person name="Max" age="30">My hobbies: Racing</Person>

        <Person name="Quân" age="31"></Person>

      </div>

    );

  }

}



* **Children** props:
  + Chúng ta có thể sử dụng props.children để refer đến các elements nằm giữa opening và closing tag của component

**src\Person\Person.js**

**const** person = (props) **=>** { *// tên param ở đây là up to you, tuy nhiên nên để là props, đây chính là object có các attribute tương ứng là name của các attribute chúng ta truyền vào lúc gọi components*

*// đây là jsx syntax, chúng ta cần import React module để sử dụng*

*// props.children ở đây sẽ là bất cứ elements nào nằm giữa opening và closing tag, không chỉ là text mà có thể là các complex HTML*

  return (

    <div>

      <p>I'm a {props.name} and I'm {props.age} years old</p>

      <p>{props.children}</p>

    </div>

  )

*// đối với class-based components, chúng ta sẽ dùng this.props trong JSX code để get value của các attribute này.*

}

**Understanding & Using State**

* Đối với các component được kế thừa từ **Component,** chúng ta có một property là **state,** là nơi mà props được set và passed vào từ outside. Lưu ý rằng chỉ có những components được kế thừa từ class **Component** mới có property này,
* Hiện giờ chúng ta đang hardcode các giá trị như **name** hay **age,** tuy nhiên sử dụng state chúng ta có thể thay đổi các giá trị này.
* Khi sử dụng property **state,** để gọi property này chúng ta phải dùng **this** keyword (ES6 syntax - **class**).
* Khi **state** thay đổi, React sẽ re-render DOM để update theo giá trị mới này của **state.**
* **Props** và **state** là các CORE concepts của React. Thực tế chỉ có những thay đổi ở **props** và **state** thì mới trigger React re-render các components và có thể updated DOM trên browser.

**src\App.js**

**class** App **extends** Component {

  state = {

    persons: [

      { name: 'Xuân Hiếu', age: 26 },

      { name: 'Max', age: 30 },

      { name: 'Quân', age: 31 },

    ]

  }

  render() {

    return (

      <div className="App">

        <h1>Hello Xuân Hiếu</h1>

        <button>Switch Name</button>

        <Person name={this.state.persons[0].name} age={this.state.persons[0].age} />

        <Person name={this.state.persons[1].name} age={this.state.persons[1].age} >My hobbies: Racing</Person>

        <Person name={this.state.persons[2].name} age={this.state.persons[2].age} ></Person>

      </div>

    );

  }

}



**Handling Events with methods**

* Chúng ta sẽ handle sự click event bằng cách như sau:
  + Đối với react, khi gán sự kiện, chúng ta sẽ dùng onClick (ví dụ với sự kiện click, chữ C sẽ được viết hoa, khác với onclick trong gán sự kiện với DOM element bình thường)
  + Phải sử dụng arrow function thì khi sử dụng **this** trong method thì con trỏ **this** lúc này mới trỏ đến instance của component.
  + Đây không phải là syntax của ES6, mà là một syntactic sugar for creating an instance method in the class constructor, chứ không phải là một phần của JS. Nếu chúng ta viết như sau sẽ không sử dụng được con trỏ **this** như mong muốn:

switchNameHandler() {

switchNameHandler = **function**() {

**react-complete-guide\src\App.js**

switchNameHandler = () **=>** {

    console.log('Was clicked');

  }

  render() {

    return (

      <div className="App">

        <h1>Hello Xuân Hiếu</h1>

        <button onClick={this.switchNameHandler}>Switch Name</button>

        <Person name={this.state.persons[0].name} age={this.state.persons[0].age} />

        <Person name={this.state.persons[1].name} age={this.state.persons[1].age} >My hobbies: Racing</Person>

        <Person name={this.state.persons[2].name} age={this.state.persons[2].age} ></Person>

      </div>

    );

  }

* Các event mà chúng ta có thể listen: [events udemy](https://www.udemy.com/course/react-the-complete-guide-incl-redux/learn/lecture/8124210#overview)

**Manipulating the state**

* Để update state, chúng ta phải dùng method của React Component là **setState**
* React sẽ chỉ quan tâm đến những state object bị thay đổi giá trị, nó sẽ không “đụng” đến những obj state khác. **React sẽ tự động merge old và new states với nhau**

**react-complete-guide\src\App.js**

**class** App **extends** Component {

  state = {

    persons: [

      { name: 'Xuân Hiếu', age: 26 },

      { name: 'Max', age: 30 },

      { name: 'Quân', age: 31 },

    ],

    otherState: 'Some other state'

  }

  switchNameHandler = () **=>** { *// arrow function để lexical con trỏ this*

*// DONT DO THIS: this.state.persons[0].name = 'Nguyễn Xuân Hiếu'; // React sẽ không render lại DOM nó "không biết" rằng chúng ta đã thay đổi state của component*

    this.setState({

      persons: [

        { name: 'Nguyễn Xuân Hiếu', age: 26 },

        { name: 'Max', age: 30 },

        { name: 'Quân', age: 31 },

      ]

    })

  }

  render() {

    return (

      <div className="App">

        <h1>Hello Xuân Hiếu</h1>

        <button onClick={this.switchNameHandler}>Switch Name</button>

        <Person name={this.state.persons[0].name} age={this.state.persons[0].age} />

        <Person name={this.state.persons[1].name} age={this.state.persons[1].age} >My hobbies: Racing</Person>

        <Person name={this.state.persons[2].name} age={this.state.persons[2].age} ></Person>

      </div>

    );

  }

}

export default App;

****

**Using the useState() Hook for state manipulation**

* Để manipulating state trong functional component, chúng ta có thể sử dụng **React Hook.** Chúng ta sẽ convert component App ở trên thành một functional component tương tự với Person.
* Sử dụng **useState()** method để manipulating state. Method này sẽ trả về một array với **duy nhất 2 elements.** 
  + Element đầu tiên là current state
  + Element thứ 2 là function cho phép chúng ta update state này.
* Chúng ta sẽ dùng ES6 destructoring để export 2 elements này.
* Sự khác biệt khi sử dụng React Hook là function update state sẽ không merge old và new states, mà nó sẽ replâce hoàn toàn old state và new state. Do đó property **otherState** sẽ không còn tồn tại nữa.
* Khi sử dụng React Hooks, chúng ta có thể dùng **useState()** nhiều lần để quản lý state của component

**react-complete-guide\src\App.js**

import React, { useState } from 'react';

import './App.css';

import Person from './Person/Person';

**const** app = (props) **=>** {

**const** [ personsState, setPersonsState ] = useState({ *// sử dụng method được export bởi react, đòng thời sử dụng destructoring để lấy ra các elements*

    persons: [

      { name: 'Xuân Hiếu', age: 26 },

      { name: 'Max', age: 30 },

      { name: 'Quân', age: 31 },

    ]

  });

**const** [ otherState, setOtherState ] = useState({ otherState: 'Some other state' }) *// không nhất thiết phải là một obj, có thể là một value hay một String*

  console.log(personsState, otherState);

**const** switchNameHandler = () **=>** {

    setPersonsState({ *// thay bằng setPersonsState mà chúng ta đã destructoring ở trên*

      persons: [

        { name: 'Nguyễn Xuân Hiếu', age: 26 },

        { name: 'Max', age: 30 },

        { name: 'Quân', age: 31 },

      ]

    });

  }

  return (

    <div className="App">

      <h1>Hello Xuân Hiếu</h1>

      <button onClick={switchNameHandler}>Switch Name</button>

      <Person name={personsState.persons[0].name} age={personsState.persons[0].age} />

      <Person name={personsState.persons[1].name} age={personsState.persons[1].age} >My hobbies: Racing</Person>

      <Person name={personsState.persons[2].name} age={personsState.persons[2].age} ></Person>

    </div>

  );

}

export default app;

****

****

**Stateful and Stateless components**

* Component như **person** component là một stateless component vì nó không có internal state management
* Ngược lại, **App** component là một stateful component

**Passing method references between components**

* Giả sử chúng ta muốn trigger **switchNameHandler** ngay cả khi click vào các Person component chứ không chỉ ở mỗi button, khi đó chúng ta sẽ passing **switchNameHandler** từ **App** component xuống **person** và tái sử dụng event này.
* Đây là một pattern, chúng ta có thể pass methods như là một props, và sau đó có thể call một method thay đổi state của một component từ một component khác.
* Ngoài ra chúng ta có thể truyền tham số cho **switchNameHandler** theo 2 cách như sau:
  + Sử dụng **bind**
  + Sử dụng **arrow function**

**react-complete-guide\src\App.js**

**class** App **extends** Component {

  state = {

    persons: [

      { name: 'Xuân Hiếu', age: 26 },

      { name: 'Max', age: 30 },

      { name: 'Quân', age: 31 },

    ],

    otherState: 'Some other state'

  }

  switchNameHandler = (newName) **=>** { *// arrow function để lexical con trỏ this*

*// DONT DO THIS: this.state.persons[0].name = 'Nguyễn Xuân Hiếu'; // React sẽ không render lại DOM nó "không biết" rằng chúng ta đã thay đổi state của component*

    this.setState({

      persons: [

        { name: newName, age: 26 },

        { name: 'Max', age: 30 },

        { name: 'Quân', age: 31 },

      ]

    })

  }

*// () => this.switchNameHandler() sẽ là () => { return this.switchNameHandler() }, return ra một function call*

  render() {

    return (

      <div className="App">

        <h1>Hello Xuân Hiếu</h1>

        <button onClick={() **=>** this.switchNameHandler('Nguyễn Xuân Hiếu')}>Switch Name</button>

        <Person

          name={this.state.persons[0].name}

          age={this.state.persons[0].age}

        />

        <Person

          name={this.state.persons[1].name}

          age={this.state.persons[1].age}

          clickName={this.switchNameHandler.bind(this, 'Maximilian')}>

            My hobbies: Racing

        </Person>

        <Person

          name={this.state.persons[2].name}

          age={this.state.persons[2].age} >

        </Person>

      </div>

    );

  }

}

**Adding 2 ways binding**

* Để thêm one way binding khá tương tự như khi chúng ta gán event click:
  + Truyền event handler từ component **App** cho component **Person.**
  + Chúng ta sẽ gặp warning đối với components Person đầu tiên và 3rd do chúng ta chỉ gán **value** props và **onchange** cho component thứ 2, và điều này sẽ khiến render ra một read-only field
  + Hiện giờ chúng ta chỉ có thể update data ở component 2nd. Chúng ta sẽ fix issues này sau khi chúng ta render list các components.

**react-complete-guide\src\App.js**

 nameChangedHandler = event **=>** { *// event sẽ được passed một các tự động bởi React như là normal js*

    this.setState({

      persons: [

        { name: 'Xuân Hiếu', age: 26 },

        { name: event.target.value, age: 30 },

        { name: 'Quân', age: 31 },

      ]

    });

  }

*// () => this.switchNameHandler() sẽ là () => { return this.switchNameHandler() }, return ra một function call*

  render() {

    return (

      <div className="App">

        <h1>Hello Xuân Hiếu</h1>

        <button onClick={() **=>** this.switchNameHandler('Nguyễn Xuân Hiếu')}>Switch Name</button>

        <Person

          name={this.state.persons[0].name}

          age={this.state.persons[0].age}

        />

        <Person

          name={this.state.persons[1].name}

          age={this.state.persons[1].age}

          clickName={this.switchNameHandler.bind(this, 'Maximilian')}

          changeName={this.nameChangedHandler}>

            My hobbies: Racing

        </Person>

        <Person

          name={this.state.persons[2].name}

          age={this.state.persons[2].age} >

        </Person>

      </div>

    );

  }

**react-complete-guide\src\Person\Person.js**

**const** person = (props) **=>** { *// tên param ở đây là up to you, tuy nhiên nên để là props, đây chính là object có các attribute tương ứng là name của các attribute chúng ta truyền vào lúc gọi components*

*// đây là jsx syntax, chúng ta cần import React module để sử dụng*

*// props.children ở đây sẽ là bất cứ elements nào nằm giữa opening và closing tag, không chỉ là text mà có thể là các complex HTML*

  return (

    <div>

      <p onClick={props.clickName}>I'm a {props.name} and I'm {props.age} years old</p>

      <p>{props.children}</p>

      <input

        type="text"

        onChange={props.changeName}

        value={props.name}/>

    </div>

  )

*// đối với class-based components, chúng ta sẽ dùng this.props trong JSX code để get value của các attribute này.*

}

****

**Adding Styling with Stylesheets**

* Có 2 cách để chúng ta có thể thêm styling cho các components (cho đến thời điêm hiện tại)
  + Cách thứ nhất styling cho Person component
    - Add **Person.css** và import file CSS này vào module Person.js
    - Thêm classname tương ứng, sử dụng attribute **className** của React

**react-complete-guide\src\Person\Person.js**

import React from 'react';

import './Person.css'; *// import file css, đây là global style*

**const** person = (props) **=>** { *// tên param ở đây là up to you, tuy nhiên nên để là props, đây chính là object có các attribute tương ứng là name của các attribute chúng ta truyền vào lúc gọi components*

*// đây là jsx syntax, chúng ta cần import React module để sử dụng*

*// props.children ở đây sẽ là bất cứ elements nào nằm giữa opening và closing tag, không chỉ là text mà có thể là các complex HTML*

  return (

    <div className="Person">

      <p onClick={props.clickName}>I'm a {props.name} and I'm {props.age} years old</p>

      <p>{props.children}</p>

      <input

        type="text"

        onChange={props.changeName}

        value={props.name}/>

    </div>

  )

*// đối với class-based components, chúng ta sẽ dùng this.props trong JSX code để get value của các attribute này.*

}

export default person;

* + Cách thứ 2 (styling cho button ở **App** component)
    - Sử dụng inline CSS

**react-complete-guide\src\App.js**

render() {

*// không dùng được một số style như cursor trong inline CSS như thế này.*

*// style này sẽ được applied với scope là chỉ component này thôi.*

**const** style = {

      backgroundColor: 'white',

      font: 'inherit',

      border: '1px solid blue',

      padding: '8px',

      cursor: 'pointer'

    }

    return (

      <div className="App">

        <h1>Hello Xuân Hiếu</h1>

        <button

          style={style}

          onClick={() **=>** this.switchNameHandler('Nguyễn Xuân Hiếu')}>

            Switch Name

        </button>

        <Person

          name={this.state.persons[0].name}

          age={this.state.persons[0].age}

        />

        <Person

          name={this.state.persons[1].name}

          age={this.state.persons[1].age}

          clickName={this.switchNameHandler.bind(this, 'Maximilian')}

          changeName={this.nameChangedHandler}>

            My hobbies: Racing

        </Person>

        <Person

          name={this.state.persons[2].name}

          age={this.state.persons[2].age} >

        </Person>

      </div>

    );

  }

**--------------- --------------- --------------- --------------- --------------- ---------------**

**Section 4: Working with Lists and Conditional**

**Rendering Content Conditional**

- Chúng ta sẽ tận dụng lại project đầu tiên

* Luôn luôn nhớ rằng, code ở trong return () nhìn như HTML nhưng không phải là HTML, mà nó là JSX, do đó sau tất cả code của chúng ta sẽ trở thành JS, vì vậy để hiển thị element một cách conditionally, chúng ta sẽ làm như sau:
  + Giả sử chúng ta muốn thay button switch name bằng button với chức năng ẩn hiện list các persons.

**react-complete-guide\src\App.js**

  state = {

    persons: [

      { name: 'Xuân Hiếu', age: 26 },

      { name: 'Max', age: 30 },

      { name: 'Quân', age: 31 },

    ],

    otherState: 'Some other state',

    showPersons: false

  }

togglePersonsHandler = () **=>** { *// luôn dùng syntax này nếu như chúng ta có ý định sử dụng THIS keyword*

    this.setState({

      showPersons: !this.state.showPersons *// toggle giá trị của showPersons*

    })

  }

return (

      <div className="App">

        <h1>Hello Xuân Hiếu</h1>

        <button

          style={style}

          onClick={this.togglePersonsHandler}>

            Toggle

        </button>

        { this.state.showPersons ?

          <div>

            <Person

              name={this.state.persons[0].name}

              age={this.state.persons[0].age}

            />

            <Person

              name={this.state.persons[1].name}

              age={this.state.persons[1].age}

              clickName={this.switchNameHandler.bind(this, 'Maximilian')}

              changeName={this.nameChangedHandler}>

                My hobbies: Racing

            </Person>

            <Person

              name={this.state.persons[2].name}

              age={this.state.persons[2].age} >

            </Person>

          </div> : null

        }

      </div>

    );

**Handling Dynamic Content “The Javascript way”**

* Sử dụng tenary conditionally không phải lúc nào cũng là cách tối ưu để render data một cách có điều kiện. Chúng ta có thể làm việc render có điều kiện một cách clean hơn bằng cách này:

**react-complete-guide\src\App.js**

render() {

**const** style = {

      backgroundColor: 'white',

      font: 'inherit',

      border: '1px solid blue',

      padding: '8px',

      cursor: 'pointer'

    }

**let** persons = null;

    if (this.state.showPersons) {

      persons = (

        <div>

          <Person

            name={this.state.persons[0].name}

            age={this.state.persons[0].age}

          />

          <Person

            name={this.state.persons[1].name}

            age={this.state.persons[1].age}

            clickName={this.switchNameHandler.bind(this, 'Maximilian')}

            changeName={this.nameChangedHandler}>

              My hobbies: Racing

          </Person>

          <Person

            name={this.state.persons[2].name}

            age={this.state.persons[2].age} >

          </Person>

        </div>

      )

    }

    return (

      <div className="App">

        <h1>Hello Xuân Hiếu</h1>

        <button

          style={style}

          onClick={this.togglePersonsHandler}>

            Toggle

        </button>

        {persons}

      </div>

    );

  }

**Outputting Lists**

* Render lists trong React không sử dụng **v-for** hay **ngFor** như VueJS hay Angular, mà chúng ta sẽ sử dụng cách tương tự với cách chúng ta render conditionally content.
* Chúng ta sẽ sử dụng JS, bởi vì sau cùng JSX cũng sẽ được convert ra JS
* Chúng ta sẽ remove method **SwitchNameHandler** và thay vào đó là **deletePersonHandler** vì không cần thiết.
* Ở **Person** component, chúng ta đã gán

onClick={props.clickName}

do đó ở để truyền event từ Person component với method ở App component chúng ta sẽ làm như sau:

**react-complete-guide\src\App.js**

  deletePersonHandler = index **=>** {

**const** persons = this.state.persons;

    persons.splice(index, 1);

    this.setState({

      persons: persons

    })

  }

render() {

*// không dùng được một số style như cursor trong inline CSS như thế này.*

*// style này sẽ được applied với scope là chỉ component này thôi.*

**…**

**let** persons = null;

    if (this.state.showPersons) {

      persons = (

        <div>

          {this.state.persons.map((person, index) **=>** {

            return <Person

              name={person.name}

              age={person.age}

              clickName={() **=>** this.deletePersonHandler(index)} /> *// có thể dùng bind, phải cùng tên với tên attribute của Person component*

          })}

        </div>

      )

    }

    return (

      <div className="App">

        <h1>Hello Xuân Hiếu</h1>

        <button

          style={style}

          onClick={this.togglePersonsHandler}>

            Toggle

        </button>

        {persons} // gọi lại biến persons ở trên

      </div>

    );

  }

**Updating State Immutably**

* Để tránh việc thay đổi trực tiếp state, chúng ta sẽ tạo ra một bản sao của mảng *persons* như sau, và đây cũng là best practice

**const** persons = this.state.persons.slice(); *// const persons = [...this.state.persons]*

**Lists & keys**

* Key prop là một một property quan trọng mà chúng ta thêm thêm vào mỗi khi render lists of data
  + Key prop giúp React update the list một cách hiệu quả hơn
  + Key nên là một giá trị unique, do đó chúng ta sẽ thêm id vào mỗi person để phân biệt.

**react-complete-guide\src\App.js**

state = {

    persons: [

      { id: 'adad1', name: 'Xuân Hiếu', age: 26 },

      { id: 'aeae2', name: 'Max', age: 30 },

      { id: 'atat3', name: 'Quân', age: 31 },

    ],

    otherState: 'Some other state',

    showPersons: false

  }

    if (this.state.showPersons) {

      persons = (

        <div>

          {this.state.persons.map((person, index) **=>** {

            return <Person

              name={person.name}

              age={person.age}

              clickName={() **=>** this.deletePersonHandler(index)} *// có thể dùng bind, phải cùng tên với tên attribute của Person component*

              key={person.id} />

          })}

        </div>

      )

    }

**Flexible Lists**

* Bây giờ chúng ta sẽ sử dụng đến prop *changeName*mà chúng ta có trong **Person** component để gán sự kiện *onChange* ở **Person** component với method để handler event ở **App** component, ở đây là *nameChangedHandler* method

**react-complete-guide\src\App.js**

  nameChangedHandler = (event, id) **=>** { *// event sẽ được passed một các tự động bởi React như là normal js*

**const** personIndex = this.state.persons.findIndex(p **=>** p.id === id);

**const** person = {...this.state.persons[personIndex]};

    person.name = event.target.value;

**const** persons = [...this.state.persons];

    persons[personIndex] = person

    this.setState({

      persons: persons

    });

  }

**let** persons = null;

    if (this.state.showPersons) {

      persons = (

        <div>

          {this.state.persons.map((person, index) **=>** {

            return <Person

              name={person.name}

              age={person.age}

              clickName={() **=>** this.deletePersonHandler(index)} *// có thể dùng bind, phải cùng tên với tên attribute của Person component*

              key={person.id}

              changeName={(event) **=>** this.nameChangedHandler(event, person.id)} *// phải cùng tên với tên attribute của Person component*

            />

          })}

        </div>

      )

    }

**--------------- --------------- --------------- --------------- --------------- ---------------**

**Section 5: Styling React Components & Elements**

* Các issues đang gặp phải:
  + Style nếu được add inline sẽ có các restriction
  + Style được import vào component sẽ trở thành global style
  + Nếu chúng ta muốn change style dynamically?

**Setting styles dynamically**

**react-complete-guide\src\App.js**

    const style = {

      backgroundColor: 'green',

      color: 'white',

      font: 'inherit',

      border: '1px solid blue',

      padding: '8px',

      cursor: 'pointer'

    }

    let persons = null;

    if (this.state.showPersons) {

      persons = (

        <div>

          {this.state.persons.map((person, index) => {

            return <Person

              name={person.name}

              age={person.age}

              clickName={() => this.deletePersonHandler(index)} // có thể dùng bind, phải cùng tên với tên attribute của Person component

              key={person.id}

              changeName={(event) => this.nameChangedHandler(event, person.id)} // phải cùng tên với tên attribute của Person component

            />

          })}

        </div>

      )

      style.backgroundColor = 'red';

    }

****

****

**Setting Class name dynamically**

**react-complete-guide\src\App.js**

const classes = [];

    if (this.state.persons.length <= 2) {

      classes.push('red'); // red

    }

    if (this.state.persons.length <= 1) {

      classes.push('bold'); // red & bold

    }

    return (

      <div className="App">

        <h1>Hello</h1>

        <p className={classes.join(' ')}>It worked</p>

        <button

          style={style}

          onClick={this.togglePersonsHandler}>

            Toggle

        </button>

        {persons}

      </div>

    );

**Adding and using Radium**

* Quay trở lại với issue chúng ta không thể sử dụng pseudo selectors hoặc các limitation tương tự như media query để responsive. Do đó chúng ta sẽ sử dụng một third party phổ biêns là **Radium**
  + Là một popular package cho React để có thể sử dụng inline style mà vẫn khắc phục được các hạn chế mà chúng ta nói đến
  + **npm i --save radium**

**react-complete-guide\src\App.js**

import React, { Component } from 'react';

import './App.css';

import Person from './Person/Person';

import Radium from 'radium';

class App extends Component {

  …

  // () => this.switchNameHandler() sẽ là () => { return this.switchNameHandler() }, return ra một function call

  render() {

    // không dùng được một số style như cursor trong inline CSS như thế này.

    // style này sẽ được applied với scope là chỉ component này thôi.

    const style = {

      backgroundColor: 'green',

      color: 'white',

      font: 'inherit',

      border: '1px solid blue',

      padding: '8px',

      cursor: 'pointer',

      ':hover': { // sau tất cả, đây sẽ được convert thành JS, có thể coi ':hover' ở đây là một property

        backgroundColor: 'lightgreen',

        color: 'black'

      }

    }

    let persons = null;

    if (this.state.showPersons) {

      persons = (

        <div>

          {this.state.persons.map((person, index) => {

            return <Person

              name={person.name}

              age={person.age}

              clickName={() => this.deletePersonHandler(index)} // có thể dùng bind, phải cùng tên với tên attribute của Person component

              key={person.id}

              changeName={(event) => this.nameChangedHandler(event, person.id)} // phải cùng tên với tên attribute của Person component

            />

          })}

        </div>

      )

      style.backgroundColor = 'red';

      style[':hover'] = { // sau tất cả, đây sẽ được convert thành JS, có thể coi ':hover' ở đây là một property

        backgroundColor: 'salmon',

        color: 'black'

      }

    }

    const classes = [];

    if (this.state.persons.length <= 2) {

      classes.push('red'); // red

    }

    if (this.state.persons.length <= 1) {

      classes.push('bold'); // red & bold

    }

    return (

      <div className="App">

        <h1>Hello</h1>

        <p className={classes.join(' ')}>It worked</p>

        <button

          style={style}

          onClick={this.togglePersonsHandler}>

            Toggle

        </button>

        {persons}

      </div>

    );

  }

}

export default Radium(App); // higher order component, có thể coi như là một component wrapping lấy một component khác, có thể dùng ở cả class-based component hay functional component

**Using Radium for media query**

* Để sử dụng được @media query mà Radium cung cấp, chúng ta phải import { StyleRoot } và “bọc” các component trong <StyleRoot><StyleRoot/> để sử dụng. Ở đây chúng ta wrap trong **App** component.

**react-complete-guide\src\App.js**

return (

      <StyleRoot>

        <div className="App">

          <h1>Hello</h1>

          <p className={classes.join(' ')}>It worked</p>

          <button

            style={style}

            onClick={this.togglePersonsHandler}>

              Toggle

          </button>

          {persons}

        </div>

      </StyleRoot>

    );

* Cta sử dụng @media query trong **Person** component

**react-complete-guide\src\Person\Person.js**

import React from 'react';

import './Person.css'; // import file css, đây là global style

import Radium from 'radium';

const person = (props) => { // tên param ở đây là up to you, tuy nhiên nên để là props, đây chính là object có các attribute tương ứng là name của các attribute chúng ta truyền vào lúc gọi components

  // đây là jsx syntax, chúng ta cần import React module để sử dụng

  // props.children ở đây sẽ là bất cứ elements nào nằm giữa opening và closing tag, không chỉ là text mà có thể là các complex HTML

  const style = {

    '@media (min-width: 500px)': {

      width: '450px'

    }

  }

  return (

    <div className="Person" style={style}>

      <p onClick={props.clickName}>I'm a {props.name} and I'm {props.age} years old</p>

      <p>{props.children}</p>

      <input

        type="text"

        onChange={props.changeName}

        value={props.name}/>

    </div>

  )

  // đối với class-based components, chúng ta sẽ dùng this.props trong JSX code để get value của các attribute này.

}

export default Radium(person); // dùng được với functional-based component

**Introducing Styled Component**

* Đây cũng là một 3rd party module: [https:/styled-components.com/](20https:/styled-components.com/)
  + **npm install --save styled-components**
  + Chúng ta sẽ style lại Person component như cách chúng ta đã làm sử dụng Radium, nhưng lần này chúng ta dùng Styled Component

**react-complete-guide\src\Person\Person.js**

import React from 'react';

import styled from 'styled-components';

**const** StyleDiv = styled.div` // already return a React component, đây là chúng ta đang tạo ra một React component

  {

    width: 60%;

    margin: 16px auto;

    border: 1px solid #eee;

    box-shadow: 0 2px 3px #ccc;

    padding: 16px;

    text-align: center;

  }

  @media (min-width: 500px) {

    width: 450px;

  }

`;

**const** person = (props) **=>** { *// tên param ở đây là up to you, tuy nhiên nên để là props, đây chính là object có các attribute tương ứng là name của các attribute chúng ta truyền vào lúc gọi components*

*// đây là jsx syntax, chúng ta cần import React module để sử dụng*

*// props.children ở đây sẽ là bất cứ elements nào nằm giữa opening và closing tag, không chỉ là text mà có thể là các complex HTML*

**const** style = {

    '@media (min-width: 500px)': {

      width: '450px'

    }

  }

  return (

*// <div className="Person" style={style}>*

    <StyleDiv>

      <p onClick={props.clickName}>I'm a {props.name} and I'm {props.age} years old</p>

      <p>{props.children}</p>

      <input

        type="text"

        onChange={props.changeName}

        value={props.name}/>

    </StyleDiv>

  )

*// đối với class-based components, chúng ta sẽ dùng this.props trong JSX code để get value của các attribute này.*

}

export default person; *// dùng được với functional-based component*

**More on Styled Component**

* Nếu chúng ta inspect các element, chúng ta sẽ thấy các element có class name được generated bởi các 3rd party mà chúng ta sử dụng, và chúng ta có thể tìm thấy class này nếu inspect head element của page.





* Tiếp tục sử dụng Styled Component với App component, và thực hiện dynamically render style.
  + Bỏ những code liên quan đến style theo cách cũ.
  + Chỉ apply style với button.

**react-complete-guide\src\App.js**

**const** StyledButton = styled.button`

  background-color: ${props **=>** props.alt ? 'red' : 'green'};

  color: white;

  font: inherit;

  border: 1px solid blue;

  padding: 8px;

  cursor: pointer;

  &:hover {

    background-color: ${props **=>** props.alt ? 'salmon' : 'lightgreen'};

    color: black;

  }

`;

return (

      <div className="App">

        <h1>Hello</h1>

        <p className={classes.join(' ')}>It worked</p>

        <StyledButton

          alt={this.state.showPersons} *// passed state của app component vào props của StyledButton component*

          onClick={this.togglePersonsHandler}

        >

            Toggle

        </StyledButton>

        {persons}

      </div>

    );

**Working with CSS Modules**

* Một cách khác để chúng ta có thể applied CSS với component scope mà không cần dùng đến các 3rd party module
  + Chúng ta cần phai config lại React app
  + **npm run eject** để chúng ta có thể access config file của webpack
  + Tiến hành config lại như sau (đối với **react-scripts 1.x**):

**react-complete-guide\config\webpack.config.dev.js**



**react-complete-guide\config\webpack.config.prod.js**



* + Thêm các CSS classes ra các file .css như bình thường:

**react-complete-guide\src\App.css**

.App {

  text-align: center;

}

.red {

  color: red;

}

.bold {

  font-weight: bold;

}

.App button {

  background-color: green;

  color: white;

  font: inherit;

  border: 1px solid blue;

  padding: 8px;

  cursor: pointer;

}

.App button:hover {

  background-color: lightgreen;

  color: black;

}

/\* item has both .Button and .Red \*/

.App button.Red {

  background-color: red;

}

.App button.Red:hover {

  background-color: salmon;

}

* + Để sử dụng các classes này, chúng ta sẽ import file CSS vào như sau:

**react-complete-guide\src\App.js**

import classesStyles from './App.css';

* + Applied các CSS classes vào React component:

**react-complete-guide\src\App.js**

render() {

    let persons = null;

    let btnClass = '';

    if (this.state.showPersons) {

      persons = (

        <div>

          {this.state.persons.map((person, index) => {

            return <Person

              name={person.name}

              age={person.age}

              clickName={() => this.deletePersonHandler(index)} // có thể dùng bind, phải cùng tên với tên attribute của Person component

              key={person.id}

              changeName={(event) => this.nameChangedHandler(event, person.id)} // phải cùng tên với tên attribute của Person component

            />

          })}

        </div>

      )

      btnClass = classesStyles.Red;

    }

    const classes = [];

    if (this.state.persons.length <= 2) {

      classes.push(classesStyles.red); // red

    }

    if (this.state.persons.length <= 1) {

      classes.push(classesStyles.bold); // red & bold

    }

    return (

      <div className={classesStyles.App}>

        <h1>Hello</h1>

        <p className={classes.join(' ')}>It worked</p>

        <button

          className={btnClass}

          onClick={this.togglePersonsHandler}

        >

          Toggle

        </button>

        {persons}

      </div>

    );

  }

**Applied CSS module with Media Queries**

* Tiếp tục Sử dụng CSS modules với **Person** component và dùng với media queries

**react-complete-guide\src\Person\Person.css**

.Person {

  width: 60%;

  margin: 16px auto;

  border: 1px solid #eee;

  box-shadow: 0 2px 3px #ccc;

  padding: 16px;

  text-align: center;

}

@media (min-width: 500px) {

  .Person {

    width: 450px;

  }

}

**react-complete-guide\src\Person\Person.js**

import React from 'react';

import styledClasses from './Person.css';

const person = (props) => { // tên param ở đây là up to you, tuy nhiên nên để là props, đây chính là object có các attribute tương ứng là name của các attribute chúng ta truyền vào lúc gọi components

  // đây là jsx syntax, chúng ta cần import React module để sử dụng

  // props.children ở đây sẽ là bất cứ elements nào nằm giữa opening và closing tag, không chỉ là text mà có thể là các complex HTML

  return (

    <div className={styledClasses.Person}>

      <p onClick={props.clickName}>I'm a {props.name} and I'm {props.age} years old</p>

      <p>{props.children}</p>

      <input

        type="text"

        onChange={props.changeName}

        value={props.name}/>

    </div>

  )

  // đối với class-based components, chúng ta sẽ dùng this.props trong JSX code để get value của các attribute này.

}

export default person; // dùng được với functional-based component

**Lưu ý:**

* Đối với react-scripts phiên bản cao hơn, không cần phải chạy lệnh eject và tùy chỉnh lại configuration, đồng thời file CSS sẽ đổi tên thành .**module.css** và có thể import vào để sử dụng bình thường.
* Đối với những class mà chúng ta muốn set scope global, chúng ta cần phả thêmi prefix class name **:global,** ví dụ **:global .Person { … }** và sau đó có thể add **className=”Person”** để sử dụng như bình thường.

**--------------- --------------- --------------- --------------- --------------- ---------------**

**Section 6: Debugging React Apps**

**Using ErrorBoundary**

* Sử dụng ErrorBoundary giúp xác định được nơi code bị sai.

**react-complete-guide\src\App.js**

import React, { Component } from 'react';

class ErrorBoundary extends Component {

  state = {

    hasError: false,

    errorMessage: ''

  }

  componentDidCatch = (error, info) => {

    this.setState({

      hasError: true,

      errorMessage: error

    })

  }

  render() {

    if (this.state.hasError) {

      return <h1>Something went wrong</h1>

    } else {

      return this.props.children;

    }

  }

}

export default ErrorBoundary;

* Wrap component có thể sẽ bị throw error, như ở đây chúng ta sẽ ví dụ với **Person** component:
  + Trong **App** component, sử dụng **key** attribute ở **ErrorBoundary** component.
  + Chỉ sử dụng **ErrorBoundary** ở những component có thể sẽ failed.

**react-complete-guide\src\Person\Person.js**

import React from 'react';

import styledClasses from './Person.css';

const person = (props) => { // tên param ở đây là up to you, tuy nhiên nên để là props, đây chính là object có các attribute tương ứng là name của các attribute chúng ta truyền vào lúc gọi components

  // đây là jsx syntax, chúng ta cần import React module để sử dụng

  // props.children ở đây sẽ là bất cứ elements nào nằm giữa opening và closing tag, không chỉ là text mà có thể là các complex HTML

  const rnd = Math.random();

  if (rnd > 0.5) {

    throw new Error('Something went wrong');

  }

  return (

    <div className={styledClasses.Person}>

      <p onClick={props.clickName}>I'm a {props.name} and I'm {props.age} years old</p>

      <p>{props.children}</p>

      <input

        type="text"

        onChange={props.changeName}

        value={props.name}/>

    </div>

  )

  // đối với class-based components, chúng ta sẽ dùng this.props trong JSX code để get value của các attribute này.

}

export default person; // dùng được với functional-based component

**react-complete-guide\src\App.js**

if (this.state.showPersons) {

      persons = (

        <div>

          {this.state.persons.map((person, index) => {

            return <ErrorBoundary key={person.id}>

                <Person

                name={person.name}

                age={person.age}

                clickName={() => this.deletePersonHandler(index)} // có thể dùng bind, phải cùng tên với tên attribute của Person component

                changeName={(event) => this.nameChangedHandler(event, person.id)} // phải cùng tên với tên attribute của Person component

              />

            </ ErrorBoundary>

          })}

        </div>

      )

      btnClass = classesStyles.Red;

    }

****

**--------------- --------------- --------------- --------------- --------------- ---------------**

**Section 7: Diving into Components & React Internals**

**A better project structure**

* Có thể thấy **Person** component của chúng ta sẽ không cần thiết phải split-up ra nữa, tuy nhiên **App** component của chúng ta thì vẫn có thể tách ra được thành những component nhỏ hơn
  + App component không nên có quá nhiều logic liên quan đến render
  + Chúng ta nên tạo ra một **Persons** component để làm container cho các **Person** component, và chúng ta sẽ move Person folder vào Persons folder
  + Cuối cùng là component chứa button và header
  + Structure chúng ta có đến thời điểm hiện tại:

****

**Spliting an App into Components**

* Đầu tiên chúng ta clean-up **App** component như sau:

**react-complete-guide\src\containers\App.js**

import React, { Component } from 'react';

import classesStyles from './App.css';

import Persons from '../components/Persons/Persons';

import Cockpit from '../components/Cockpit/Cockpit';

class App extends Component {

  state = {

    persons: [

      { id: 'adad1', name: 'Xuân Hiếu', age: 26 },

      { id: 'aeae2', name: 'Max', age: 30 },

      { id: 'atat3', name: 'Quân', age: 31 },

    ],

    otherState: 'Some other state',

    showPersons: false

  }

  deletePersonHandler = index => {

    …

  }

  nameChangedHandler = (event, id) => { // event sẽ được passed một các tự động bởi React như là normal js

    …

  }

  togglePersonsHandler = () => { // luôn dùng syntax này nếu như chúng ta có ý định sử dụng THIS keyword

    this.setState({

      showPersons: !this.state.showPersons // toggle giá trị của showPersons

    })

  }

  // () => this.switchNameHandler() sẽ là () => { return this.switchNameHandler() }, return ra một function call

  render() {

    let persons = null;

    if (this.state.showPersons) {

      persons = <Persons

        persons={this.state.persons}

        clicked={this.deletePersonHandler}

        changed={this.nameChangedHandler}

      />

    }

    return (

      <div className={classesStyles.App}>

        <Cockpit

          showPersons={this.state.showPersons}

          persons={this.state.persons}

          clicked={this.togglePersonsHandler}

        />

        {persons}

      </div>

    );

  }

}

export default App; // higher order component, có thể coi như là một component wrapping lấy một component khác, có thể dùng ở cả class-based component hay functional component

**react-complete-guide\src\containers\App.css**

.App {

  text-align: center;

}

* Tiếp tục đến **Person** và **Persons** component

**react-complete-guide\src\components\Persons\Persons.js**

import React from 'react';

import Person from './Person/Person';

const persons = (props) => {

  return(

    props.persons.map((person, index) => {

      return <Person

        name={person.name}

        age={person.age}

        key={person.id}

        clickName={() => props.clicked(index)}

        changeName={(event) => props.changed(event, person.id)}

      />

    })

  )

}

export default persons;

**react-complete-guide\src\components\Persons\Person\Person.js**

import React from 'react';

import styledClasses from './Person.css';

const person = (props) => {

  return (

    <div className={styledClasses.Person}>

      <p

        onClick={props.clickName}>

          I'm a {props.name} and I'm {props.age} years old

      </p>

      <p>{props.children}</p>

      <input

        type="text"

        onChange={props.changeName}

        value={props.name}/>

    </div>

  )

  // đối với class-based components, chúng ta sẽ dùng this.props trong JSX code để get value của các attribute này.

}

export default person; // dùng được với functional-based component

* Cuối cùng là **Cockpit** component:

**react-complete-guide\src\components\Cockpit\Cockpit.js**

import React from 'react';

import classesStyles from './Cockpit.css';

const cockpit = props => {

  const classes = [];

  let btnClass = '';

  if (props.showPersons) {

    btnClass = classesStyles.Red;

  }

  if (props.persons.length <= 2) {

    classes.push(classesStyles.red); // red

  }

  if (props.persons.length <= 1) {

    classes.push(classesStyles.bold); // red & bold

  }

  return (

    <div className={classesStyles.Cockpit}>

      <h1>Hello</h1>

      <p className={classes.join(' ')}>It worked</p>

      <button

        className={btnClass}

        onClick={props.clicked}

      >

        Toggle

      </button>

    </div>

  )

}

export default cockpit;

**react-complete-guide\src\components\Cockpit\Cockpit.css**

.red {

  color: red;

}

.bold {

  font-weight: bold;

}

.Cockpit button {

  background-color: green;

  color: white;

  font: inherit;

  border: 1px solid blue;

  padding: 8px;

  cursor: pointer;

}

.Cockpit button:hover {

  background-color: lightgreen;

  color: black;

}

/\* item has both .Button and .Red \*/

.Cockpit button.Red {

  background-color: red;

}

.Cockpit button.Red:hover {

  background-color: salmon;

}

**Stateless and Stateful components**

**Class-based vs Functional Components**

****

**react-complete-guide\src\index.js**

import React from 'react';

import ReactDOM from 'react-dom';

import './index.css';

import App from './containers/App';

import registerServiceWorker from './registerServiceWorker';

ReactDOM.render(<App appTitle="Person Manager" />, document.getElementById('root'));

registerServiceWorker();

**react-complete-guide\src\containers\App.js**

return (

      <div className={classesStyles.App}>

        <Cockpit

          title={this.props.appTitle} // Ở đây chúng ta dùng this.props vì đây là class-based component

          showPersons={this.state.showPersons}

          persons={this.state.persons}

          clicked={this.togglePersonsHandler}

        />

        {persons}

      </div>

    );

**Class Component Lifecycle Overview**

* **Lifecycle** component chỉ vailable trong **Class-based** component

****

1. Component Lifecycle – Creation
   1. Construcor(props)
      1. Để customer constructor, chúng ta phải call **super(props)**
      2. DO: set up state
      3. DON’T: cause side-effects
   2. getDerivedStateFromProps(props, state)
      1. DO: sync state
      2. DON’T cause side-effects
   3. render()
      1. prepare & structur your JSX code
   4. Render child components
   5. componentDidMount()
      1. DO: cause side-effects
      2. DON’T update state (trigger re-render)

**react-complete-guide\src\containers\App.js**

import React, { Component } from 'react';

import classesStyles from './App.css';

import Persons from '../components/Persons/Persons';

import Cockpit from '../components/Cockpit/Cockpit';

class App extends Component {

  constructor(props) {

    super(props);

    console.log('[App.js] constructor');

    /\* Chúng ta có thể init state ở đây

    this.state = {

      persons: [

        { id: 'adad1', name: 'Xuân Hiếu', age: 26 },

        { id: 'aeae2', name: 'Max', age: 30 },

        { id: 'atat3', name: 'Quân', age: 31 },

      ],

      otherState: 'Some other state',

      showPersons: false

    }

    \*/

  }

  state = {

    persons: [

      { id: 'adad1', name: 'Xuân Hiếu', age: 26 },

      { id: 'aeae2', name: 'Max', age: 30 },

      { id: 'atat3', name: 'Quân', age: 31 },

    ],

    otherState: 'Some other state',

    showPersons: false

  }

  static getDerivedStateFromProps(props, state) {

    console.log('[App.js] getDerivedStateFromProps', props);

    return state;

  }

  componentWillMount() { // sẽ bị bỏ đi trong tương lai, không nên sử dụng

    console.log('[App.js] componentWillMount');

  }

  componentDidMount() {

    console.log('[App.js] componentDidMount');

  }

  deletePersonHandler = index => {

    const persons = this.state.persons.slice(); // const persons = [...this.state.persons]

    persons.splice(index, 1);

    this.setState({

      persons: persons

    });

  }

  nameChangedHandler = (event, id) => { // event sẽ được passed một các tự động bởi React như là normal js

    const personIndex = this.state.persons.findIndex(p => p.id === id);

    const person = {...this.state.persons[personIndex]};

    person.name = event.target.value;

    const persons = [...this.state.persons];

    persons[personIndex] = person

    this.setState({

      persons: persons

    });

  }

  togglePersonsHandler = () => { // luôn dùng syntax này nếu như chúng ta có ý định sử dụng THIS keyword

    this.setState({

      showPersons: !this.state.showPersons // toggle giá trị của showPersons

    })

  }

  // () => this.switchNameHandler() sẽ là () => { return this.switchNameHandler() }, return ra một function call

  render() {

    console.log('[App.js] render');

    let persons = null;

    if (this.state.showPersons) {

      persons = <Persons

        persons={this.state.persons}

        clicked={this.deletePersonHandler}

        changed={this.nameChangedHandler}

      />

    }

    return (

      <div className={classesStyles.App}>

        <Cockpit

          title={this.props.appTitle} // Ở đây chúng ta dùng this.props vì đây là class-based component

          showPersons={this.state.showPersons}

          persons={this.state.persons}

          clicked={this.togglePersonsHandler}

        />

        {persons}

      </div>

    );

  }

}

export default App; // higher order component, có thể coi như là một component wrapping lấy một component khác, có thể dùng ở cả class-based component hay functional component

1. Component Lifecycle – Update: khi mà props hoặc state thay đổi sẽ là 2 triggers khiến cho component có thể được re-evaluated 🡺 Update
   1. getDerivedStateFromProps(props, state): hiếm khi sử dụng, sẽ không dùng đến trong course này.
      1. DO: sync state to Props
      2. DON’T: cause side-effects
   2. shouldComponentUpdate(nextProps, nextState): có thể dùng để cancel updating process
      1. DO: quyết định xem continue hoặc not
      2. DON’T: cause side-effects
   3. Render()
   4. Update child components props
   5. getSnapshotBeforeUpdate(prevProps, prevState): ít khi sử dụng, có thể sử dụng trong những trường hợp muốn scrolling đến position cần thiết …
      1. DO: last-minute DOM ops
      2. DON’T: cause side-effects
   6. componentDidUpdate(): đây là một hook hay được sử dụng, giả sử khi chúng ta muốn fetch data từ server
      1. DO: cause side-effects
      2. DON’T: update state (trigger re-render)
   7. Ngoài ra còn một số hook nữa tuy nhiên không còn được hỗ trợ và sẽ bị removed đi trong tương lai.

**react-complete-guide\src\components\Persons\Persons.js**

import React, { Component } from 'react';

import Person from './Person/Person';

**class** Persons **extends** Component { *// convert thành class-based component để sử dụng lifecycle hook*

*/\**

*static getDerivedStateFromProps(props, state) { // rất ít khi sử dụng*

*console.log('[Persons.js] getDerivedStateFromProps');*

*return state;*

*}*

*\*/*

  shouldComponentUpdate(nexxtProps, nexxtState) { *// bắt buộc phải return true hoặc false, không return gì không phải là 1 option*

    console.log('[Persons.js] shouldComponentUpdate');

    return true;

  }

  getSnapshotBeforeUpdate(prevProps, prevState) {

    console.log('[Persons.js] getSnapshotBeforeUpdate');

    return { message: 'Snapshot!!!' }; *// sẽ có waring nếu chúng ta không return gì.*

  }

  componentDidUpdate(presProps, prevState, snapshot) { *// snapshot ở đây chính là return từ getSnapshotBeforeUpdate*

    console.log('[Persons.js] componentDidUpdate');

    console.log('Snapshot: ', snapshot);

  }

  render() {

    console.log('[Persons.js] rendering ...');

    return(

      this.props.persons.map((person, index) **=>** {

        return <Person

          name={person.name}

          age={person.age}

          key={person.id}

          clickName={() **=>** this.props.clicked(index)}

          changeName={(event) **=>** this.props.changed(event, person.id)}

        />

      })

    )

  }

}

export default Persons;

****

**Using useEffect() in Functional Components**

* Functional component không có cách để quản lý lifecycle hook như class-based component, tuy nhiên chúng ta có thể dùng **useEffect()** để can thiệp vào các *lifecycle* của component.
* Để sử dụng **useEffect()** chúng ta phải truyền vào một callback function, và function sẽ được execute ở mọi hook (creation, update …)
* Combile cả **componentDidMount** và **componentDidUpdate**
* Có thể dùng **useEffect()** bao nhiêu lần tùy vào mục đích.

**react-complete-guide\src\components\Cockpit\Cockpit.js**

import React, { useEffect } from 'react';

import classesStyles from './Cockpit.css';

**const** cockpit = props **=>** {

  useEffect(() **=>** {

    console.log('[Cockpit.js] useEffect');

*// HTTP request ...*

    setTimeout(() **=>** {

      alert('Saved data to cloud');

    }, 1000);

  }, [props.persons]); *// chỉ được executed khi mà props.persons thay đổi. Giả sử khi chúng ta changed input hay delete một item ...*

  useEffect(() **=>** {

    console.log('[Cockpit.js] useEffect');

*// HTTP request ...*

    setTimeout(() **=>** {

      alert('Saved data to cloud in the first time rendered');

    }, 2000);

  }, []); *// tricky để chỉ run khi render first time, vì không có item nào thay đổi giá trị.*

**const** classes = [];

**let** btnClass = '';

  if (props.showPersons) {

    btnClass = classesStyles.Red;

  }

  if (props.persons.length <= 2) {

    classes.push(classesStyles.red); *// red*

  }

  if (props.persons.length <= 1) {

    classes.push(classesStyles.bold); *// red & bold*

  }

  return (

    <div className={classesStyles.Cockpit}>

      <h1>{props.title}</h1>

      <p className={classes.join(' ')}>It worked</p>

      <button

        className={btnClass}

        onClick={props.clicked}

      >

        Toggle

      </button>

    </div>

  )

}

export default cockpit;

**Cleaning up with Lifecycle Hookss & useEffects()**

* Giả sử đổi với **Persons** component, mỗi khi component được removed (toggle), chúng ta muốn cleanup (clean database, remove các event listener …), chúng ta sẽ sử dụng một hook hữu ích là **componentWillUnmount**

**react-complete-guide\src\components\Persons\Persons.js**

  componentWillUnmount() {

    console.log('[Persons.js] componentWillUnmount');

  }

* Đối với functional-based component
  + Chúng ta sẽ áp dụng với component **Cockpit**, tuy nhiên phải thêm một button để ẩn component này đi trong **App** component để demo cleanup feature.
  + Thêm **showCockpit** để toggle hiển thị của cockpit
  + Add event cho button

**react-complete-guide\src\containers\App.js**

state = {

    persons: [

      { id: 'adad1', name: 'Xuân Hiếu', age: 26 },

      { id: 'aeae2', name: 'Max', age: 30 },

      { id: 'atat3', name: 'Quân', age: 31 },

    ],

    otherState: 'Some other state',

    showPersons: false,

    showCockpit: true

  }

return (

      <div className={classesStyles.App}>

        <button onClick={

          () **=>** {

            this.setState(

              { showCockpit: !this.state.showCockpit }

            )}

          }>

            Remove Cockpit

        </button>

        {this.state.showCockpit ? <Cockpit

          title={this.props.appTitle} *// Ở đây chúng ta dùng this.props vì đây là class-based component*

          showPersons={this.state.showPersons}

          persons={this.state.persons}

          clicked={this.togglePersonsHandler}

        /> : null}

        {persons}

      </div>

    );

**react-complete-guide\src\components\Cockpit\Cockpit.js**

useEffect(() **=>** {

    console.log('[Cockpit.js] useEffect');

*// HTTP request ...*

    setTimeout(() **=>** {

      alert('Saved data to cloud in the first time rendered');

    }, 2000);

    return () **=>** { *// function này sẽ được run khi component bị removed*

      console.log('[Cockpit.js] cleanup work in useEffect');

    }

  }, []); *// tricky để chỉ run khi render first time, vì không có item nào thay đổi giá trị.*

* Chúng ta có thể tận dụng cleanup này để clear timeout

**react-complete-guide\src\components\Cockpit\Cockpit.js**

useEffect(() **=>** {

    console.log('[Cockpit.js] useEffect');

*// HTTP request ...*

**const** timer = setTimeout(() **=>** {

      alert('Saved data to cloud in the first time rendered'); *// fake*

    }, 2000);

    return () **=>** { *// function này sẽ được run khi component bị removed, tương đương với hook componentWillUnmount của class-based component*

*// clearTimeout(timer); // clear timeout để chỉ hiển thị alert khi render*

      console.log('[Cockpit.js] cleanup work in useEffect');

    }

  }, []); *// tricky để chỉ run khi render first time, vì không có item nào thay đổi giá trị.*

**Using shouldComponentUpdate for optimization**

* Có một issue với app của chúng ta, là trong **Person** component hiện tại **shouldComponentUpdate** đang luôn luôn return về true, do đó ngay cả khi component bị tác động là **Cockpit** component, thì **Person** component cũng sẽ bị render lại. 🡺 Đây là điều chúng ta hoàn toàn không mong muốn, giảm hiệu năng của ứng dụng.

**react-complete-guide\src\components\Persons\Persons.js**

shouldComponentUpdate(nextProps, nextState) { *// bắt buộc phải return true hoặc false, không return gì không phải là 1 option*

    console.log('[Persons.js] shouldComponentUpdate');

    if (nextProps.persons !== this.props.persons) { *// chỉ có thể compare đúng nếu như chúng ta thực sự created 1 copy của persons như chúng ta đã làm*

      return true;

    }

    return false;

  }

**Optimizing Functional Components with React.memo()**

* Đối với functional-based component, chúng ta sẽ dùng [**React.memo()**](https://viblo.asia/p/reactmemo-Ljy5VM7Glra)như sau:
  + Thay vì passed cả persons array, giờ chúng ta chỉ passed persons.length và React có thể “catch” được sự thay đổi này.

**react-complete-guide\src\components\Cockpit\Cockpit.js**

import React, { useEffect } from 'react';

import classesStyles from './Cockpit.css';

**const** cockpit = props **=>** {

useEffect(() **=>** {

    console.log('[Cockpit.js] useEffect');

*// HTTP request ...*

    setTimeout(() **=>** {

      alert('Saved data to cloud in the first time rendered'); *// fake*

    }, 1000);

    return () **=>** { *// function này sẽ được run khi component bị removed, tương đương với hook componentWillUnmount của class-based component*

*// clearTimeout(timer); // clear timeout để chỉ hiển thị alert khi render*

      console.log('[Cockpit.js] cleanup work in useEffect');

    }

  }, []); *// tricky để chỉ run khi render first time, vì không có item nào thay đổi giá trị.*

  useEffect(() **=>** {

    console.log('[Cockpit.js] 2nd useEffect');

    return () **=>** { *// function này sẽ được run khi component bị removed, tương đương với hook componentWillUnmount của class-based component*

*// clearTimeout(timer); // clear timeout để chỉ hiển thị alert khi render*

      console.log('[Cockpit.js] 2nd cleanup work in useEffect');

    }

  });

**const** classes = [];

**let** btnClass = '';

  if (props.showPersons) {

    btnClass = classesStyles.Red;

  }

  if (props.personsLength <= 2) {

    classes.push(classesStyles.red); *// red*

  }

  if (props.personsLength <= 1) {

    classes.push(classesStyles.bold); *// red & bold*

  }

  return (

    <div className={classesStyles.Cockpit}>

      <h1>{props.title}</h1>

      <p className={classes.join(' ')}>It worked</p>

      <button

        className={btnClass}

        onClick={props.clicked}

      >

        Toggle

      </button>

    </div>

  )

}

export default React.memo(cockpit);

**When should you optimize?**

* Chỉ thực hiện tối ưu khi cần thiết
  + Không nên làm dụng nếu như component (thường là child component) luôn luôn được update nếu như parent component được update và ảnh hưởng đến child component

**PureComponents instead of shouldComponentUpdate**

* Thay vì việc phải compare như thế này nếu muốn theo dõi tất cả các properties thay đổi để trigger việc re-render:

**src\components\Persons\Persons.js**

  shouldComponentUpdate(nextProps, nextState) { // bắt buộc phải return true hoặc false, không return gì không phải là 1 option

    console.log('[Persons.js] shouldComponentUpdate');

    if (

      nextProps.persons !== this.props.persons ||

      nextProps.changed !== this.props.changed ||

      nextProps.clicked !== this.props.clicked

    ) { // chỉ có thể compare đúng nếu như chúng ta thực sự created 1 copy của persons như chúng ta đã làm

      console.log('[Persons.js] got updated');

      return true;

    }

    return false;

  }

Chúng ta có thể sử dụng

**src\components\Persons\Persons.js**

import React, { PureComponent } from 'react';

import Person from './Person/Person';

class Persons extends PureComponent { // convert thành class-based component để sử dụng lifecycle hook

getSnapshotBeforeUpdate(prevProps, prevState) {

    console.log('[Persons.js] getSnapshotBeforeUpdate');

    return { message: 'Snapshot!!!' }; // sẽ có waring nếu chúng ta không return gì.

  }

  componentDidUpdate(prevProps, prevState, snapshot) { // snapshot ở đây chính là return từ getSnapshotBeforeUpdate

    console.log('[Persons.js] componentDidUpdate');

    console.log('Snapshot: ', snapshot);

  }

  componentWillUnmount() {

    console.log('[Persons.js] componentWillUnmount');

  }

  render() {

    console.log('[Persons.js] rendering ...');

    return(

      this.props.persons.map((person, index) => {

        return <Person

          name={person.name}

          age={person.age}

          key={person.id}

          clickName={() => this.props.clicked(index)}

          changeName={(event) => this.props.changed(event, person.id)}

        />

      })

    )

  }

}

export default Persons;

**How React updates The DOM**

* Render() is called: là một suggestion HTML should look like in the end



**Rendering adjacent JSX elements**

* Hiện tại chúng ta luôn luôn phải return duy nhất 1 container element trong một component, không thể render 2 elements được.
* Tuy nhiên thực tế React chp phép render array of adjacent elements nếu như tất cả các elements đều có **key** attribute, ví dụ với **Persons** component:

**src\components\Persons\Persons.js**

render() {

    console.log('[Persons.js] rendering ...');

    return(

      this.props.persons.map((person, index) => {

        return <Person

          name={person.name}

          age={person.age}

          key={person.id}

          clickName={() => this.props.clicked(index)}

          changeName={(event) => this.props.changed(event, person.id)}

        />

      })

    )

}

do đó chúng ta có thể return nhiều element sitting next to each other bằng cách return 1 array các element và thêm **key** attribute cho các elements này:

**src\components\Persons\Person\Person.js**

import React, { Component } from 'react';

// import styledClasses from './Person.css';

class Person extends Component { // convert thành class-based component để sử dụng lifecycle hook

  render() {

    console.log('[Person.js] rendering ...');

    return [

        <p

          key="i1"

          onClick={this.props.clickName}>

            I'm a {this.props.name} and I'm {this.props.age} years old

        </p>,

        <p key="i2">{this.props.children}</p>,

        <input

          key="i3"

          type="text"

          onChange={this.props.changeName}

          value={this.props.name}

        />

    ]

  }

  // đối với class-based components, chúng ta sẽ dùng this.props trong JSX code để get value của các attribute này.

}

export default Person;

* Cách làm thứ 2 là chúng ta sẽ create một wrapping component mà không thực sự render một actual HTML mà chỉ đơn giản là để fullfill

**src\hoc\Auxiliary.js**

// đối với window, tên của component này nên đặt là Auxiliary.js, còn đối với Mac là Aux

const auxiliary = props => props.children; // children là một property đơn giản output tất cả những gì bên trong openning và closing tag

export default auxiliary;

**src\components\Persons\Person\Person.js**

import React, { Component } from 'react';

// import styledClasses from './Person.css';

import Auxiliary from '../../../hoc/Auxiliary';

class Person extends Component { // convert thành class-based component để sử dụng lifecycle hook

  render() {

    console.log('[Person.js] rendering ...');

    return (

      <Auxiliary>

        <p

          onClick={this.props.clickName}>

            I'm a {this.props.name} and I'm {this.props.age} years old

        </p>,

        <p>{this.props.children}</p>,

        <input

          type="text"

          onChange={this.props.changeName}

          value={this.props.name}

        />

      </Auxiliary>

    )

  }

  // đối với class-based components, chúng ta sẽ dùng this.props trong JSX code để get value của các attribute này.

}

export default Person;

**Using React.Fragment**

* Từ React 16.2, react cung cấp một built-in aux component

**src\components\Persons\Person\Person.js**

import React, { Component, Fragment } from 'react';

// import styledClasses from './Person.css';

import Auxiliary from '../../../hoc/Auxiliary';

class Person extends Component { // convert thành class-based component để sử dụng lifecycle hook

  render() {

    console.log('[Person.js] rendering ...');

    return (

      <Fragment>

        <p

          onClick={this.props.clickName}>

            I'm a {this.props.name} and I'm {this.props.age} years old

        </p>,

        <p>{this.props.children}</p>,

        <input

          type="text"

          onChange={this.props.changeName}

          value={this.props.name}

        />

      </Fragment>

    )

  }

  // đối với class-based components, chúng ta sẽ dùng this.props trong JSX code để get value của các attribute này.

}

export default Person;

**High Order Component (HOC)**

* Một số example của sử dụng HOC như muốn create một component setup a div containss a class hoặc sets up class, hoặc chúng ta có thể add các error handling mà chúng ta muốn wrap bất cứ component nào đó có thể throw ra các exception

**src\hoc\WithClass.js**

import React from 'react';

const withClass = props => (

  <div

    className={props.classes}>

      {props.children}

  </div>

);

export default withClass;

**src\containers\App.js**

return (

      <WithClass classes={classesStyles.App}>

        <button onClick={

          () => {

            this.setState(

              { showCockpit: !this.state.showCockpit }

            )}

          }>

            Toggle Cockpit

        </button>

        {this.state.showCockpit ? <Cockpit

          title={this.props.appTitle} // Ở đây chúng ta dùng this.props vì đây là class-based component

          showPersons={this.state.showPersons}

          personsLength={this.state.persons.length}

          clicked={this.togglePersonsHandler}

        /> : null}

        {persons}

      </WithClass>

    );

**Another Form of HOC**

**src\hoc\withClass.js**

import React from 'react';

const withClass = (WrappedComponent, className) => { // WrappedComponent sẽ là một ref đến một component được wrapped bên trong HOC, các args sau sẽ phụ thuộc vào mục đích sử dụng HOC của chúng ta, như ở trường hợp này, chúng ta muốn một DIV với className được truyền vào. Chúng ta sẽ không dùng withClass như là một component, do đó lowerCase first letter (naming convention)

  return props => { // có thể viết gọn return => ( ... )

    return (

      <div className={className}>

        <WrappedComponent />

      </div>

    )

  }

};

export default withClass;

* Chú ý cách chúng ta đổi tên Upper Case của **WithClass** thành **withClass** để rõ ràng hơn

**src\containers\App.js**

import withClass from '../hoc/withClass';

return (

      <Auxiliary classes={classesStyles.App}>

        <button onClick={

          () => {

            this.setState(

              { showCockpit: !this.state.showCockpit }

            )}

          }>

            Toggle Cockpit

        </button>

        {this.state.showCockpit ? <Cockpit

          title={this.props.appTitle} // Ở đây chúng ta dùng this.props vì đây là class-based component

          showPersons={this.state.showPersons}

          personsLength={this.state.persons.length}

          clicked={this.togglePersonsHandler}

        /> : null}

        {persons}

      </Auxiliary>

    );

export default withClass(App, classesStyles.App);

**Passing unknown props**

* Bây giờ chúng ta sẽ tái sử dụng withClass HOC để style lại cho person component.

**src\components\Persons\Person\Person.js**

import React, { Component } from 'react';

// import styledClasses from './Person.css';

import classesStyles from './Person.css';

import Auxiliary from '../../../hoc/Auxiliary';

import withClass from '../../../hoc/withClass';

class Person extends Component { // convert thành class-based component để sử dụng lifecycle hook

  render() {

    console.log('[Person.js] rendering ...');

    return (

      <Auxiliary>

        <p

          onClick={this.props.clickName}>

            I'm a {this.props.name} and I'm {this.props.age} years old

        </p>,

        <p>{this.props.children}</p>,

        <input

          type="text"

          onChange={this.props.changeName}

          value={this.props.name}

        />

      </Auxiliary>

    )

  }

  // đối với class-based components, chúng ta sẽ dùng this.props trong JSX code để get value của các attribute này.

}

export default withClass(Person, classesStyles.Person);

* Kết quả:

****

* Lý do: Data bị misssing do những gì chúng ta vừa làm là lấy **Person** component và trước khi export nó, chúng ta pass component vào **withClass** function, và trong **withClass** function, nó nhận vào component và output component này, và không hề passing vào một **props** nào.
  + Chúng ta có thể passing props như sau:

**src\hoc\WithClass.js**

import React from 'react';

const withClass = (WrappedComponent, className) => { // WrappedComponent sẽ là một ref đến một component được wrapped bên trong HOC, các args sau sẽ phụ thuộc vào mục đích sử dụng HOC của chúng ta, như ở trường hợp này, chúng ta muốn một DIV với className được truyền vào. Chúng ta sẽ không dùng withClass như là một component, do đó lowerCase first letter (naming convention)

  return props => { // có thể viết gọn return => ( ... ), props ở đây chính là props của WrappedComponent

    return (

      <div className={className}>

        <WrappedComponent {...props} />

      </div>

    )

  }

};

export default withClass;

**Setting state correctly**

* Behind screen setState sẽ không ngay lập tức trigger an update of state của component trong re-render cycle, do đó nếu chúng ta muốn update state dựa vào giá trị state cũ, chúng ta nên làm như sau:

**src\containers\App.js**

  nameChangedHandler = (event, id) => { // event sẽ được passed một các tự động bởi React như là normal js

    const personIndex = this.state.persons.findIndex(p => p.id === id);

    const person = {...this.state.persons[personIndex]};

    person.name = event.target.value;

    const persons = [...this.state.persons];

    persons[personIndex] = person

    this.setState((prevState, props) => {

      return {

        persons: persons,

        changeCounter: prevState.changeCounter + 1

      }

    });

  }

**Using PropType**

* Sử dụng một package mới: **npm i --save prop-types,** cách dùng như sau:

**src\components\Persons\Person\Person.js**

Person.propTypes = {

  clickName: PropTypes.func,

  name: PropTypes.string,

  age: PropTypes.number,

  changeName: PropTypes.func

};

export default withClass(Person, classesStyles.Person);

**Using Refs**

* Giả sử sau khi render các Person component, chúng ta muốn input cuối cùng được focus, và React cung cấp cho chúng ta cách để select một element, bằng concept **refs. Refs** gần giống như **key,** và có rất nhiều cách để sử dụng attribute này.
  + Cách 1:

**src\components\Persons\Person\Person.js**

class Person extends Component { // convert thành class-based component để sử dụng lifecycle hook

  componentDidMount() {

    this.inputElement.focus();

  }

  render() {

    console.log('[Person.js] rendering ...');

    return (

      <Auxiliary>

        <p

          onClick={this.props.clickName}>

            I'm a {this.props.name} and I'm {this.props.age} years old

        </p>,

        <p>{this.props.children}</p>,

        <input

          ref={(inputEle) => {this.inputElement = inputEle}}

          type="text"

          onChange={this.props.changeName}

          value={this.props.name}

        />

      </Auxiliary>

    )

  }

  // đối với class-based components, chúng ta sẽ dùng this.props trong JSX code để get value của các attribute này.

}

* + Cách 2:

**src\components\Persons\Person\Person.js**

class Person extends Component { // convert thành class-based component để sử dụng lifecycle hook

  constructor(props) {

    super(props);

    this.inputElementRef = React.createRef();

  }

  componentDidMount() {

    // this.inputElement.focus();

    this.inputElementRef.current.focus();

  }

  render() {

    console.log('[Person.js] rendering ...');

    return (

      <Auxiliary>

        <p

          onClick={this.props.clickName}>

            I'm a {this.props.name} and I'm {this.props.age} years old

        </p>,

        <p>{this.props.children}</p>,

        <input

          // ref={(inputEle) => {this.inputElement = inputEle}} // cách 1

          ref={this.inputElementRef}

          type="text"

          onChange={this.props.changeName}

          value={this.props.name}

        />

      </Auxiliary>

    )

  }

  // đối với class-based components, chúng ta sẽ dùng this.props trong JSX code để get value của các attribute này.

}

**Refs with React Hooks**

* Đối với functional-component, React hook cũng cung cấp cho chúng ta concept tương tự. Giả sử đối với **Cockpit** component, chúng ta muốn button toggle được click mỗi khi page được load.
  + **toggleBtnRef** sẽ được click khi component được render, do đó chúng ta sẽ dùng trong hook được chạy khi mà component được render (comment out the setTimeOut)

**src\components\Cockpit\Cockpit.js**

import React, { useEffect, useRef } from 'react';

const cockpit = props => {

const toggleBtnRef = useRef(null); // create a ref đến element

  useEffect(() => {

    console.log('[Cockpit.js] useEffect');

    // HTTP request ...

    /\*

    setTimeout(() => {

      alert('Saved data to cloud in the first time rendered'); // fake

    }, 1000);

    \*/

    toggleBtnRef.current.click(); // click ở đây sau khi component đã được render

    return () => { // function này sẽ được run khi component bị removed, tương đương với hook componentWillUnmount của class-based component

      // clearTimeout(timer); // clear timeout để chỉ hiển thị alert khi render

      console.log('[Cockpit.js] cleanup work in useEffect');

    }

  }, []); // tricky để chỉ run khi render first time, vì không có item nào thay đổi giá trị.

. . . . .

return (

    <div className={classesStyles.Cockpit}>

      <h1>{props.title}</h1>

      <p className={classes.join(' ')}>It worked</p>

      <button

        ref={toggleBtnRef} // trỏ element đến ref item

        className={btnClass}

        onClick={props.clicked}

      >

        Toggle

      </button>

    </div>

   )

}

export default React.memo(cockpit);

**Understanding Pop Chain Problems**

* Để resolved việc passed data giữa các component qua parent component và qua nhiều “tầng” component parent - children, ta dùng Context API (gần giống service concept của Angular)
  + Chúng ta cần set các default value cho object này
  + Context này thực chất là một component
  + Wrapped lấy tất cả các parts của application cần access đến context này để lấy dữ liệu.
  + Chúng ta sẽ **Provider** và **Consumer** đối với Context component này

**src\context\auth-context.js**

import React from 'react';

**const** authContext = React.createContext({ *// globally available JS object (or value, array ...), có thể passed vào các component mà không cần dùng đến props, globally available không hoàn toàn chính xác, mà chúng ta có thể decide xem object sẽ available ở những đâu*

  authenticated: false,

  login: () **=>** {}

});

export default authContext;

* Đầu tiên chúng ta thực hiện **provide** context ở **App** component:
  + Import **AuthContext** và sử dụng như component.
  + Chỉ cần wrapped với **persons** và **Cockpit** component

**src\containers\App.js**

        <AuthContext.Provider

          value={

            {

              authenticated: this.state.authenticated,

              login: this.loginHandler

            }

          }>

          {this.state.showCockpit ? <Cockpit

            title={this.props.appTitle} *// Ở đây chúng ta dùng this.props vì đây là class-based component*

            showPersons={this.state.showPersons}

            personsLength={this.state.persons.length}

            clicked={this.togglePersonsHandler}

          /> : null}

          {persons}

        </AuthContext.Provider>

* Sau đó **consum** context này ở **Person** và **Cockpit** component bằng **AuthContext.consumer**

**src\components\Persons\Person\Person.js**

render() {

    console.log('[Person.js] rendering ...');

    return (

      <Auxiliary>

        {*/\* AuthContext.Consumer sẽ không take JSX code as a child, mà là 1 function, do đó chúng ta sẽ nhận một func bên trong, với arg là context object \*/*}

        <AuthContext.Consumer>

          {(context) **=>** context.authenticated ? <p>Authenticated!!!</p> : <p>Please login!!!</p>}

        </AuthContext.Consumer>

          <p

            onClick={this.props.clickName}>

              I'm a {this.props.name} and I'm {this.props.age} years old

          </p>,

          <p>{this.props.children}</p>,

          <input

*// ref={(inputEle) => {this.inputElement = inputEle}} // cách 1*

            ref={this.inputElementRef}

            type="text"

            onChange={this.props.changeName}

            value={this.props.name}

          />

      </Auxiliary>

    )

  }

**src\components\Cockpit\Cockpit.js**

  return (

    <div className={classesStyles.Cockpit}>

      <h1>{props.title}</h1>

      <p className={classes.join(' ')}>It worked</p>

      <button

        ref={toggleBtnRef} *// trỏ element đến ref item*

        className={btnClass}

        onClick={props.clicked}

      >

        Toggle

      </button>

      <AuthContext.Consumer>

        {(context **=>**

          <button onClick={context.login}>

            Login

          </button>

        )}

      </AuthContext.Consumer>

    </div>

  )

**contextType & useContext()**

* Có những trường hợp chúng ta muốn access context không chỉ ở render mà trong **componentDidMount** hay những hook khác.
  + Đối với class-based component:

**src\components\Persons\Person\Person.js**

**class** Person **extends** Component { *// convert thành class-based component để sử dụng lifecycle hook*

**constructor**(props) {

    super(props);

    this.inputElementRef = React.createRef();

  }

**static** contextType = AuthContext;

  componentDidMount() {

*// this.inputElement.focus();*

    this.inputElementRef.current.focus();

    console.log(this.context.authenticated);

  }

  render() {

    console.log('[Person.js] rendering ...');

    return (

      <Auxiliary>

          {this.context.authenticated ? <p>Authenticated!!!</p> : <p>Please login!!!</p>}

          <p

            onClick={this.props.clickName}>

              I'm a {this.props.name} and I'm {this.props.age} years old

          </p>,

          <p>{this.props.children}</p>,

          <input

*// ref={(inputEle) => {this.inputElement = inputEle}} // cách 1*

            ref={this.inputElementRef}

            type="text"

            onChange={this.props.changeName}

            value={this.props.name}

          />

      </Auxiliary>

    )

  }

}

* + Đối với functional-component

**src\components\Cockpit\Cockpit.js**

**const** cockpit = props **=>** {

**const** toggleBtnRef = useRef(null); *// create a ref đến element*

**const** authContext = useContext(AuthContext); *// React sẽ make connection behind the scenes*

. . . . . .

  return (

    <div className={classesStyles.Cockpit}>

      <h1>{props.title}</h1>

      <p className={classes.join(' ')}>It worked</p>

      <button

        ref={toggleBtnRef} *// trỏ element đến ref item*

        className={btnClass}

        onClick={props.clicked}

      >

        Toggle

      </button>

      <button onClick={authContext.login}>

        Login

      </button>

    </div>

  )

}

**Section 8: A Real App: the Burger Builder (Basic Version)**

**Planning a React App**

1. Component Tree / Component Structure
2. Application State (Data)
3. Components vs Containers

Chapter 119 + 120

**Creating a Layout Component**

**src\components\Layout\Layout.js**

import React from 'react';

import Auxiliary from '../../hoc/Auxiliary';

**const** layout = (props) **=>** (

  <Auxiliary>

    <div>Toolbar, SideDrawer, Backdrop</div>

    <main>

      {props.children}

    </main>

  </Auxiliary>

)

export default layout;

**src\App.js**

**class** App **extends** Component {

  render() {

    return (

      <div>

        <Layout>

          <p>test</p>

        </Layout>

      </div>

    );

  }

}

****

**Starting Implementation of The Burger Builder Container**

**src\containers\BurgerBuilder\BurgerBuilder.js**

import React from 'react';

import Aux from '../../hoc/Auxiliary';

**class** BurgerBuilder **extends** React.Component {

  render() {

    return(

      <Aux>

        <div>Burger</div>

        <div>Build Controller</div>

      </Aux>

    );

  }

}

export default BurgerBuilder;

**Adding a Dynamic Ingredient Component + Adding Prop Type to Validation**

**src\components\Burger\BurgerIngredient\BurgerIngredient.js**

import React from 'react';

import PropTypes from 'prop-types';

import classes from './BurgerIngredient.css'; *// file style được pre-created để sử dụng*

**class** BurgerIngredient **extends** React.Component {

  render() {

**let** ingredient = null;

    switch (this.props.type) {

      case ('bread-bottom'):

        ingredient = <div className={classes.BreadBottom}></div>;

        break;

      case ('bread-top'):

        ingredient = (

          <div className={classes.BreadTop}>

            <div className={classes.Seeds1}></div>

            <div className={classes.Seeds2}></div>

          </div>

        );

        break;

      case ('meat'):

        ingredient = <div className={classes.Meat}></div>;

        break;

      case ('cheese'):

        ingredient = <div className={classes.Cheese}></div>;

        break;

      case ('salad'):

        ingredient = <div className={classes.Salad}></div>;

        break;

      case ('bacon'):

        ingredient = <div className={classes.Bacon}></div>;

        break;

      default:

        ingredient = null;

    }

    return ingredient;

  }

}

BurgerIngredient.propTypes = {

  type: PropTypes.string.isRequired

}

export default BurgerIngredient;

**Starting the Burger Component**

* Burger component này sẽ là container cho các ingredient. Tạm thời chúng ta sẽ hardcode các ingredients như sau:

**src\components\Burger\Burger.js**

import React from 'react';

import classes from './Burger.css';

import BurgerIngredient from './BurgerIngredient/BurgerIngredient';

**const** burger = props **=>** {

  return(

    <div className={classes.Burger}>

      <BurgerIngredient type="bread-top" />

      <BurgerIngredient type="cheese" />

      <BurgerIngredient type="meat" />

      <BurgerIngredient type="bread-bottom" />

    </div>

  );

}

export default burger;

* Sử dụng component này bên trong **BurgerBuilder** component

**src\containers\BurgerBuilder\BurgerBuilder.js**

**class** BurgerBuilder **extends** React.Component {

  render() {

    return(

      <Aux>

        <Burger />

        <div>Build Controller</div>

      </Aux>

    );

  }

}

****

**Outputting Burger Ingredients Dynamically**

* **State** của ứng dụng sẽ được đặt trong **BurgerBuilder** component.
* Tạm thời hardcord giá trị của state, và thay đổi **Burgur** component để có thể dynamically render các ingredients:

**src\components\Burger\Burger.js**

**const** burger = props **=>** {

**const** transformedIngredients = Object.keys(props.ingredients) *// get an array of key items*

    .map(ingredientName **=>** {

      return [...Array(props.ingredients[ingredientName])].map((\_, i) **=>** {

        return <BurgerIngredient key={ingredientName + i} type={ingredientName} />

      });

    });

  return(

    <div className={classes.Burger}>

      <BurgerIngredient type="bread-top" />

      {transformedIngredients}

      <BurgerIngredient type="bread-bottom" />

    </div>

  );

}

****

**src\containers\BurgerBuilder\BurgerBuilder.js**

**class** BurgerBuilder **extends** React.Component {

state = {

    ingredients: {

      salad: 1,

      bacon: 1,

      cheese: 2,

      meat: 2

    }

  }

………

}

**Calculating the Ingredient Sum Dynamically**

* Nếu như user chưa chọn một ingredient nào, chúng ta muốn show một thông báo cho người dùng lựa chọn ingredient. Áp dụng reduce để transform mảng **transformedIngredients** thành mảng 1 chiều và checking the length của array này:

**src\components\Burger\Burger.js**

**const** burger = props **=>** {

**let** transformedIngredients = Object.keys(props.ingredients) *// get an array of key items*

    .map(ingredientName **=>** {

      return [...Array(props.ingredients[ingredientName])].map((\_, i) **=>** {

        return <BurgerIngredient key={ingredientName + i} type={ingredientName} />

      });

    })

    .reduce((arr, el) **=>** {

      return arr.concat(el);

    }, []);

  if (transformedIngredients.length === 0) {

    transformedIngredients = <p>Please start adding some ingredients</p>

  }

  return(

    <div className={classes.Burger}>

      <BurgerIngredient type="bread-top" />

      {transformedIngredients}

      <BurgerIngredient type="bread-bottom" />

    </div>

  );

}

Nếu user chưa select bất cứ ingredient nào thì burger sẽ display như thế này:



**Adding the Build Controls Component**

* **BuildControls** component sẽ contain nhiều **BuildControl** component. Component này sẽ có nhiệm vụ thêm hay xóa bớt các ingredient theo lựa chọn của user.

**src\components\Burger\BuildControls\BuildControl\BuildControl.js**

import classes from './BuildControl.css';

**const** buildControl = props **=>** {

  return (

    <div className={classes.BuildControl}>

      <div className={classes.Label}>{props.label}</div>

      <button className={classes.Less}>Less</button>

      <button className={classes.More}>More</button>

    </div>

  )

}

export default buildControl;

**Outputting Multiple Build Controls**

**src\components\Burger\BuildControls\BuildControls.js**

import BuildControl from './BuildControl/BuildControl';

**const** controls = [

  { label: 'Salad', type: 'salad' },

  { label: 'Bacon', type: 'bacon' },

  { label: 'Cheese', type: 'cheese' },

  { label: 'Meat', type: 'meat' }

]

**const** buildControls = props **=>** {

  return (

    <div className={classes.BuildControls}>

      {controls.map(control **=>** {

        return (

          <BuildControl key={control.label} label={control.label}/>

        )

      })}

    </div>

  )

}

Sử dụng **BuildControls** component trong **BurgerBuilder** component:

**src\containers\BurgerBuilder\BurgerBuilder.js**

render() {

    return(

      <Aux>

        <Burger ingredients={this.state.ingredients} />

        <BuildControls />

      </Aux>

    );

  }



**Connecting state to build control**

**src\containers\BurgerBuilder\BurgerBuilder.js**

**class** BurgerBuilder **extends** React.Component {

**constructor**(props) {

    super(props);

    console.log('[BurgerBuilder component] constructor')

  }

  state = {

    ingredients: {

      salad: 0,

      bacon: 0,

      cheese: 0,

      meat: 0

    },

    totalPrice: 4

  }

  addIngredientHandler = type **=>** {

**const** oldCount = this.state.ingredients[type];

**const** updatedCount = oldCount + 1;

**const** updatingIngredient = {...this.state.ingredients};

    updatingIngredient[type] = updatedCount;

**const** priceAddition = INGREDIENT\_PRICES[type];

**const** newPrice = this.state.totalPrice + priceAddition;

    this.setState({

      totalPrice: newPrice,

      ingredients: updatingIngredient

    });

  }

  removeIngredientHandler = type **=>** {

**const** oldCount = this.state.ingredients[type];

    if (oldCount <= 0) {

      return;

    }

**const** updatedCount = oldCount - 1;

**const** updatingIngredient = {...this.state.ingredients};

    updatingIngredient[type] = updatedCount;

**const** priceDeduction = INGREDIENT\_PRICES[type];

**const** newPrice = this.state.totalPrice - priceDeduction;

    this.setState({

      totalPrice: newPrice,

      ingredients: updatingIngredient

    });

  }

  render() {

**const** disabledInfo = {

      ...this.state.ingredients

    }

    for (**let** key in disabledInfo) {

      disabledInfo[key] = disabledInfo[key] <= 0;

    }

    return(

      <Aux>

        <Burger ingredients={this.state.ingredients} />

        <BuildControls

          ingredientAdded={this.addIngredientHandler}

          ingredientRemoved={this.removeIngredientHandler}

          disabled={disabledInfo}

        />

      </Aux>

    );

  }

}

**src\components\Burger\BuildControls\BuildControls.js**

**const** buildControls = props **=>** {

  return (

    <div className={classes.BuildControls}>

      {controls.map(control **=>** {

        return (

          <BuildControl

            key={control.label}

            label={control.label}

            added={() **=>** props.ingredientAdded(control.type)}

            removed={() **=>** props.ingredientRemoved(control.type)}

            disabled={props.disabled[control.type]}

          />

        )

      })}

    </div>

  )

}

**src\components\Burger\BuildControls\BuildControl\BuildControl.js**

**const** buildControl = props **=>** {

  return (

    <div className={classes.BuildControl}>

      <div className={classes.Label}>{props.label}</div>

      <button

        disabled={props.disabled}

        onClick={props.removed}

        className={classes.Less}>

          Less

      </button>

      <button

        onClick={props.added}

        className={classes.More}>

          More

      </button>

    </div>

  )

}