**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 + Displaying and Updating the Burger Price**

**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}

price={this.state.totalPrice}

        />

      </Aux>

    );

  }

}

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

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

  return (

    <div className={classes.BuildControls}>

<p>Current price: <strong>{props.price.toFixed(2)}</strong></p>

      {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>

  )

}

****

**Adding the Order button**

* Add checkout button (Order)
  + Chỉ cho phép checkout nếu như burger có ít nhất 1 ingredient được lựa chọn
  + Button này được đặt ở **BuildControls** component
  + Kiểm tra xem có purchasable ở BurgerBuilder component và passed xuống **BuildControls** component.

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

  state = {

    ingredients: {

      salad: 0,

      bacon: 0,

      cheese: 0,

      meat: 0

    },

    totalPrice: 4,

    purchasable: false

  }

  updatePurchaseState(updatingIngredients) {

**const** ingredients = {...updatingIngredients};

**const** sum = Object.keys(ingredients)

      .map(igKey **=>** {

        return ingredients[igKey]

      })

      .reduce((sum, el) **=>** {

        return sum + el;

      }, 0);

    this.setState({

      purchasable: sum > 0

    })

  }

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

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

  return (

    <div className={classes.BuildControls}>

      <p>Current price: <strong>{props.price.toFixed(2)}</strong></p>

      {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]}

          />

        );

      })}

      <button

        disabled={!props.purchasable}

        className={classes.OrderButton}>

          ORER NOW

      </button>

    </div>

  )

}

****

**Creating the Order Summary Modal**

* Ideal của việc tạo ra một modal là có một wrapping element provides styling và sau đó wraps các content mà chúng ta muốn show trong modal đó.
  + Đồng thời chúng ta sẽ tạo thêm một **Backdrop** component vì chúng ta sẽ re-use component này
  + **Modal** component sẽ được put ở **BurgerBuilder** component
  + **OrderSummary** là component được wrapped trong **Modal** component với chức năng summary lại các ingredients và output ra cho User. Truyền ingredients từ **BurgerBuilder** vào component.

**src\components\UI\Modal\Modal.js**

const modal = props => (

  <div

    className={classes.Modal}

    style={{

      display: props.show ? 'block' : 'none',

      transforrm: props.show ? 'translateY(0)' : 'translateY(-100vh)',

      opacity: props.show ? '1' : '0'

    }

  }>

    {props.children}

  </div>

)

export default modal;

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

const orderSummary = props => {

  const ingredientSummary = Object.keys(props.ingredients)

    .map(igKey => {

      return (

        <li key={igKey}>

          <span style={{textTransform: 'capitalize'}}>

            {igKey}</span>: {props.ingredients[igKey]}

        </li>);

    });

  return (

  <Aux>

    <h3>Your Order</h3>

    <p>A delicious burger with following ingredients:</p>

    <ul>

      {ingredientSummary}

    </ul>

    <p>Continue to Checkout?</p>

  </Aux>

  )

};

export default orderSummary;

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

  render() {

    const disabledInfo = {

      ...this.state.ingredients

    }

    for (let key in disabledInfo) {

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

    }

    return(

      <Aux>

        <Modal show={this.state.purchasing}>

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

        </Modal>

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

        <BuildControls

          ingredientAdded={this.addIngredientHandler}

          ingredientRemoved={this.removeIngredientHandler}

          disabled={disabledInfo}

          purchasable={this.state.purchasable}

          price={this.state.totalPrice}

          ordered={this.purchaseHandler}

        />

      </Aux>

    );

  }

**Implementing the Backdrop Component**

* Đây là component làm “nên” cho modal
* Nếu bấm ra ngoài modal là backdrop thì modal sẽ gone
* Chúng ta sẽ put component này vào trong **Modal** component.

**src\components\UI\Backdrop\Backdrop.js**

import classes from './Backdrop.css';

const backdrop = props => {

  return (

    props.show ? <div className={classes.Backdrop} onClick={props.clicked}></div> : null

  )

}

export default backdrop;

**src\components\UI\Backdrop\Backdrop.css**

.Backdrop {

  width: 100%;

  height: 100%;

  position: fixed;

  z-index: 100; /\* phải bé hơn modal \*/

  left: 0;

  top: 0;

  background-color: rgba(0, 0, 0, 0.5);

}

**src\components\UI\Modal\Modal.js**

const modal = props => (

  <Aux>

    <Backdrop show={props.show} clicked={props.modalClosed}/>

    <div

      className={classes.Modal}

      style={{

        display: props.show ? 'block' : 'none',

        transforrm: props.show ? 'translateY(0)' : 'translateY(-100vh)',

        opacity: props.show ? '1' : '0'

      }

    }>

      {props.children}

    </div>

  </Aux>

)

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

  purchaseCancelHandler = () => {

    this.setState({ purchasing: false });

  }

    return(

      <Aux>

        <Modal

          show={this.state.purchasing}

          modalClosed={this.purchaseCancelHandler}

        >

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

        </Modal>

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

        <BuildControls

          ingredientAdded={this.addIngredientHandler}

          ingredientRemoved={this.removeIngredientHandler}

          disabled={disabledInfo}

          purchasable={this.state.purchasable}

          price={this.state.totalPrice}

          ordered={this.purchaseHandler}

        />

      </Aux>

    );

**Adding a Custom Button Component**

* Tạo ra một seprated component cho các button để re-use
* Các button component nên có style riêng, được passing từ bên ngoài vào.

**src\components\UI\Button\Button.js**

import classes from './Button.css';

const button = props => (

  <button

    onClick={props.clicked}

    className={[classes.Button, classes[props.btnType]].join(' ')}

  >

    {props.children}

  </button>

)

export default button;

**Adding a Toolbar**

**Theo Code không note ra nữa.**

Để import ảnh dùng trong app:

**src\components\Logo\Logo.js**

import React from 'react';

import classes from './Logo.css';

import burgerLogo from '../../assets/images/burger-logo.png';

const logo = \_ => (

  <div className={classes.Logo}>

    <img src={burgerLogo} alt="Burger"/>

  </div>

);

export default logo;

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

**Section 9: Reaching out to the Web (Http / Ajax)**

**Understanding Httl Request in React**

**Understanding our Project and Introducing Axios**

**-** npm i axios –save

**Creating a Http Request to GET Data**

* Chúng ta sẽ make HTTP request ở dâu?
  + Chúng ta sẽ dùng ở lifecycle hook mà chúng ta có thể cause các side-effect 🡺 componentDidMount
  + HTTP request là một side effect, nó không làm ảnh hưởng đến logic nhưng nó fetching new data và nếu component sử dụng data mới này để làm output thì sự thay đổi này chắc chắn sẽ effect đến application.
  + **componentDidMount** là nơi thích hợp để gây ra các side effect nhưng không updating state vì nó sẽ trigger re-render

****

**src\containers\Blog\Blog.js**

**class** Blog **extends** Component {

  componentDidMount() {

*// https://jsonplaceholder.typicode.com/posts*

    axios.get('https://jsonplaceholder.typicode.com/posts')

      .then(res **=>** {

        console.log(res);

      })

  }

  render() {

   . . . . .

  }

}

**Rendering Fetched Data to the Screen + Transforming Data**

**src\containers\Blog\Blog.js**

**class** Blog **extends** Component {

  state = {

    posts: []

  }

  componentDidMount() {

*// https://jsonplaceholder.typicode.com/posts*

    axios.get('https://jsonplaceholder.typicode.com/posts')

      .then(res **=>** {

**const** posts = res.data.slice(0, 4);

**const** updatedPosts = posts.map(post **=>** {

          return {

            ...post,

            author: 'Xuân Hiếu' *// tạm thời hardcode giá trị của author*

          }

        })

        this.setState({

          posts: updatedPosts

        });

      });

  }

  render() {

**const** posts = this.state.posts.map(post **=>** {

      return <Post title={post.title} userId={post.author} key={post.id}/>

    });

    return (

      <div>

        <section className="Posts">

          {posts}

        </section>

        <section>

          <FullPost />

        </section>

        <section>

          <NewPost />

        </section>

      </div>

    );

  }

}

Có thể sử dụng async / await như sau (optionally):

**src\containers\Blog\Blog.js**

**async** componentDidMount() {

*// https://jsonplaceholder.typicode.com/posts*

**const** { data } = await axios.get('https://jsonplaceholder.typicode.com/posts');

**const** posts = data.slice(0, 4);

**const** updatedPosts = posts.map(post **=>** {

      return {

        ...post,

        author: 'Xuân Hiếu' *// tạm thời hardcode giá trị của author*

      }

    });

    this.setState({

      posts: updatedPosts

    });

  }

**Making a Post Selectable + Fetching data on update (Without creating infinite loop)**

* Bây giờ chúng ta sẽ thực hiện việc load single post khi click on the article.
  + Ở **Blog** component chúng ta sẽ thêm property **selectedPostId** để lấy id của post được selected
  + Passing ID này xuống **FullPost** component

**src\components\Post\Post.js**

**const** post = (props) **=>** (

  <article className="Post" onClick={props.clicked}>

    <h1>{props.title}</h1>

    <div className="Info">

      <div className="Author">{props.userId}</div>

    </div>

  </article>

);

**src\containers\Blog\Blog.js**

**class** Blog **extends** Component {

  state = {

    posts: [],

    selectedPostId: null

  }

  postSelectedHandler = id **=>** {

    this.setState({

      selectedPostId: id

    });

  }

**async** componentDidMount() {

*// https://jsonplaceholder.typicode.com/posts*

**const** { data } = await axios.get('https://jsonplaceholder.typicode.com/posts');

**const** posts = data.slice(0, 4);

**const** updatedPosts = posts.map(post **=>** {

      return {

        ...post,

        author: 'Xuân Hiếu' *// tạm thời hardcode giá trị của author*

      }

    });

    this.setState({

      posts: updatedPosts

    });

  }

  render() {

**const** posts = this.state.posts.map(post **=>** {

      return (

        <Post

          title={post.title}

          userId={post.author}

          key={post.id}

          clicked={() **=>** this.postSelectedHandler(post.id)}

        />

      )

    });

    return (

      <div>

        <section className="Posts">

          {posts}

        </section>

        <section>

          <FullPost

            id={this.state.selectedPostId}

          />

        </section>

        <section>

          <NewPost />

        </section>

      </div>

    );

  }

}

* Xử lý ở **FullPost** component
  + Lifecycle hook mà chúng ta nên sử dụng để call a HTTP request, component của chúng ta đã được render ngay từ đầu, nên bây giờ chúng ta không sử dụng *componentDidMount* nữa.
  + Chúng ta sẽ sử dụng *componentDidUpdate* là nơi để call HTTP request, tuy nhiên chúng ta sẽ gặp một infinite loop do component update state và sẽ lại trigger việc update component

**src\components\FullPost\FullPost.js**

**class** FullPost **extends** Component {

  state = {

    title: null,

    body: null,

    id: null

  }

**async** componentDidUpdate() {

*// https://jsonplaceholder.typicode.com/posts/1*

    if (this.props.id && this.state.id !== this.props.id) { *// check nếu như chúng ta GET post lần đầu tiên thì mới call HTTP request*

**const** { data } = await axios.get(`https://jsonplaceholder.typicode.com/posts/${this.props.id}`);

      this.setState({

        title: data.title,

        body: data.body,

        id: data.id

      });

    }

  }

  render() {

**let** post = <p style={{ textAlign: "center", color: 'red' }}>Please select a Post!</p>;

    if (this.props.id) {

      post = (

        <div className="FullPost">

          <h1>{this.state.title}</h1>

          <p>{this.state.body}</p>

          <div className="Edit">

            <button className="Delete">Delete</button>

          </div>

        </div>

      );

    }

    return post;

  }

}

**POSTing Data to the Server**

**src\components\NewPost\NewPost.js**

**class** NewPost **extends** Component {

  state = {

    title: "",

    content: "",

    author: "Max",

  };

  postDataHandler = **async** () **=>** {

*// POST https://jsonplaceholder.typicode.com/posts*

**const** post = {

      title: this.state.title,

      body: this.state.content,

      author: this.state.author

    }

**const** {data} = await axios.post('https://jsonplaceholder.typicode.com/posts', post);

    console.log(data);

  };

  render() {

    return (

      <div className="NewPost">

        <h1>Add a Post</h1>

        <label>Title</label>

        <input

          type="text"

          value={this.state.title}

          onChange={(event) **=>** this.setState({ title: event.target.value })}

        />

        <label>Content</label>

        <textarea

          rows="4"

          value={this.state.content}

          onChange={(event) **=>** this.setState({ content: event.target.value })}

        />

        <label>Author</label>

        <select

          value={this.state.author}

          onChange={(event) **=>** this.setState({ author: event.target.value })}

        >

          <option value="Max">Max</option>

          <option value="Manu">Manu</option>

        </select>

        <button onClick={this.postDataHandler}>Add Post</button>

      </div>

    );

  }

}

**Sending a DELETE request**

****

**Handling Error Locally**

* Sending a request không phải lúc nào cũng success 🡪 cần phải handle error
  + Nếu sử dụng Promise, chúng ta có thể catch error.
  + Nếu dùng async / await chúng ta sẽ dùng try / catch và display một thông báo nếu có lỗi xảy ra:

**src\containers\Blog\Blog.js**

**class** Blog **extends** Component {

  state = {

    posts: [],

    selectedPostId: null,

    error: false

  }

  postSelectedHandler = id **=>** {

    . . . .

  }

**async** componentDidMount() {

*// https://jsonplaceholder.typicode.com/posts*

    try {

**const** { data } = await axios.get('https://jsonplaceholder.typicode.com/postss'); *// throw error in here*

**const** posts = data.slice(0, 4);

**const** updatedPosts = posts.map(post **=>** {

        return {

          ...post,

          author: 'Xuân Hiếu' *// tạm thời hardcode giá trị của author*

        }

      });

      this.setState({

        posts: updatedPosts,

        error: false

      });

    } catch(e) {

      this.setState({

        error: true

      });

    }

  }

  render() {

**let** posts;

    if (this.state.error) {

      posts = <p>Something went wrong!!!</p>

    } else {

      posts = this.state.posts.map(post **=>** {

        return (

          <Post

            title={post.title}

            userId={post.author}

            key={post.id}

            clicked={() **=>** this.postSelectedHandler(post.id)}

          />

        );

      });

    };

    return (

      . . . . .

    );

  }

}

**Adding Interceptors to Execute Code Globally**

**src\index.js**

import React from 'react';

import ReactDOM from 'react-dom';

import './index.css';

import App from './App';

import registerServiceWorker from './registerServiceWorker';

import axios from 'axios';

axios.interceptors.request.use(request **=>** {

  console.log(request);

*// Edit request config if needed*

  return request;

}, err **=>** {

  console.log(err);

  return Promise.reject(err);

});

axios.interceptors.response.use(response **=>** {

  console.log(response);

  return response;

}, err **=>** {

  console.log(err);

  return Promise.reject(err);

});

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

registerServiceWorker();

**Setting a Default Global Configuration for Axios**

**src\index.js**

axios.defaults.baseURL = 'https://jsonplaceholder.typicode.com';

**Creating and Using Axios Instances**

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

**Section 10: Burger Builder Project: Accessing a Server**

**Creating the Axios Instance**

**src\axios-orders.js**

import axios from 'axios';

const instance = axios.create({

  baseURL: 'https://react-burger-builder-64bad-default-rtdb.firebaseio.com/'

});

export default instance;

**Sending a POST request**

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

  purchaseContinueHandler = async () => {

    // alert('You continue!!!');

    const order = {

      ingredients: this.state.ingredients,

      price: this.state.totalPrice,

      customer: {

        name: 'xhieu2206',

        address: {

          street: 'Hanoi',

          zipCode: '100000',

          country: 'Vietnam'

        },

        email: 'xhieu04@gmail.com'

      },

      deliveryMethod: 'fastest'

    }

    try {

      const res = await axios.post('/order.json', order);

      console.log(res);

    } catch(err) {

      console.log(err);

    }

  }

**Displaying a Spinner while Sending a Request**

* Hiển thị spinner trong Modal thay cho Order Summary, và sau khi việc sending data hoàn tất, Modal should be closed
  + [**https://projects.lukehaas.me/css-loaders/**](https://projects.lukehaas.me/css-loaders/)để lấy spinner css cần dùng.
  + Phải thay đổi Modal component để lofecycle shouldComponentUpdate vẫn trigger khi mà props.children thay đổi, vì chúng ta đặt Spinner component bên trong Modal component

**src\components\UI\Modal\Modal.js**

  shouldComponentUpdate(nextProps, nextState) {

    return nextProps.show !== this.props.show || nextProps.children !== this.props.children;

  }

* + Thêm state để quản lý việc hiển thị của Spinner component

**src\components\UI\Spinner\Spinner.js**

import React from 'react';

import classes from './Spinner.css';

const spinner = \_ => (

  <div className={classes.Loader}>Loading ...</div>

)

export default spinner;

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

****

  purchaseContinueHandler = () => {

    this.setState({

      loading: true

    });

    const order = {

      ingredients: this.state.ingredients,

      price: this.state.totalPrice,

      customer: {

        name: 'xhieu2206',

        address: {

          street: 'Hanoi',

          zipCode: '100000',

          country: 'Vietnam'

        },

        email: 'xhieu04@gmail.com'

      },

      deliveryMethod: 'fastest'

    }

    axios.post('/order.json', order)

      .then(\_ => {

        this.setState({

          loading: false,

          purchasing: false

        });

      })

      .catch(\_ => {

        this.setState({

          loading: false,

          purchasing: false

        })

      });

  }

  render() {

    const disabledInfo = {

      ...this.state.ingredients

    }

    for (let key in disabledInfo) {

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

    }

    let orderSummary = <OrderSummary

      ingredients={this.state.ingredients}

      purchaseCancelled={this.purchaseCancelHandler}

      purchaseContinued={this.purchaseContinueHandler}

      price={this.state.totalPrice}

    />

    if (this.state.loading) {

      orderSummary = <Spinner />;

    }

    return(

      <Aux>

        <Modal

          show={this.state.purchasing}

          modalClosed={this.purchaseCancelHandler}

        >

          {orderSummary}

        </Modal>

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

        <BuildControls

          . . . .

        />

      </Aux>

    );

  }

**Handling Errors**

* Chúng ta sẽ tạo ra một global error handler sẽ hiển thị một modal khi có errors, HOC sẽ là component mà chúng ta muốn sử dụng
  + Component này sẽ wrapped lấy BurgerBuilder và chúng ta sẽ dùng component ở bất cứ components nào có thể xuất hiện lỗi đến từ việc sending HTTP request

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

export default withErrorHandler(BurgerBuilder, axios); // BurgerBuilder is WrappedComponent

* + Tạo ra HOC **withErrorHandler** để handle error:

**src\hoc\withErrorHandler\withErrorHandler.js**

import React from 'react';

import Modal from '../../components/UI/Modal/Modal';

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

const withErrorHandler = (WrappedComponent, axios) => {

  return class extends React.Component {

    state = {

      error: null

    }

    componentDidMount() {

      axios.interceptors.request.use(request => { // nếu không có lỗi thì set lại giá trị của error = null

        this.setState({

          error: null

        });

        return request;

      });

      axios.interceptors.response.use(res => res, error => {

        this.setState({

          error: error

        });

      });

    }

    errorConfirmedHandler = () => {

      this.setState({

        error: null

      });

    }

    render() {

      return (

        <Aux>

          <Modal

            show={this.state.error}

            modalClosed={this.errorConfirmedHandler}

          >

            {this.state.error ? this.state.error.message : null}

          </Modal>

          <WrappedComponent {...this.props} />

        </Aux>

      )

    }

  }

}

export default withErrorHandler;

**Retrieving Data from the Backend**

* Thay vì hardcode các ingredient, chúng ta sẽ get các ingredients từ backend thông qua API.
  + Sử dụng lifecycle **componentDidMount** vì chúng ta muốn load các ingredients ngay từ khi **BurgerBuilder** được render lần đầu tiên
  + Cần handler các component render sử dụng ingredients từ state, chỉ render khi mà ingredients từ state khác null

  async componentDidMount () {

    const { data } = await axios.get('https://react-burger-builder-64bad-default-rtdb.firebaseio.com/ingredients');

    const ingredients = {...data};

    this.setState({

      ingredients: ingredients

    });

  }

  render() {

    const disabledInfo = {

      ...this.state.ingredients

    }

    for (let key in disabledInfo) {

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

    }

    let orderSummary = null; // init null cho orderSummary

    let burger = <Spinner />; // init với spinner trong thời gian loading ingredients từ backend

    if (this.state.ingredients) { // nếu có ingredients mới render burger và orderSummary

      burger = (

        <Aux>

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

          <BuildControls

            ingredientAdded={this.addIngredientHandler}

            ingredientRemoved={this.removeIngredientHandler}

            disabled={disabledInfo}

            purchasable={this.state.purchasable}

            price={this.state.totalPrice}

            ordered={this.purchaseHandler}

          />

        </Aux>

      )

      orderSummary = <OrderSummary

        ingredients={this.state.ingredients}

        purchaseCancelled={this.purchaseCancelHandler}

        purchaseContinued={this.purchaseContinueHandler}

        price={this.state.totalPrice}

      />

    }

    if (this.state.loading) {

      orderSummary = <Spinner />;

    }

    return(

      <Aux>

        <Modal

          show={this.state.purchasing}

          modalClosed={this.purchaseCancelHandler}

        >

          {orderSummary}

        </Modal>

        {burger}

      </Aux>

    );

  }

* **withErrorHandler** component sẽ không hoạt động do chúng ta setting up interceptors bên trong **componentDidMount** ở HOC và nó worked ở POST request, **componentDidMount** được completed bên trong các child component, và child component của chúng ta trong trường hợp này là **BurgerBuilder.** 
  + Thay vì sử dụng componentDidMount, chúng ta sẽ dùng componentWillMount (không khuyến khích) hoặc bên trong constructor của HOC
  + Xem lại chapter 184



**Removing old interceptors**

* Sử dụng interceptors như chúng ta đang làm sẽ có một vài issue như sau:
  + **withErrorHandler** có thể wrap bất cứ component nào có thể gây lỗi
  + Issue sẽ xuất hiện nếu như sử dụng component này để handle error cho cả những component khác, *componentWillMount* sẽ được gọi lại liên tục, cũng như sẽ có rất nhiều **withErrorHandler** trong application của chúng ta. 🡺 rất nhiều component không được sử dụng những vẫn sẽ tồn tại. 🡺 *componentWillUnmount* came to rescure

**src\hoc\withErrorHandler\withErrorHandler.js**

return **class** **extends** React.Component {

    state = {

      error: null

    }

    componentWillMount() {

      this.reqInterceptor = axios.interceptors.request.use(request **=>** { *// nếu không có lỗi thì set lại giá trị của error = null*

        this.setState({

          error: null

        });

        return request;

      });

      this.resInterceptor = axios.interceptors.response.use(res **=>** res, error **=>** {

        this.setState({

          error: error

        });

      });

    }

    componentWillUnmount() {

      axios.interceptors.request.eject(this.reqInterceptor);

      axios.interceptors.response.eject(this.resInterceptor);

    }

    errorConfirmedHandler = () **=>** {

      this.setState({

        error: null

      });

    }

    render() {

      return (

        <Aux>

          <Modal

            show={this.state.error}

            modalClosed={this.errorConfirmedHandler}

          >

            {this.state.error ? this.state.error.message : null}

          </Modal>

          <WrappedComponent {...this.props} />

        </Aux>

      )

    }

  }

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

**Section 11: Multi-Page\_felling in a SPA: Routing**

* Routing là khả năng show ra nhiều pages khác nhau cho user

****

**Setting up Links**

**src\containers\Blog\Blog.js**

    return (

      <div className="Blog">

        <header>

          <nav>

            <ul>

              <li><a href="/">Home</a></li>

              <li><a href="/new-post">New Post</a></li>

            </ul>

          </nav>

        </header>

        <section className="Posts">

          {posts}

        </section>

        <section>

          <FullPost

            id={this.state.selectedPostId}

          />

        </section>

        <section>

          <NewPost />

        </section>

      </div>

    );

**src\containers\Blog\Blog.css**

.Blog ul {

    list-style: none;

    margin: 0;

    padding: 0;

    width: 100%;

    text-align: center;

}

.Blog li {

    display: inline-block;

    margin: 20px;

}

.Blog a {

    text-decoration: none;

    color: black;

}

.Blog a:hover,

.Blog a:active {

    color: #FA923F;

}

****

**Setting up the Router Package**

* Install package:
  + **npm I –save react-router react-router-dom**
* Enable routing in app
  + Có thể thực hiện trong App.js hoặc index.js
  + Wrap the part of app should be able to render routes and to read route

**src\App.js**

**class** App **extends** Component {

  render() {

    return (

      <BrowserRouter>

        {*/\* bây giờ chúng ta có thể sử dụng routing features ở bất cứ đâu bên trong BrowserRouter này. Các sub-components cũng vậy \*/*}

        <div className="App">

          <Blog />

        </div>

      </BrowserRouter>

    );

  }

}

* + Đồng thời chúng ta sẽ re-structure lại các folder, bây giờ FullPost và NewPost cũng sẽ được coi như là cáccontainer component



**src\containers\Blog\Posts\Posts.js**

import React from 'react';

import axios from 'axios';

import './Posts.css'; *// chúng ta không dùng CSS module trong app này*

import Post from '../../../components/Post/Post';

**class** Posts **extends** React.Component {

  state = {

    posts: []

  }

**async** componentDidMount() {

*// https://jsonplaceholder.typicode.com/posts*

    try {

**const** { data } = await axios.get('/posts');

**const** posts = data.slice(0, 4);

**const** updatedPosts = posts.map(post **=>** {

        return {

          ...post,

          author: 'Xuân Hiếu' *// tạm thời hardcode giá trị của author*

        }

      });

      this.setState({

        posts: updatedPosts

      });

    } catch(e) {

      console.log(e);

    }

  }

  postSelectedHandler = id **=>** {

    this.setState({

      selectedPostId: id

    });

  }

  render() {

**let** posts;

    if (this.state.error) {

      posts = <p>Something went wrong!!!</p>

    } else {

      posts = this.state.posts.map(post **=>** {

        return (

          <Post

            title={post.title}

            userId={post.author}

            key={post.id}

            clicked={() **=>** this.postSelectedHandler(post.id)}

          />

        );

      });

    };

    return(

      <section className="Posts">

        {posts}

      </section>

    )

  }

}

export default Posts;

* + **Blog** component của chúng ta bây giờ sẽ chỉ như thế này

**src\containers\Blog\Blog.js**

**class** Blog **extends** Component {

  render() {

    return (

      <div className="Blog">

        <header>

          <nav>

            <ul>

              <li><a href="/">Home</a></li>

              <li><a href="/new-post">New Post</a></li>

            </ul>

          </nav>

        </header>

        <Posts />

      </div>

    );

  }

}

**Setting Up and Rerendering Routes + Switching Between Pages**

        <Route

          path="/" *// tell react về path mà chúng ta muốn apply JSX ở render*

          exact *// xác nhận đây path phải exact với path attribute.*

          render={() **=>** <h1>Home</h1>} *// 1 function trả về JSX*

        />

**src\containers\Blog\Blog.js**

**class** Blog **extends** Component {

  render() {

    return (

      <div className="Blog">

        <header>

          <nav>

            <ul>

              <li><a href="/">Home</a></li>

              <li><a href="/new-post">New Post</a></li>

            </ul>

          </nav>

        </header>

        <Route

          path="/" *// tell react về path mà chúng ta muốn apply JSX ở render*

          exact *// xác nhận đây path phải exact với path attribute.*

          component={Posts} *// 1 reference đến function hoặc class mà chúng ta muốn render*

        />

        <Route

          path="/new-post" *// tell react về path mà chúng ta muốn apply JSX ở render*

          exact *// xác nhận đây path phải exact với path attribute.*

          component={NewPost} *// 1 reference đến function hoặc class mà chúng ta muốn render*

        />

      </div>

    );

  }

}

****

**Using Links to Switch Pages**

* Chúng ta cần prevent việc page bị reload mỗi khi bấm vào links

**src\containers\Blog\Blog.js**

**class** Blog **extends** Component {

  render() {

    return (

      <div className="Blog">

        <header>

          <nav>

            <ul>

              <li><Link to="/">Home</Link></li>

              <li><Link to={{

                pathname: '/new-post',

                hash: '#submit',

                search: '?quick-submit=true'

              }}>New Post</Link></li>

            </ul>

          </nav>

        </header>

        <Route

          path="/" *// tell react về path mà chúng ta muốn apply JSX ở render*

          exact *// xác nhận đây path phải exact với path attribute.*

          component={Posts} *// 1 reference đến function hoặc class mà chúng ta muốn render*

        />

        <Route

          path="/new-post" *// tell react về path mà chúng ta muốn apply JSX ở render*

          exact *// xác nhận đây path phải exact với path attribute.*

          component={NewPost} *// 1 reference đến function hoặc class mà chúng ta muốn render*

        />

      </div>

    );

  }

}

****

**Using Routing – Related Props**

****

**The “withRouter” HOC and Route Props**

* Các routing related props không được passing đến các sub- component trong component tree.
  + Chúng ta không access được các properties ở trên trong **Post** component
  + Để pass các attributes này vào props của child component có 2 cách:
    - Cách 1: thủ công

****

* + - Cách 2: sử dụng HOC **withRouter** được cung cấp bởi *react-route-dom*

**src\components\Post\Post.js**

import React from "react";

import { withRouter } from 'react-router-dom';

import "./Post.css";

**const** post = (props) **=>** {

  console.log(props) // sẽ có các attribute liên quan đến route

  return (

    <article className="Post" onClick={props.clicked}>

      <h1>{props.title}</h1>

      <div className="Info">

        <div className="Author">{props.userId}</div>

      </div>

    </article>

  )

};

export default withRouter(post);

**Absolute vs Relative Paths**

* Giả sử chúng ta muốn navigate đến route mới có một phần được append vào route cũ:

****

**Styling the Active Route**

* Trong trường hợp đối với các Link đang active, chúng ta muốn style các link này
  + Thay component **Link** bằng **NavLink**
  + Bây giờ các link được generate bởi React sẽ có thêm class **.active** tương ứng
  + Style cho class này tương ứng
    - Có thể styling thông qua CSS file hoặc dùng props **activeStyle** của NavLink, cách dùng tương tự inline-style
  + Để thay đổi tên class tương ứng vởi link đang được active

**src\containers\Blog\Blog.js**

              <li>

                <NavLink

                to="/"

                activeClassName="my-active"

                activeStyle={{

                  color: '#FA923F',

                  textDecoration: 'underline'

                }}

                exact>

                  Home

                </NavLink>

              </li>

              <li>

                <NavLink

                activeClassName="active"

                activeStyle={{

                  textDecoration: 'underline'

                }}

                to={{

                  pathname: '/new-post',

                  hash: '#submit',

                  search: '?quick-submit=true'

                }}>

                  New Post

                </NavLink>

              </li>

****

**Passing Route Params**

* Bây giờ khi click vào post, chúng ta muốn load component FullPost
  + Cần phải get dynamic routing vì mỗi một Post đều có một ID
  + Đồng thời wrap các Post component với **Link** component cung cấp bởi react-route
  + Thay đổi lại **key** attribute bên thích hợp.

**src\containers\Blog\Blog.js**

        <Route

          path='/:id'

          exact

          component={FullPost}

        >

**src\containers\Blog\Posts\Posts.js**

  render() {

**let** posts;

    if (this.state.error) {

      posts = <p>Something went wrong!!!</p>

    } else {

      posts = this.state.posts.map(post **=>** {

        return (

          <Link

          to={'/' + post.id}

          key={post.id}>

            <Post

              title={post.title}

              userId={post.author}

              clicked={() **=>** this.postSelectedHandler(post.id)}

            />

          </Link>

        );

      });

    };

* Extract route parameter ở **FullPost** component
  + Thay **componentDidUpdate** thành **componentDidMount** vì bây giờ chúng ta không update component nữa mà sẽ mount lại component

**src\containers\Blog\FullPost\FullPost.js**

**async** componentDidMount() {

**const** postId = this.props.match.params.id;

*// https://jsonplaceholder.typicode.com/posts/1*

    if (postId) { *// không cần check gì vì component được mount lại chứ không phải get updated*

**const** { data } = await axios.get(`/posts/${postId}`);

      this.setState({

        title: data.title,

        body: data.body,

        id: data.id

      });

    }

  }

**Parsing Query Parameters & the Fragment**

**Chapter 204:** [**Parsing query parameters & the Fragment**](https://www.udemy.com/course/react-the-complete-guide-incl-redux/learn/lecture/8140669#content)

**Using Switch to Load a Single Route**

****

* Nếu setup route như trên, sẽ có lúc react app sẽ coi “new-post” là id và treat nó như là một id của post, và kết quả sẽ như trên
  + Chúng ta có thể fix bằng cách chuyển route của render component **FullPost** bằng “posts/:id”
  + Sử dụng **Switch** component của react-router
    - **Switch** sẽ nói với react rằng “hãy chỉ load duy nhất một route ứng với route đầu tiên matched với các routes đã setup”
  + Order rất quan trọng trong việc setup route, nếu chúng ta đặt route của “new-post” trước route cho **FullPost**, react sẽ không bao giờ matched được route cho **NewPost**

**src\containers\Blog\Blog.js**

        <Switch>

          <Route

            path="/" *// tell react về path mà chúng ta muốn apply JSX ở render*

            exact *// xác nhận đây path phải exact với path attribute.*

            component={Posts} *// 1 reference đến function hoặc class mà chúng ta muốn render*

          />

          <Route

            path="/new-post" *// tell react về path mà chúng ta muốn apply JSX ở render*

            component={NewPost} *// 1 reference đến function hoặc class mà chúng ta muốn render*

          />

          <Route path='/:id' exact component={FullPost} />

        </Switch>

**Navigating Programmatically**

* Giả sử trong trường hợp chúng ta muốn navigate progreammatically, khi người dùng bấm vào Post và navigate đến FullPost page mà không cần dùng đến **Link** component, ta có thể làm như sau:
  + Remove **Link** component
  + Re-use lại method *postSelectedHandler*
  + Sử dụng history object mà chúng ta received từ props

**src\containers\Blog\Posts\Posts.js**

  postSelectedHandler = id **=>** {

    this.props.history.push({

      pathname: '/' + id

    });

  }

  render() {

**let** posts;

    if (this.state.error) {

      posts = <p>Something went wrong!!!</p>

    } else {

      posts = this.state.posts.map(post **=>** {

        return (

          <Post

            title={post.title}

            userId={post.author}

            clicked={() **=>** this.postSelectedHandler(post.id)}

            key={post.id}

          />

        );

      });

    };

    return(

      <section className="Posts">

        {posts}

      </section>

    )

  }

**Additional Information Regarding Active Links**

**Understanding the Nested Routes**

* Thay vì hiển thị FullPost ra 1 page riêng, chúng ta muốn hiện thị FullPost ở dưới trang Posts (Home page) môi khi click vào a single post
  + Phải bỏ ***exact*** khi setup route cho “/posts”, nếu không chúng ta sẽ không bao giờ react được route “/”, mà chúng ta lại render **FullPost** bên trong **Posts** component, nên chúng ta không bao giờ có thể render được **FullPost** component.

**src\containers\Blog\Blog.js**

          <Route

            path="/posts" *// tell react về path mà chúng ta muốn apply JSX ở render*

            component={Posts} *// 1 reference đến function hoặc class mà chúng ta muốn render*

          />

**src\containers\Blog\Posts\Posts.js**

  postSelectedHandler = id **=>** {

    this.props.history.push({

      pathname: '/posts/' + id

    });

  }

    return(

      <div>

        <section className="Posts">

          {posts}

        </section>

        <Route path={this.props.match.url + '/:id'} exact component={FullPost} />

      </div>

    )



**Creating Dynamic nested routes**

* Hiện tại nếu chúng ta đã access một **FullPost** page, khi click sang các post khác, **FullPost** component không được updated với Post mà chúng ta click vào dù trên URL giá trị id có thay đổi.
  + BTS (behind the scenes) react router không replace component all the time
  + Chúng ta có thể fix bằng cách thêm **componentDidUpdate** method
  + Bây giờ chúng ta phải thêm check điều kiện để tránh infinite loop khi get updated component
  + Chú ý rằng ID chúng ta lấy từ route param sẽ là String, do đó cần parseInt trước khi so sánh

**src\containers\Blog\FullPost\FullPost.js**

**async** componentDidMount() {

    await this.loadData();

  }

**async** componentDidUpdate() {

    await this.loadData();

  }

**async** loadData() {

**const** postId = this.props.match.params.id;

*// https://jsonplaceholder.typicode.com/posts/1*

    if (postId && this.state.id !== parseInt(postId)) { *// check nếu param id trên route thay đổi để avoid infinite loop*

**const** { data } = await axios.get(`/posts/${postId}`);

      this.setState({

        title: data.title,

        body: data.body,

        id: data.id

      });

    }

  }

**Redirecting Requests**

* Giả sử chúng ta muốn redirecting user sang “/posts” khi user visit “/”
  + Duplicate Route component với “/” 🡪 Posts component
  + Sử dụng **Redirect** component

**src\containers\Blog\Blog.js**

          <Redirect from="/" to="/posts" />

**Conditional Redirects**

* Giả sử sau khi tạo một post mới, chúng ta muốn redirect user về Posts page

**src\containers\Blog\NewPost\NewPost.js**

state = {

    title: "",

    content: "",

    author: "Xuân Hiếu",

    submitted: false

  };

postDataHandler = **async** () **=>** {

*// POST https://jsonplaceholder.typicode.com/posts*

**const** post = {

      title: this.state.title,

      body: this.state.content,

      author: this.state.author

    }

**const** {data} = await axios.post('/posts', post);

    this.setState({

      submitted: true

    });

    console.log(data);

  };

render() {

**let** redirect = null;

    if (this.state.submitted) {

      redirect = <Redirect to="/posts" />

    }

    return (

      <div className="NewPost">

        {redirect}

        . . .

      </div>

    );

  }

**Using the History Props to Redirect (Replace)**

**src\containers\Blog\NewPost\NewPost.js**

  postDataHandler = **async** () **=>** {

*// POST https://jsonplaceholder.typicode.com/posts*

**const** post = {

      title: this.state.title,

      body: this.state.content,

      author: this.state.author

    }

**const** {data} = await axios.post('/posts', post);

*// this.props.history.push('/posts');*

    this.props.history.replace('/posts');

    console.log(data);

  };

**Working with Guards**

* Ở các framewwork khác (Angular hoặc VueJS), Guards có thể coi như middleware kiểm tra một vài điều kiện trước khi cho phép user truy cập vào một routes cụ thể nào đó
* Tuy nhiên ở React, giả sử chúng ta muốn chặn người dùng vào trang New Post, chúng ta chỉ đơn giản render Route cho New Post conditionally
* 🡺 Không có guards trong React, mọi thứ đều do Devs quyết định.

**src\containers\Blog\Blog.js**

          {this.state.auth ? <Route

            path="/new-post" *// tell react về path mà chúng ta muốn apply JSX ở render*

            component={NewPost} *// 1 reference đến function hoặc class mà chúng ta muốn render*

          /> : null}

**Handling the 404 Case (Unknown Routes)**

* Cách này sẽ không có tác dụng nếu chúng ta vẫn sử dụng **Redirect** component như trước chúng ta đã dùng

**src\containers\Blog\Blog.js**

        <Switch>

          {this.state.auth ? <Route

            path="/new-post" *// tell react về path mà chúng ta muốn apply JSX ở render*

            component={NewPost} *// 1 reference đến function hoặc class mà chúng ta muốn render*

          /> : null}

          <Route

            path="/posts" *// tell react về path mà chúng ta muốn apply JSX ở render*

            component={Posts} *// 1 reference đến function hoặc class mà chúng ta muốn render*

          />

          <Route render={() **=>** <h1>Not Found Page</h1>} />

          {*/\* <Redirect from="/" to="/posts" /> \*/*}

        </Switch>

**Loading Routes Lazily**

* Worked with react-router 4 và create-react-app vì phụ thuộc nhiều vào Webpack

**src\hoc\asyncComponent.js**

import React from 'react';

**const** asyncComponent = (importComponent) **=>** {

  return **class** **extends** React.Component {

    state = {

      component: null

    }

    componentDidMount() {

      importComponent()

        .then(cmp **=>** {

          this.setState({

            component: cmp.default

          });

        });

    }

    render() {

**const** C = this.state.component;

      return C ? <C {...this.props} /> :null;

    }

  }

}

export default asyncComponent;

**src\containers\Blog\Blog.js**

*// import NewPost from './NewPost/NewPost';*

import asyncComponent from '../../hoc/asyncComponent';

**const** AsyncNewPost = asyncComponent(() **=>** {

  return import('./NewPost/NewPost');

});

          {this.state.auth ? <Route

            path="/new-post" *// tell react về path mà chúng ta muốn apply JSX ở render*

            component={AsyncNewPost} *// async import component*

          /> : null}

**Lazy Loading with React Suspense (16.6) (Tham khảo)**

* Đây là một cách khác để lazy loading routes có thể áp dụng từ phiên bản 16.6 của react: [bài 216](https://www.udemy.com/course/react-the-complete-guide-incl-redux/learn/lecture/12296826#content)

**Routing and Server Deployment**

****

****

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

**Section 12: Adding Routing to Burger Project**

* Chúng ta sẽ adding một pagr để cho phépcheckout burger
* Sẽ có một form checkout để adding some information

**Building the Checkout container**

* Chúng ta sẽ một container component mới với những state của riêng nó.
  + Checkout component sẽ là một summary về những thông tin về burger mà user đã đặt
  + Button để cancel và continue
  + Sau khi user bấm continue, load the contact form
  + Prepare:

**src\App.js**

**class** App **extends** Component {

  render() {

    return (

      <div>

        <Layout>

          <BurgerBuilder />

          <Checkout />

        </Layout>

      </div>

    );

  }

}

**src\containers\Checkout\Checkout.js**

**class** Checkout **extends** React.Component {

  state = { *// tạm thời hardcode*

    ingredients: {

      salad: 1,

      meat: 1,

      cheese: 1,

      bacon: 1

    }

  }

  render() {

    return (

      <div>

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

      </div>

    )

  }

}

**src\components\Order\CheckoutSummary\CheckoutSummary.js**

**const** checkoutSummary = props **=>** {

  return (

    <div className={classes.CheckoutSummary}>

      <h1>We hope it tastes well!!!</h1>

      <div style={{

        width: '100%',

        margin: 'auto'

      }}>

        <Burger ingredients={props.ingredients} />

      </div>

      <Button btnType="Danger" clicked={() **=>** {}}>Cancel</Button>

      <Button btnType="Success" clicked={() **=>** {}}>Continue</Button>

    </div>

  )

}

**Setting Up Routing & Routes**

* **npm i --save react-router-dom**

**src\index.js**

import { BrowserRouter } from 'react-router-dom';

**const** app = (

  <BrowserRouter>

    <App />

  </BrowserRouter>

)

ReactDOM.render(app, document.getElementById('root'));

registerServiceWorker();

**src\App.js**

**class** App **extends** Component {

  render() {

    return (

      <div>

        <Layout>

          <Switch>

            <Route path="/checkout" component={Checkout} />

            <Route path="/" component={BurgerBuilder} />

          </Switch>

        </Layout>

      </div>

    );

  }

}

**Navigating to the Checkout page**

* Khi click on **Continue** ở modal, chúng ta muốn navigate sang checkout page.

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

  purchaseContinueHandler = () **=>** { *// tạm commnet-out code cũ vì chúng ta sẽ không handle việc sending order ở đây*

*// this.setState({*

*//   loading: true*

*// });*

*// const order = {*

*//   ingredients: this.state.ingredients,*

*//   price: this.state.totalPrice,*

*//   customer: {*

*//     name: 'xhieu2206',*

*//     address: {*

*//       street: 'Hanoi',*

*//       zipCode: '100000',*

*//       country: 'Vietnam'*

*//     },*

*//     email: 'xhieu04@gmail.com'*

*//   },*

*//   deliveryMethod: 'fastest'*

*// }*

*// axios.post('/orders.json', order)*

*//   .then(\_ => {*

*//     this.setState({*

*//       loading: false,*

*//       purchasing: false*

*//     });*

*//   })*

*//   .catch(\_ => {*

*//     this.setState({*

*//       loading: false,*

*//       purchasing: false*

*//     })*

*//   });*

    this.props.history.push({

      pathname: '/checkout'

    })

  }

**Navigating Back & To Next Page**

**src\containers\Checkout\Checkout.js**

  checkoutCancelledHandler = () **=>** {

    this.props.history.goBack(); *// go back to BurgerBuilder*

  }

  checkoutContinuedHandler = () **=>** {

    this.props.history.replace('/checkout/contact-data'); *// go to contact form to fill-out some information*

*// TODO later*

  }

  render() {

    return (

      <div>

        <CheckoutSummary

          ingredients={this.state.ingredients}

          checkoutCancelled={this.checkoutCancelledHandler}

          checkoutContinued={this.checkoutContinuedHandler}

        />

      </div>

    )

  }

**src\components\Order\CheckoutSummary\CheckoutSummary.js**

**const** checkoutSummary = props **=>** {

  return (

    <div className={classes.CheckoutSummary}>

      <h1>We hope it tastes well!!!</h1>

      <div style={{

        width: '100%',

        margin: 'auto'

      }}>

        <Burger ingredients={props.ingredients} />

      </div>

      <Button btnType="Danger" clicked={props.checkoutCancelled}>Cancel</Button>

      <Button btnType="Success" clicked={props.checkoutContinued}>Continue</Button>

    </div>

  )

}

**Passing Ingredients via Query Params**

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

  purchaseContinueHandler = () **=>** { *// tạm commnet-out code cũ vì chúng ta sẽ không handle việc sending order ở đây*

**let** queryParams = '?';

    for (**const** key in this.state.ingredients) {

      queryParams = queryParams.concat(`${key}=${this.state.ingredients[key]}&`);

    }

    this.props.history.push({

      pathname: '/checkout',

      search: queryParams.slice(0, queryParams.length - 1)

    });

  }

**src\containers\Checkout\Checkout.js**

  componentDidMount() {

**const** query = new URLSearchParams(this.props.location.search)

**const** ingredients = {};

    for (**const** param of query.entries()) {

      ingredients[param[0]] = +param[1]

    }

    this.setState({

      ingredients: ingredients

    })

  }

**Navigating to the Contact Data Component**

* Component này sẽ là một container và sẽ có state của riêng nó
  + Chúng ta sẽ sử dụng nested route, hiển thị ContactData component inside Checkout component

**src\containers\Checkout\Checkout.js**

  render() {

    return (

      <div>

        <CheckoutSummary

          ingredients={this.state.ingredients}

          checkoutCancelled={this.checkoutCancelledHandler}

          checkoutContinued={this.checkoutContinuedHandler}

        />

        <Route

          path={this.props.match.path + '/contact-data'}

          component={ContactData}

        />

      </div>

    )

  }

**src\containers\Checkout\ContactData\ContactData.js**

**class** ContactData **extends** React.Component {

  state = {

    name: '',

    email: '',

    address: {

      street: '',

      postalCode: ''

    }

  }

  render() {

    return (

      <div className={classes.ContactData}>

        <h4>Enter your Contact please</h4>

        <form>

          <input className={classes.Input} type="text" name="name" placeholder="Your Name" />

          <input className={classes.Input} type="email" name="email" placeholder="Your Email" />

          <input className={classes.Input} type="street" name="street" placeholder="Street" />

          <input className={classes.Input} type="postal" name="postal" placeholder="Postal Code" />

          <Button btnType="Success">ORDER</Button>

        </form>

      </div>

    )

  }

}

**Order Submission & Passing Data between Pages**

* Đây là một trick rất hay để passing data vào component được render bởi nested route.

**src\containers\Checkout\Checkout.js**

  render() {

    return (

      <div>

        <CheckoutSummary

          ingredients={this.state.ingredients}

          checkoutCancelled={this.checkoutCancelledHandler}

          checkoutContinued={this.checkoutContinuedHandler}

        />

        <Route

          path={this.props.match.path + '/contact-data'}

          render={() **=>** (<ContactData {...this.state} />)}

        />

      </div>

    )

  }

**src\containers\Checkout\ContactData\ContactData.js**

  orderHandler = (e) **=>** {

    e.preventDefault();

    console.log(this.props); *// we have ingredients here passed from Checkout component*

  }

* Bây giờ, chúng ta sẽ sending request từ ContactData component
  + Passed price từ **BurgerBuilder** đến ContactData component (tiếp tục dùng query param tại thời điểm này)
  + Thêm **Spinner** component trong lúc sending data ở **ContactData** component
  + Redirect về “/” sau khi đã submit thành công (sử dụng cách passing props)

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

  purchaseContinueHandler = () **=>** { *// tạm commnet-out code cũ vì chúng ta sẽ không handle việc sending order ở đây*

**let** queryParams = '?';

    for (**const** key in this.state.ingredients) {

      queryParams = queryParams.concat(`${key}=${this.state.ingredients[key]}&`);

    }

    this.props.history.push({

      pathname: '/checkout',

      search: queryParams + `price=${this.state.totalPrice}`

    });

  }

**src\containers\Checkout\Checkout.js**

**class** Checkout **extends** React.Component {

  state = { *// tạm thời hardcode*

    ingredients: null,

    totalPrice: 0

  }

  componentDidMount() {

**const** query = new URLSearchParams(this.props.location.search)

**const** ingredients = {};

**let** totalPrice;

    for (**const** param of query.entries()) {

      if (param[0] === 'price') {

        totalPrice = parseFloat(param[1])

      } else {

        ingredients[param[0]] = +param[1];

      }

    }

    this.setState({

      ingredients: ingredients,

      totalPrice: totalPrice

    })

  }

  checkoutCancelledHandler = () **=>** {

    this.props.history.goBack(); *// go back to BurgerBuilder*

  }

  checkoutContinuedHandler = () **=>** {

    this.props.history.replace('/checkout/contact-data'); *// go to contact form to fill-out some information*

  }

  render() {

    return (

      <div>

        {this.state.ingredients ? <CheckoutSummary

          ingredients={this.state.ingredients}

          checkoutCancelled={this.checkoutCancelledHandler}

          checkoutContinued={this.checkoutContinuedHandler}

        /> : <Spinner />} // check xem có ingredients không để hiển thị CheckoutSummary

        <Route

          path={this.props.match.path + '/contact-data'}

          render={(props) **=>** (<ContactData {...this.state} {...props} />)} // cách dưới cũng được

*// render={() => (<ContactData {...this.state} {...this.props} />)}*

        />

      </div>

    )

  }

}

**src\containers\Checkout\ContactData\ContactData.js**

**class** ContactData **extends** React.Component {

  state = {

    name: '',

    email: '',

    address: {

      street: '',

      postalCode: ''

    },

    loading: false

  }

  orderHandler = (e) **=>** {

    e.preventDefault();

    this.setState({

      loading: true

    });

**const** order = {

      ingredients: this.props.ingredients,

      price: this.props.totalPrice,

      customer: { *// tạm thời hardcode, handle khi học đến Form module*

        name: 'xhieu2206',

        address: {

          street: 'Hanoi',

          zipCode: '100000',

          country: 'Vietnam'

        },

        email: 'xhieu04@gmail.com'

      },

      deliveryMethod: 'fastest'

    }

    axios.post('/orders.json', order)

      .then(\_ **=>** {

        this.setState({

          loading: false

        });

        this.props.history.push({ *// redirect về "/" nếu submit thành công*

          pathname: '/'

        });

      })

      .catch(\_ **=>** {

        this.setState({

          loading: false

        });

      });

  }

  render() {

    return (

      <div className={classes.ContactData}>

        <h4>Enter your Contact please</h4>

        {!this.state.loading ? (<form>

          <input className={classes.Input} type="text" name="name" placeholder="Your Name" />

          <input className={classes.Input} type="email" name="email" placeholder="Your Email" />

          <input className={classes.Input} type="street" name="street" placeholder="Street" />

          <input className={classes.Input} type="postal" name="postal" placeholder="Postal Code" />

          <Button btnType="Success" clicked={this.orderHandler}>ORDER</Button>

        </form>) : <Spinner />}

      </div>

    )

  }

}

**Adding an Orders Page**

* Chúng ta sẽ cần một page để loading các orders từ backend
  + **Orders** component sẽ bao gồm nhiều **Order** component.

**src\App.js**

          <Switch>

            <Route path="/checkout" component={Checkout} />

            <Route path="/orders" component={Orders} />

            <Route path="/" component={BurgerBuilder} />

          </Switch>

**src\containers\Orders\Orders.js**

import Order from '../../components/Order/Order';

**class** Orders **extends** React.Component {

  render() {

    return (

      <div>

        <Order />

      </div>

    );

  }

}

export default Orders;

**Implementing Navigation Links**

**src\components\Navigation\NavigationItems\NavigationItems.js**

**const** navigationItems = () **=>** {

  return (

    <ul className={classes.NavigationItems}>

      <NavigationItem

        link="/"

        exact

      >

        Burger Builder

      </NavigationItem>

      <NavigationItem

        link="/orders"

      >

        Orders

      </NavigationItem>

    </ul>

  )

}

**src\components\Navigation\NavigationItems\NavigationItem\NavigationItem.js**

import { NavLink } from 'react-router-dom';

. . . . .

**const** navigationItem = props **=>** {

  return (

    <li className={classes.NavigationItem}>

      <NavLink

        exact={props.exact}

        to={props.link}

        activeClassName={classes.active}>

          {props.children}

      </NavLink>

    </li>

  )

}

**Fetching Orders**

**src\containers\Orders\Orders.js**

**class** Orders **extends** React.Component {

  state = {

    orders: [],

    loading: true

  }

  componentDidMount() {

    axios.get(`https://react-burger-builder-64bad-default-rtdb.firebaseio.com/orders.json`)

      .then(({data}) **=>** {

**const** orderArr = [];

        for (**const** key in data) {

          orderArr.push({

            ...data[key],

            id: key

          });

        }

        console.log(orderArr);

        this.setState({

          orders: [...orderArr],

          loading: false

        });

      })

      .catch(err **=>** {

        this.setState({

          loading: false

        })

      });

  }

  render() {

**let** orders = <Spinner />;

    if (this.state.orders.length > 0) {

      orders = this.state.orders.map(order **=>** {

        return (

          <Order

            key={order.id}

            price={order.price}

            ingredients={order.ingredients}

          />

        )

      });

    }

    return (

      <div>

        {orders}

      </div>

    );

  }

}

export default withErrorHandler(Orders, axios);

**src\components\Order\Order.js**

**const** order = props **=>** {

**const** ingredients = [];

  for (**let** igName in props.ingredients) {

    ingredients.push({name: igName, amount: props.ingredients[igName]})

  };

**const** ingredientsOutput = ingredients.map(ig **=>** {

    return <span

      style={{ textTransform: 'capitalize', display: 'inline-block', margin: '0 8px', border: '1px solid #ccc', padding: '2px' }}

      key={ig.name}>{ig.name} - ({ig.amount}) </span>

  });

  return (

    <div className={classes.Order}>

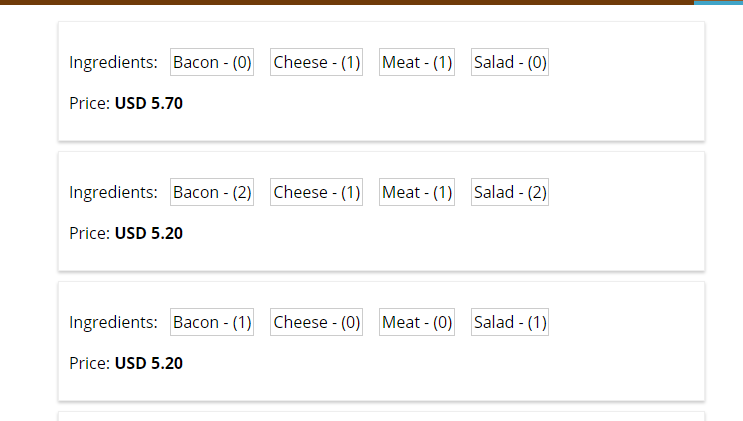
      <p>Ingredients: {ingredientsOutput}</p>

      <p>Price: <strong>USD {parseFloat(props.price).toFixed(2)}</strong></p>

    </div>

  )

};

****

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

**Section 13: Forms and Form Validation**

**Creating a Custom Dynamic for the Form**

* Tạo component cho việc render các Input
* Thay thế các input element bằng các customer

**src\components\UI\Input\Input.js**

const input = props => {

  let inputElement = null;

  switch (props.inputtype) {

    case ('input'):

      inputElement = <input className={classes.InputElement} {...props} />;

      break;

    case ('textarea'):

      inputElement = <textarea className={classes.InputElement} {...props} />;

      break;

    default:

      inputElement = <input className={classes.InputElement} {...props} />;

  }

  return (

    <div className={classes.Input}>

      <label>{props.placeholder}</label>

      {inputElement}

    </div>

  )

};

export default input;

**src\containers\Checkout\ContactData\ContactData.js**

  render() {

    return (

      <div className={classes.ContactData}>

        <h4>Enter your Contact please</h4>

        {!this.state.loading ? (<form>

          <Input inputype="input" type="text" name="name" placeholder="Your Name" />

          <Input inputype="input" type="email" name="email" placeholder="Your Email" />

          <Input inputype="input" type="street" name="street" placeholder="Street" />

          <Input inputype="input" type="postal" name="postal" placeholder="Postal Code" />

          <Button btnType="Success" clicked={this.orderHandler}>ORDER</Button>

        </form>) : <Spinner />}

      </div>

    )

  }

**Setting Up a JS Config for the Form + Dynamically Create Inputs based on JS Config + Adding a Dropdown Component**

* Chúng ta sẽ dynamically generate các Input component dựa vào state của ContactData

**src\containers\Checkout\ContactData\ContactData.js**

state = {

    orderForm: {

      name: {

        elementType: 'input',

        elementConfig: {

          type: 'text',

          placeholder: 'Your Name'

        },

        value: ''

      },

      email: {

        elementType: 'input',

        elementConfig: {

          type: 'email',

          placeholder: 'Your Email'

        },

        value: ''

      },

      country: {

        elementType: 'input',

        elementConfig: {

          type: 'text',

          placeholder: 'Your Country'

        },

        value: ''

      },

      street: {

        elementType: 'input',

        elementConfig: {

          type: 'text',

          placeholder: 'Street'

        },

        value: ''

      },

      zipCode: {

        elementType: 'input',

        elementConfig: {

          type: 'text',

          placeholder: 'ZIP CODE'

        },

        value: ''

      },

      deliveryMethod: {

        elementType: 'select',

        elementConfig: {

          options: [

            {value: 'fastest', displayValue: 'Fastest'},

            {value: 'cheapest', displayValue: 'Cheapest'},

          ],

          placeholder: 'Delivery Method:'

        },

        value: ''

      }

    },

    loading: false

  }

  render() {

    const formElementsArray = [];

    for (let key in this.state.orderForm) {

      formElementsArray.push({

        id: key,

        config: this.state.orderForm[key]

      });

    }

    return (

      <div className={classes.ContactData}>

        <h4>Enter your Contact please</h4>

        {!this.state.loading ? (

        <form>

          {formElementsArray.map(formElement => {

            return <Input

              key={formElement.id}

              elementType={formElement.config.elementType}

              elementConfig={formElement.config.elementConfig}

              value={formElement.config.value}

            />

          })}

          <Button btnType="Success" clicked={this.orderHandler}>ORDER</Button>

        </form>

        ) : <Spinner />}

      </div>

    )

  }

**src\components\UI\Input\Input.js**

const input = props => {

  let inputElement = null;

  switch (props.elementType) {

    case ('input'):

      inputElement = <input

        className={classes.InputElement}

        {...props.elementConfig}

        value={props.value} />;

      break;

    case ('textarea'):

      inputElement = <textarea

        className={classes.InputElement}

        {...props.elementConfig}

        value={props.value} />;

      break;

    case ('select'):

      inputElement = (

        <select

          className={classes.InputElement}

          value={props.value}

        >

          {props.elementConfig.options.map(option => {

            return <option

              key={option.value}

              value={option.value}>

                {option.displayValue}

              </option>

          })}

        </select>

      )

      break;

    default:

      inputElement = <input

        className={classes.InputElement}

        {...props.elementConfig}

        value={props.value} />;

  }

  return (

    <div className={classes.Input}>

      <label>{props.elementConfig.placeholder}</label>

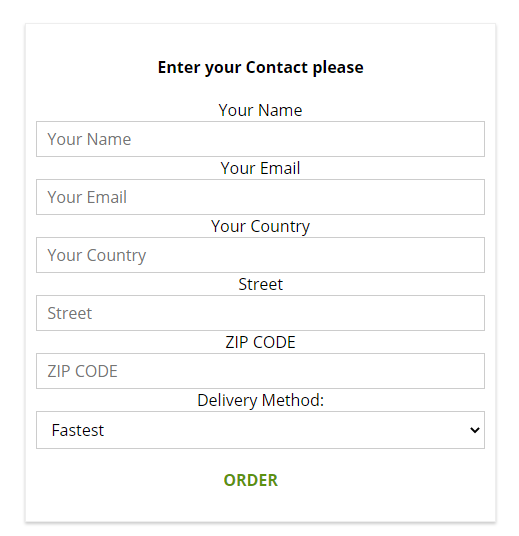
      {inputElement}

    </div>

  )

};

export default input;

****

**Handling User Input**

* Passing method để handle input changed trong **Input** component

**src\components\UI\Input\Input.js**

  switch (props.elementType) {

    case ('input'):

      inputElement = <input

        className={classes.InputElement}

        {...props.elementConfig}

        onChange={props.changed}

        value={props.value} />;

      break;

    case ('textarea'):

      inputElement = <textarea

        className={classes.InputElement}

        {...props.elementConfig}

        onChange={props.changed}

        value={props.value} />;

      break;

    case ('select'):

      inputElement = (

        <select

          className={classes.InputElement}

          value={props.value}

          onChange={props.changed}

        >

          {props.elementConfig.options.map(option => {

            return <option

              key={option.value}

              value={option.value}>

                {option.displayValue}

              </option>

          })}

        </select>

      )

      break;

    default:

      inputElement = <input

        className={classes.InputElement}

        {...props.elementConfig}

        onChange={props.changed}

        value={props.value} />;

  }

**src\containers\Checkout\ContactData\ContactData.js**

  changedInputHandler = (e, key) => {

    let orderForm = {...this.state.orderForm};

    orderForm[key].value = e.target.value;

    this.setState({

      orderForm: orderForm

    });

  }

  render() {

    const formElementsArray = [];

    for (let key in this.state.orderForm) {

      formElementsArray.push({

        id: key,

        config: this.state.orderForm[key]

      });

    }

    return (

      <div className={classes.ContactData}>

        <h4>Enter your Contact please</h4>

        {!this.state.loading ? (

        <form>

          {formElementsArray.map(formElement => {

            return <Input

              key={formElement.id}

              elementType={formElement.config.elementType}

              elementConfig={formElement.config.elementConfig}

              value={formElement.config.value}

              changed={(e) => this.changedInputHandler(e, formElement.id)}

            />

          })}

          <Button btnType="Success" clicked={this.orderHandler}>ORDER</Button>

        </form>

        ) : <Spinner />}

      </div>

    )

  }

**Handling form submission**

**src\containers\Checkout\ContactData\ContactData.js**

  orderHandler = (e) => {

    e.preventDefault();

    this.setState({

      loading: true

    });

    const formData = {};

    for (let formEle in this.state.orderForm) {

      formData[formEle] = this.state.orderForm[formEle].value;

    }

    const order = {

      ingredients: this.props.ingredients,

      price: this.props.totalPrice,

      orderData: formData

    }

    axios.post('/orders.json', order)

      .then(\_ => {

        this.setState({

          loading: false

        });

        this.props.history.push({ // redirect về "/" nếu submit thành công

          pathname: '/'

        });

      })

      .catch(\_ => {

        this.setState({

          loading: false

        });

      });

  }

        <form onSubmit={this.orderHandler} > // không cần handle click button, sử dụng event onSubmit

          {formElementsArray.map(formElement => {

            return <Input

              key={formElement.id}

              elementType={formElement.config.elementType}

              elementConfig={formElement.config.elementConfig}

              value={formElement.config.value}

              changed={(e) => this.inputChangedHandler(e, formElement.id)}

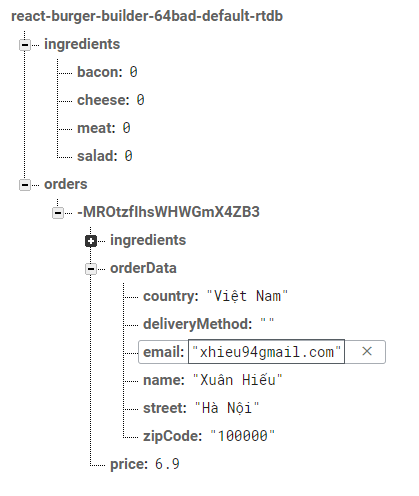
            />

          })}

          <Button btnType="Success" clicked={this.orderHandler}>ORDER</Button>

        </form>

        ) : <Spinner />}

****

**Adding Custom Form Validation**

* React không có những thư viện để thực hiện validation như Angular, chúng ta sẽ làm validation 1 cách thủ công.
* Mỗi khi input changed, chúng ta sẽ handle xem input này có valid hay không.

**src\containers\Checkout\ContactData\ContactData.js**

**class** ContactData **extends** React.Component {

  state = {

    orderForm: {

      name: {

        elementType: 'input',

        elementConfig: {

          type: 'text',

          placeholder: 'Your Name'

        },

        value: '',

        validation: {

          required: true

        },

        valid: false

      },

      email: {

        elementType: 'input',

        elementConfig: {

          type: 'email',

          placeholder: 'Your Email'

        },

        value: '',

        validation: {

          required: true

        },

        valid: false

      },

      country: {

        elementType: 'input',

        elementConfig: {

          type: 'text',

          placeholder: 'Your Country'

        },

        value: '',

        validation: {

          required: true

        },

        valid: false

      },

      street: {

        elementType: 'input',

        elementConfig: {

          type: 'text',

          placeholder: 'Street'

        },

        value: '',

        validation: {

          required: true

        },

        valid: false

      },

      zipCode: {

        elementType: 'input',

        elementConfig: {

          type: 'text',

          placeholder: 'ZIP CODE'

        },

        value: '',

        validation: {

          required: true,

          minLength: 5,

          maxLength: 5

        },

        valid: false

      },

      deliveryMethod: {

        elementType: 'select',

        elementConfig: {

          options: [

            { value: 'fastest', displayValue: 'Fastest' },

            { value: 'cheapest', displayValue: 'Cheapest' },

          ],

          placeholder: 'Delivery Method:'

        },

        value: ''

      }

    },

    loading: false

  }

  orderHandler = (e) **=>** {

    . . . .

  }

  checkValidity(value, rules) {

**let** isValid = true;

*// check for required field*

    if (rules.required) {

      isValid = value.trim() !== '' && isValid;

    }

*// check for min length*

    if (rules.minLength) {

      isValid = value.trim().length >= rules.minLength && isValid;

    }

*// check for max length*

    if (rules.maxLength) {

      isValid = value.trim().length <= rules.maxLength && isValid;

    }

    return isValid;

  }

  inputChangedHandler = (e, key) **=>** {

**let** orderForm = {...this.state.orderForm};

    orderForm[key].value = e.target.value;

    orderForm[key].valid = this.checkValidity(e.target.value, orderForm[key].validation);

    this.setState({

      orderForm: orderForm

    });

  }

  render() {

**const** formElementsArray = [];

    for (**let** key in this.state.orderForm) {

      formElementsArray.push({

        id: key,

        config: this.state.orderForm[key]

      });

    }

    return (

      <div className={classes.ContactData}>

        <h4>Enter your Contact please</h4>

        {!this.state.loading ? (

        <form onSubmit={this.orderHandler} >

          {formElementsArray.map(formElement **=>** {

            return <Input

              key={formElement.id}

              elementType={formElement.config.elementType}

              elementConfig={formElement.config.elementConfig}

              value={formElement.config.value}

              changed={(e) **=>** this.inputChangedHandler(e, formElement.id)}

            />

          })}

          <Button btnType="Success" clicked={this.orderHandler}>ORDER</Button>

        </form>

        ) : <Spinner />}

      </div>

    )

  }

}

**Adding Validation Feedback**

* Chúng ta sẽ display một visual indicator gì đó nếu như một field input is invalid
* Đối với mỗi field trong **orderForm** trong state của ContactData, thêm một prop để check xem input đã được touch hay chưa trước khi display visual indicator

**src\containers\Checkout\ContactData\ContactData.js**

      name: {

        elementType: 'input',

        elementConfig: {

          type: 'text',

          placeholder: 'Your Name'

        },

        value: '',

        validation: {

          required: true

        },

        valid: false,

        touched: false *// tương tự với các field có validation khác*

      },

  inputChangedHandler = (e, key) **=>** {

**let** orderForm = {...this.state.orderForm};

    orderForm[key].value = e.target.value;

    orderForm[key].valid = this.checkValidity(e.target.value, orderForm[key].validation);

    orderForm[key].touched = true; *// nếu input đã được thay đổi, toggle touch props value*

    console.log(orderForm);

    this.setState({

      orderForm: orderForm

    });

  }

        {!this.state.loading ? (

        <form onSubmit={this.orderHandler} >

          {formElementsArray.map(formElement **=>** {

            return <Input

              key={formElement.id}

              elementType={formElement.config.elementType}

              elementConfig={formElement.config.elementConfig}

              value={formElement.config.value}

              invalid={!formElement.config.valid} *// truyền valid props vào cho Input component*

              shouldValidate={formElement.config.validation} *// để không check validation cho dropdown*

              touched={formElement.config.touched}

              changed={(e) **=>** this.inputChangedHandler(e, formElement.id)}

            />

          })}

          <Button btnType="Success" clicked={this.orderHandler}>ORDER</Button>

        </form>

        ) : <Spinner />}

**src\components\UI\Input\Input.js**

**const** input = props **=>** {

**let** inputElement = null;

**const** inputClasses = [classes.InputElement];

  if (props.invalid && props.shouldValidate && props.touched) {

    inputClasses.push(classes.Invalid);

  }

  switch (props.elementType) {

    case ('input'):

      inputElement = <input

        className={inputClasses.join(' ')}

        {...props.elementConfig}

        onChange={props.changed}

        value={props.value} />;

      break;

    case ('textarea'):

      inputElement = <textarea

        className={inputClasses.join(' ')}

        {...props.elementConfig}

        onChange={props.changed}

        value={props.value} />;

      break;

    case ('select'):

      inputElement = (

        <select

          className={inputClasses.join(' ')}

          value={props.value}

          onChange={props.changed}

        >

          {props.elementConfig.options.map(option **=>** {

            return <option

              key={option.value}

              value={option.value}>

                {option.displayValue}

              </option>

          })}

        </select>

      )

      break;

    default:

      inputElement = <input

        className={inputClasses.join(' ')}

        {...props.elementConfig}

        onChange={props.changed}

        value={props.value} />;

  }

  return (

    <div className={classes.Input}>

      <label>{props.elementConfig.placeholder}</label>

      {inputElement}

    </div>

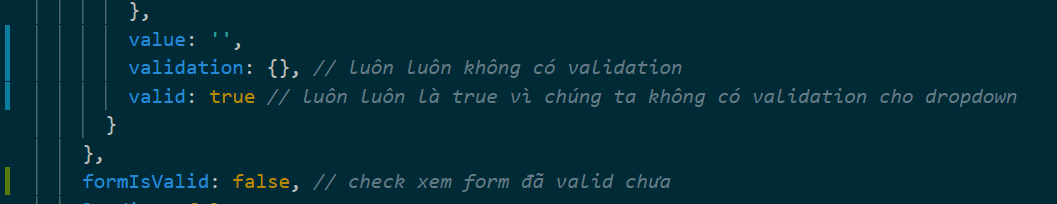
  )

};

**Handling Overall Form Validity**

* Chúng ta sẽ block việc cho phép user submit the form nếu như có field chưa valid

**src\containers\Checkout\ContactData\ContactData.js**

****

  inputChangedHandler = (e, key) **=>** {

**let** orderForm = {...this.state.orderForm};

    orderForm[key].value = e.target.value;

    orderForm[key].valid = this.checkValidity(e.target.value, orderForm[key].validation);

    orderForm[key].touched = true; *// nếu input đã được thay đổi, toggle touch props value*

**let** formIsValid = true;

    for (**let** key in orderForm) {

      if (!orderForm[key].valid) formIsValid = false;

    }

    this.setState({

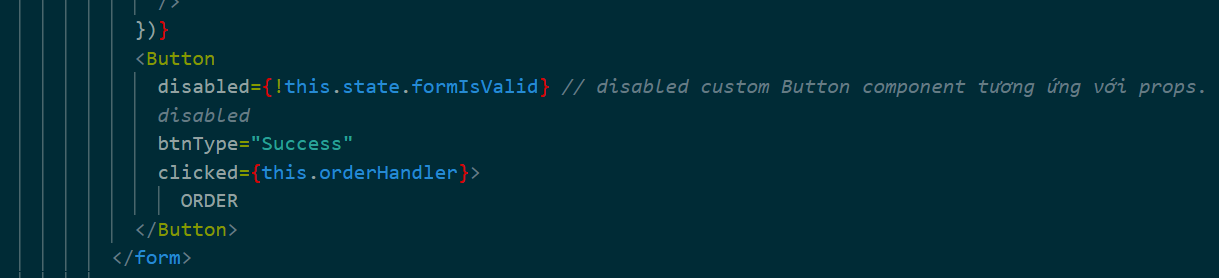
      orderForm: orderForm,

      formIsValid: formIsValid

    });

  }

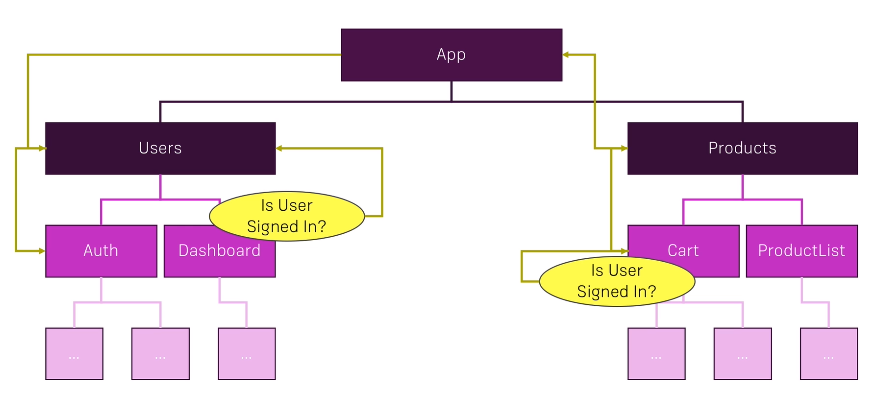
* Disabled Button component tương ứng với validity của order form

****

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

**Section 14: Redux**

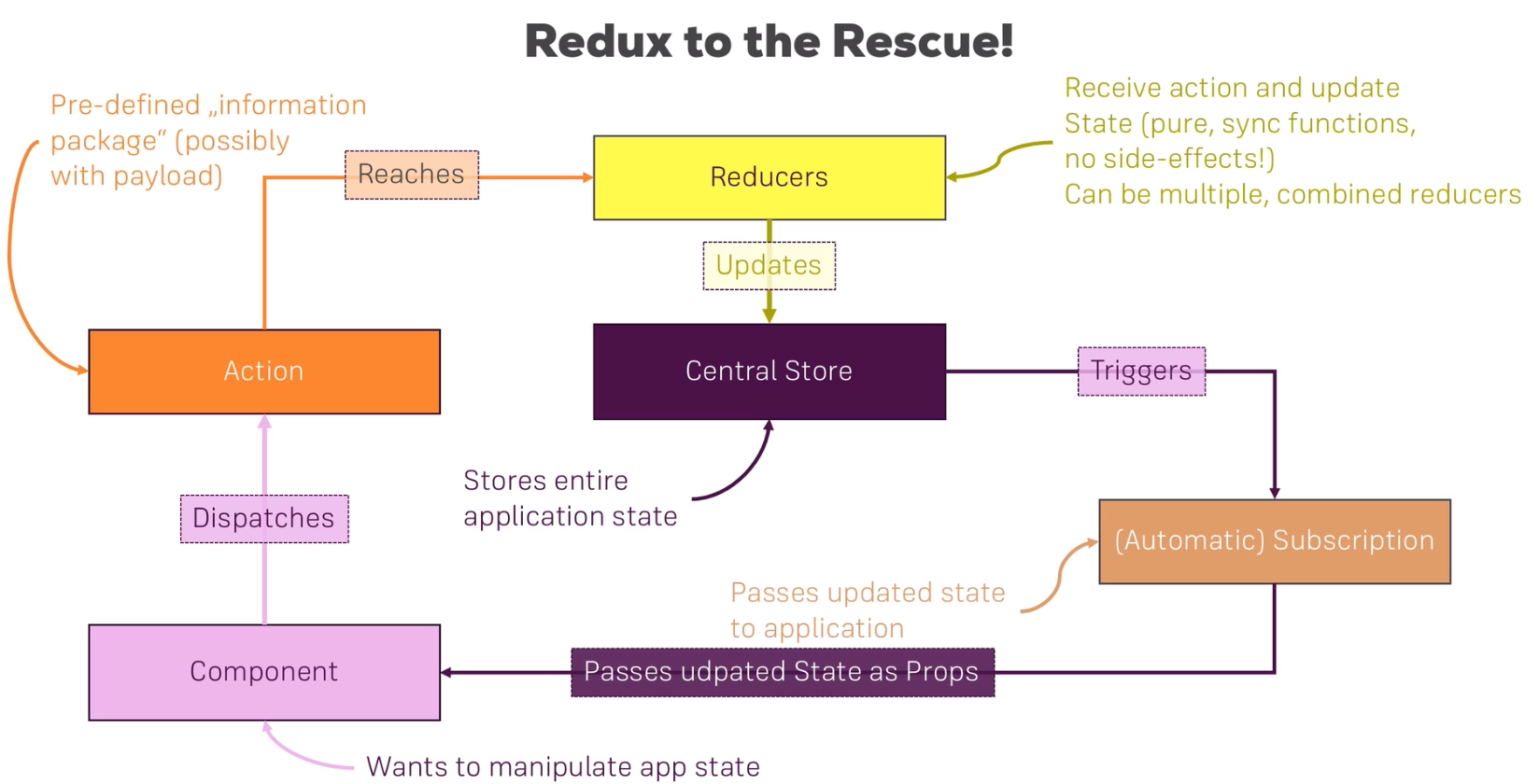
* State là gì?
  + Có thể hiêu state là những gì mà component của chúng ta cần để xác định những gì chúng ta sẽ render, hay cách là chúng ta render ra các component như thế nào.
* Sự phức tạp của quản lý state
  + State có thể rất phức tạp
  + Có thể nhìn ứng dụng sau để hiểu việc quản lý state có thể sẽ phức tạp như thế nào trong các ứng dụng lớn:

****

* Giải pháp:
  + Set some global variable sẽ store state của cả ứng dụng và chúng ta có thể access ở bất cứ đầu trong ứng dụng
  + 🡺 Lý do redux xuất hiện

**Understanding the Redux Flow**

* Central Store:
  + Mỗi redux appplcation sẽ có một Central Store
  + Cần phải nhớ rằng redux là một 3rd party library hoạt động độc lập với react

****

**Setting up Reducer and Store**

* Install Redux: **npm install –save redux**
* Như đã nói, redux hoạt động hoàn toàn độc lập đối với react application, do đó chúng ta sẽ sử dụng Redux một cách độc lập với ứng dụng này.

**redux-basics.js**

**const** redux = require('redux');

**const** createStore = redux.createStore;

**const** initState = {

  counter: 0

}

*/\* Reducer \*/*

**const** rootReducer = (state = initState, action) **=>** {

  return state;

}

*/\* Store \*/*

*// - Store cần được khởi tạo với một reducer, sau tất cả chúng ta chỉ có một reducer ngay cả khi chúng ta combile nhiều reducer*

*// - Reducer liên kết trực tiếp với store, là thứ duy nhất có thể update state.*

**const** store = createStore(rootReducer);

console.log(store.getState()); *// { counter: 0 }*

*// Dispatching Action*

*// Subcription*

**Dispatching Actions**

**const** rootReducer = (state = initState, action) **=>** {

  if (action.type === 'INC\_COUNTER') {

    return {

      ...state,

      counter: state.counter + 1

    };

  }

  if (action.type === 'ADD\_COUNTER') {

    return {

      ...state,

      counter: state.counter + action.value

    };

  }

  return state;

}

*/\* Store \*/*

*// - Store cần được khởi tạo với một reducer, sau tất cả chúng ta chỉ có một reducer ngay cả khi chúng ta combile nhiều reducer*

*// - Reducer liên kết trực tiếp với store, là thứ duy nhất có thể update state.*

**const** store = createStore(rootReducer);

console.log("First: ", store.getState()); *// { counter: 0 }*

*// Dispatching Action*

store.dispatch({

  type: 'INC\_COUNTER' *// convention*

});

store.dispatch({ type: 'ADD\_COUNTER', value: 10 });

console.log("Second: ", store.getState()); *// { counter: 11 }*

**Adding Subscriptions**

* Thay vì gọi getState() như trên để lấy giá trị snapshot của current state mà chúng ta muốn được inform mỗi khi có sự thay đổi của state

**redux-basics.js**

**const** store = createStore(rootReducer);

console.log("First: ", store.getState()); *// { counter: 0 }*

*// Subcription*

store.subscribe(() **=>** { *// nhận vào 1 callback function được executed mỗi khi state được updated*

  console.log('[Subscription]', store.getState());

});

*// Dispatching Action*

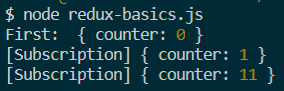
store.dispatch({

  type: 'INC\_COUNTER' *// convention*

});

store.dispatch({ type: 'ADD\_COUNTER', value: 10 });

*// console.log("Second: ", store.getState()); // { counter: 11 }*

****

**Connecting React to Redux**

* Chúng ta sẽ create store ngay trước khi application start, như ở trường hợp này sẽ là ở index.js

**src\index.js**

import { createStore } from 'redux';

import reducer from './store/reducer';

**const** store = createStore(reducer);

**src\store\reducer.js**

**const** initState = {

  counter: 0

};

**const** reducer = (state = initState, action) **=>** {

  return state;

}

export default reducer;

**Connecting the Store to React**

* Như đã biết redux alone là một library độc lập, do đó để connect redux với react app, chúng ta sẽ dùng package:
  + **Npm install –save react-redux**
  + Chúng ta sẽ store state của toàn ứng dụng trong index.js như sau:

**src\index.js**

import { createStore } from 'redux';

import { Provider } from 'react-redux';

import reducer from './store/reducer';

const store = createStore(reducer);

ReactDOM.render(<Provider store={store}><App /></Provider>, document.getElementById('root')); // đây là một helper component để inject store vào React component

* + Để passing store này xuống các component, chúng ta sử dụng component **Provider** được cung cấp bởi react-redux, gần giống như context API

**src\store\reducer.js**

const initState = {

  counter: 0

};

const reducer = (state = initState, action) => {

  return state;

}

export default reducer;

* + Để embed state này vào component, chúng ta sẽ làm như sau:

**src\containers\Counter\Counter.js**

import React, { Component } from 'react';

import { connect } from 'react-redux'; // đây là 1 function, nhận vào 1 configuration object, và trả về một HOC sẽ nhận vào component được passed state từ store

import CounterControl from '../../components/CounterControl/CounterControl';

import CounterOutput from '../../components/CounterOutput/CounterOutput';

class Counter extends Component {

    state = {

        counter: 0

    }

    counterChangedHandler = ( action, value ) => {

        switch ( action ) {

            case 'inc':

                this.setState( ( prevState ) => { return { counter: prevState.counter + 1 } } )

                break;

            case 'dec':

                this.setState( ( prevState ) => { return { counter: prevState.counter - 1 } } )

                break;

            case 'add':

                this.setState( ( prevState ) => { return { counter: prevState.counter + value } } )

                break;

            case 'sub':

                this.setState( ( prevState ) => { return { counter: prevState.counter - value } } )

                break;

        }

    }

    render () {

        return (

            <div>

                <CounterOutput value={this.props.ctr} /> {/\* ở đây chúng ta sẽ dùng props chứ không còn là state nữa. \*/}

                <CounterControl label="Increment" clicked={() => this.counterChangedHandler( 'inc' )} />

                <CounterControl label="Decrement" clicked={() => this.counterChangedHandler( 'dec' )}  />

                <CounterControl label="Add 5" clicked={() => this.counterChangedHandler( 'add', 5 )}  />

                <CounterControl label="Subtract 5" clicked={() => this.counterChangedHandler( 'sub', 5 )}  />

            </div>

        );

    }

}

const mapStateToProps = state => { // configuration object

    return {

        ctr: state.counter

    };

}

export default connect(mapStateToProps)(Counter);

**Dispatching Actions from within the Component**

* Dispatch action trong component cũng rất đơn giản và gần tương tự với state:

**src\containers\Counter\Counter.js**

    render () {

        return (

            <div>

                <CounterOutput value={this.props.ctr} /> {/\* ở đây chúng ta sẽ dùng props chứ không còn là state nữa. \*/}

                <CounterControl label="Increment" clicked={this.props.onIncrementCounter} /> {/\* dùng action được dispatched. \*/}

                <CounterControl label="Decrement" clicked={() => this.counterChangedHandler( 'dec' )}  />

                <CounterControl label="Add 5" clicked={() => this.counterChangedHandler( 'add', 5 )}  />

                <CounterControl label="Subtract 5" clicked={() => this.counterChangedHandler( 'sub', 5 )}  />

            </div>

        );

    }

}

const mapStateToProps = state => { // configuration object

    return {

        ctr: state.counter

    };

}

const mapDispatchToProps = dispatch => {

    return {

        onIncrementCounter: () => dispatch({

            type: 'INCREMENT'

        })

    };

}

export default connect(mapStateToProps, mapDispatchToProps)(Counter); // second arg chính là actions là chúng ta muốn dispatch từ component này

**Passing and Retrieving Data with Action**

* Chúng ta sẽ truyền các data vào dưới dạng object named **payload.**

**src\containers\Counter\Counter.js**

import React, { Component } from 'react';

import { connect } from 'react-redux'; // đây là 1 function, nhận vào 1 configuration object, và trả về một HOC sẽ nhận vào component được passed state từ store

import CounterControl from '../../components/CounterControl/CounterControl';

import CounterOutput from '../../components/CounterOutput/CounterOutput';

class Counter extends Component {

    render () {

        return (

            <div>

                <CounterOutput value={this.props.ctr} /> {/\* ở đây chúng ta sẽ dùng props chứ không còn là state nữa. \*/}

                <CounterControl label="Increment" clicked={this.props.onIncrementCounter} /> {/\* dùng action được dispatched. \*/}

                <CounterControl label="Decrement" clicked={this.props.onDecreasementCounter}  />

                <CounterControl label="Add 5" clicked={this.props.onAddCounter}  />

                <CounterControl label="Subtract 5" clicked={this.props.onSubtractCounter}  />

            </div>

        );

    }

}

const mapStateToProps = state => { // configuration object

    return {

        ctr: state.counter

    };

}

const mapDispatchToProps = dispatch => {

    return {

        onIncrementCounter: () => dispatch({

            type: 'INCREMENT'

        }),

        onDecreasementCounter: () => dispatch({

            type: 'DECREASEMENT'

        }),

        onAddCounter: () => dispatch({

            type: 'ADD',

            payload: {

                value: 5

            }

        }),

        onSubtractCounter: () => dispatch({

            type: 'SUBTRACT',

            payload: {

                value: 5

            }

        })

    };

}

export default connect(mapStateToProps, mapDispatchToProps)(Counter); // second arg chính là actions là chúng ta muốn dispatch từ component này

**src\store\reducer.js**

const initState = {

  counter: 0

};

const reducer = (state = initState, action) => {

  if (action.type === 'INCREMENT') {

    return {

      counter: state.counter + 1

    }

  } else if (action.type === 'DECREASEMENT') {

    return {

      counter: state.counter - 1

    }

  } else if (action.type === 'ADD') {

    return {

      counter: state.counter + action.payload.value

    }

  } else if (action.type === 'SUBTRACT') {

    return {

      counter: state.counter - action.payload.value

    }

  }

  return state;

}

export default reducer;

**Switch-case in reducer**

**src\store\reducer.js**

**const** reducer = (state = initState, action) **=>** {

  switch (action.type) {

    case 'INCREMENT':

      return {

        counter: state.counter + 1

      }

    case 'DECREASEMENT':

      return {

        counter: state.counter - 1

      }

    case 'ADD':

      return {

        counter: state.counter + action.payload.value

      }

    case 'SUBTRACT':

      return {

        counter: state.counter - action.payload.value

      }

    default:

      return state;

      break;

  }

}

**Updating State Immutably**

**src\store\reducer.js**

**const** reducer = (state = initState, action) **=>** {

  switch (action.type) {

    case 'INCREMENT':

      return {

        counter: state.counter + 1,

        result: [...state.result] *// chúng ta phải thực hiện thêm lại state cho result, vì trong redux state sẽ không được merged mà sẽ override giá trị cũ*

      }

    case 'DECREASEMENT':

      return {

        ...state, *// update state immutably*

        counter: state.counter - 1

      }

    case 'ADD':

      return {

        counter: state.counter + action.payload.value, *// counter là một value type nên có thể làm như thế này*

        result: [...state.result]

      }

    case 'SUBTRACT':

      return {

        ...state,

        counter: state.counter - action.payload.value

      }

    case 'STORE\_RESULT':

**const** newState = Object.assign({}, state); *// clone the old object, mục đích để update state immutably*

      return {

        counter: newState.counter,

*// result: [...newState.result, newState.counter]*

        result: state.result.concat({id: new Date(), value: state.counter}) *// đây cũng là 1 cách.*

      }

    case 'DELETE\_RESULT':

      console.log(action.payload.id);

    default:

      return state;

  }

}

**Updating Array Immutably**

**src\containers\Counter\Counter.js**

**class** Counter **extends** Component {

    render () {

        return (

            <div>

                . . . . . .

                <hr />

                <button onClick={this.props.onStoreResult}>Store Result</button>

                <ul>

                    {this.props.storedResult.map((item) **=>** (

                        <li

                            onClick={() **=>** this.props.onDeleteResult(item.id)}

                            key={item.id}

                        >

                            {item.value}

                        </li>

                    ))}

                </ul>

            </div>

        );

    }

}

**const** mapStateToProps = state **=>** { *// configuration object*

    return {

        ctr: state.counter,

        storedResult: state.result

    };

}

**const** mapDispatchToProps = dispatch **=>** {

    return {

        . . .

        onDeleteResult: (id) **=>** dispatch({

            type: 'DELETE\_RESULT',

            payload: {

                id: id

            }

        })

    };

}

**src\store\reducer.js**

**const** reducer = (state = initState, action) **=>** {

  switch (action.type) {

    . . . . . . . .

    case 'DELETE\_RESULT':

**const** id = action.payload.id;

**const** newArr = [...state.result];

**const** updatedArr = newArr.filter(item **=>** item.id !== id);

      return {

        ...state,

        result: updatedArr

      }

    default:

      return state;

  }

}

**Immutable Update Pattern**

* Một số sai lầm phổ biến trong immutable update pattern:
  + Chỉ tạo ra clone cho một level của object, ví dụ

1. function updateNestedState(state, action) {
2. // Problem: this only does a shallow copy!
3. let newState = {...state};
5. // ERROR: nestedState is still the same object!
6. newState.nestedState.nestedField = action.data;
8. return newState;
9. }
   * + Chỉ tạo ra bản copy ở top level là chưa đủ, các *nestedState* object cũng phải được copied

* Cách làm đúng: phải copy tất cả các level của nested data
  + Đây là ví dụ khi chúng ta muốn update immutable cho các nested state, và việc này đôi khi rất cồng kềnh và khó theo dõi
  + Hiển nhiên là mỗi tầng của nesting đều cần được clone, và đây là lý do chúng ta được khuyến khích giữ cho state được “làm mịn” và

1. function updateVeryNestedField(state, action) {
2. return {
3. ...state,
4. first : {
5. ...state.first,
6. second : {
7. ...state.first.second,
8. [action.someId] : {
9. ...state.first.second[action.someId],
10. fourth : action.someValue
11. }
12. }
13. }
14. }
15. }

Tham khảo chapter 265: <https://www.udemy.com/course/react-the-complete-guide-incl-redux/learn/lecture/8303068#content>

**Outsourcing Action Types**

* Trong reducer của chúng ta, có rất nhiều cases dù đều được đu=ịnh danh khác nhau, và dispatch actions bên trong component cũng khác nhau, tuy nhiên khi ứng dụng trở nên lớn hơn, việc xác định các identification cũng rất khó.

🡺 Chúng ta sẽ thực hiện việc outsource các action types thành các constant, và sau đó import các constant

**src\store\actions.js**

export **const** INCREMENT = 'INCREMENT';

export **const** DECREASEMENT = 'DECREASEMENT';

export **const** ADD = 'ADD';

export **const** SUBTRACT = 'SUBTRACT';

export **const** STORE\_RESULT = 'STORE\_RESULT';

export **const** DELETE\_RESULT = 'DELETE\_RESULT';

**src\store\reducer.js**

import \* as actionTypes from './actions';

. . . . .

**const** reducer = (state = initState, action) **=>** {

  switch (action.type) {

    case actionTypes.INCREMENT:

      . . . .

    default:

      return state;

  }

}

export default reducer;

**src\containers\Counter\Counter.js**

        onIncrementCounter: () **=>** dispatch({

            type: actionTypes.INCREMENT

        }),

        onDecreasementCounter: () **=>** dispatch({

            type: actionTypes.DECREASEMENT

        }),

        onAddCounter: () **=>** dispatch({

            type: actionTypes.ADD,

            payload: {

                value: 5

            }

        }),

        onSubtractCounter: () **=>** dispatch({

            type: actionTypes.SUBTRACT,

            payload: {

                value: 5

            }

        }),

        onStoreResult: () **=>** dispatch({

            type: actionTypes.STORE\_RESULT

        }),

        onDeleteResult: (id) **=>** dispatch({

            type: actionTypes.DELETE\_RESULT,

            payload: {

                id: id

            }

        })

**Combining Multiple Reducers**

* Khi ứng dụng của chúng ta lớn dần, chúng ta sẽ muốn sử dụng nhiều reducer trong ứng dụng
* Mặc dù redux như đã nói, sau tất cả chúng ta sẽ chỉ có 1 reducer, tuy nhiên redux cung cấp cho chúng ta methods để có để có multiple reducer trong ứng dụng và combine các reducer này lại với nhau
* Trong ứng dụng trên, chúng ta sẽ sử dụng 2 reducer khác nhau để handle **result** và **counter**

(Khá dài, chủ yếu xem code)

**src\index.js**

import { createStore, combineReducers } from 'redux'; *// combineReducers là func nhận vào js object mapping các reducers thành các slices của state như là các input và sẽ merges tất cả thành một reducer và 1 state cho chúng ta (như chúng ta làm ban đầu)*

import { Provider } from 'react-redux';

import counterReducer from './store/reducers/counter';

import resultReducer from './store/reducers/result';

**const** rootReducer = combineReducers({

  ctr: counterReducer,

  res: resultReducer

});

**const** store = createStore(rootReducer);

**src\store\reducers\counter.js**

import \* as actionTypes from '../actions';

**const** initState = {

  counter: 0

};

**const** counterReducer = (state = initState, action) **=>** {

  switch (action.type) {

    case actionTypes.INCREMENT:

      return {

        ...state,

        counter: state.counter + 1,

      }

    case actionTypes.DECREASEMENT:

      return {

        ...state, *// update state immutably*

        counter: state.counter - 1

      }

    case actionTypes.ADD:

      return {

       ...state,

       counter: state.counter + action.payload.value

      }

    case actionTypes.SUBTRACT:

      return {

        ...state,

        counter: state.counter - action.payload.value

      }

    default:

      return state;

  }

}

export default counterReducer;

**src\store\reducers\result.js**

import \* as actionTypes from '../actions';

**const** initState = {

  result: []

};

**const** resultReducer = (state = initState, action) **=>** {

  switch (action.type) {

    case actionTypes.STORE\_RESULT:

**const** newState = Object.assign({}, state); *// clone the old object, mục đích để update state immutably*

      return {

        counter: newState.counter,

*// result: [...newState.result, newState.counter]*

        result: state.result.concat({id: (new Date()).toISOString(), value: action.payload.counter}) *// đây cũng là 1 cách.*

      }

    case actionTypes.DELETE\_RESULT:

**const** id = action.payload.id;

**const** newArr = [...state.result];

**const** updatedArr = newArr.filter(item **=>** item.id !== id);

      return {

        ...state,

        result: updatedArr

      }

    default:

      return state;

  }

}

export default resultReducer;

**src\containers\Counter\Counter.js**

**class** Counter **extends** Component {

    render () {

        return (

            <div>

                <CounterOutput value={this.props.ctr} /> {*/\* ở đây chúng ta sẽ dùng props chứ không còn là state nữa. \*/*}

                <CounterControl label="Increment" clicked={this.props.onIncrementCounter} /> {*/\* dùng action được dispatched. \*/*}

                <CounterControl label="Decrement" clicked={this.props.onDecreasementCounter}  />

                <CounterControl label="Add 5" clicked={this.props.onAddCounter}  />

                <CounterControl label="Subtract 5" clicked={this.props.onSubtractCounter}  />

                <hr />

                <button onClick={() **=>** this.props.onStoreResult(this.props.ctr)}>Store Result</button>

                <ul>

                    {this.props.storedResult.map(item **=>** (

                        <li

                            onClick={() **=>** this.props.onDeleteResult(item.id)}

                            key={item.id}

                        >

                            {item.value}

                        </li>

                    ))}

                </ul>

            </div>

        );

    }

}

**const** mapStateToProps = state **=>** { *// configuration object*

    return {

        ctr: state.ctr.counter,

        storedResult: state.res.result

    };

}

**const** mapDispatchToProps = dispatch **=>** {

    return {

        onIncrementCounter: () **=>** dispatch({

            type: actionTypes.INCREMENT

        }),

        onDecreasementCounter: () **=>** dispatch({

            type: actionTypes.DECREASEMENT

        }),

        onAddCounter: () **=>** dispatch({

            type: actionTypes.ADD,

            payload: {

                value: 5

            }

        }),

        onSubtractCounter: () **=>** dispatch({

            type: actionTypes.SUBTRACT,

            payload: {

                value: 5

            }

        }),

        onStoreResult: (counter) **=>** dispatch({

            type: actionTypes.STORE\_RESULT,

            payload: {

                counter: counter

            }

        }),

        onDeleteResult: (id) **=>** dispatch({

            type: actionTypes.DELETE\_RESULT,

            payload: {

                id: id

            }

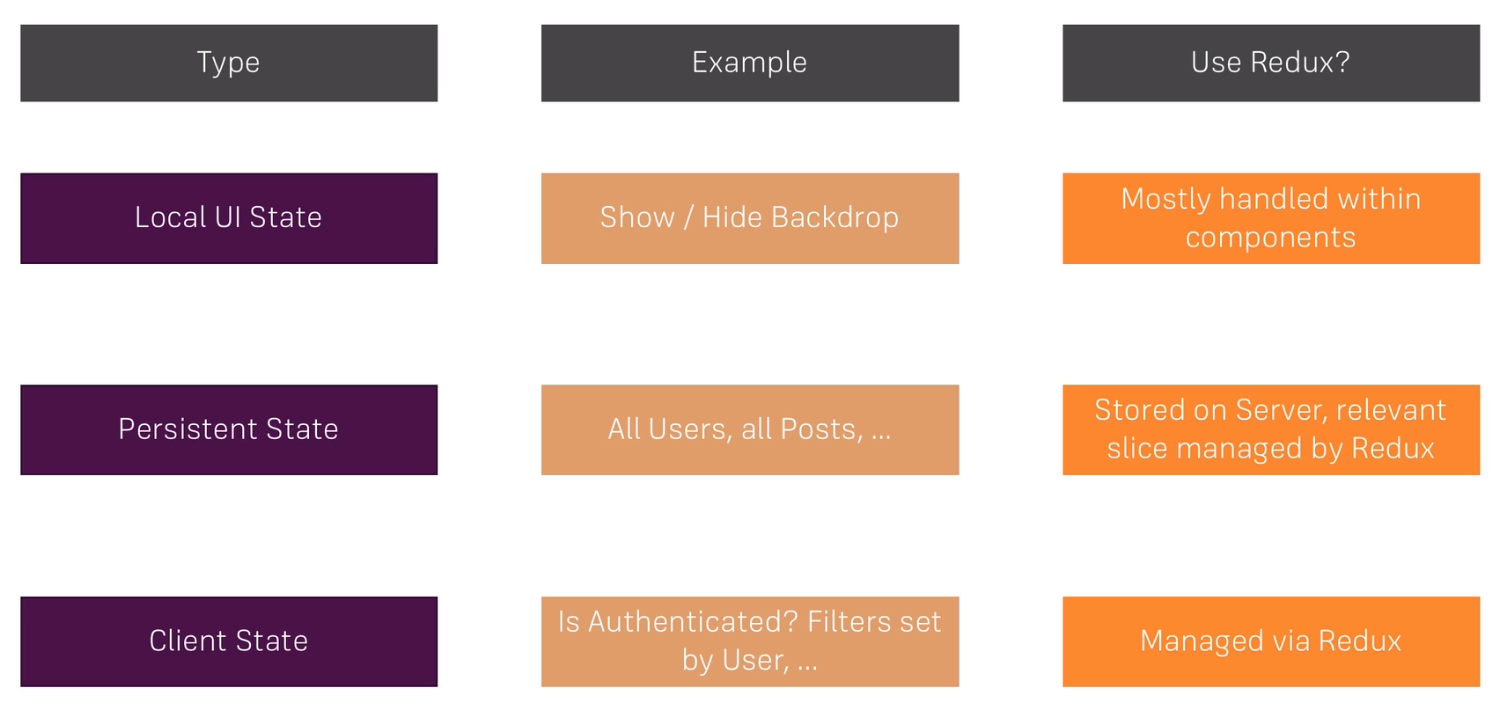
        })

    };

}

**Understanding State Types**

* Types of State và những state như thế nào thì nên sử dụng redux để store lại?



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

**Section 15: Adding Redux to our Project**

**src\store\reducer.js**

const initState = {

  ingredients: {

    salad: 0,

    bacon: 0,

    cheese: 0,

    meat: 0

  },

  totalPrice: 4

}

const reducer = (state = initState, action) => {

  switch (action.type) {

    case actionTypes.ADD\_INGREDIENT:

      return {

        ...state,

        ingredients: {

          ...state.ingredients, // cần phải clone cả các nested object, không chỉ object cha

          [action.ingredientName]: state.ingredients[action.ingredientName] + 1 // syntax để override dynamically property trong JS object

        }

      }

    case actionTypes.REMOVE\_INGREDIENT:

      return {

        ...state,

        ingredients: {

          ...state.ingredients, // cần phải clone cả các nested object, không chỉ object cha

          [action.ingredientName]: state.ingredients[action.ingredientName] - 1 // syntax để override dynamically property trong JS object

        }

      }

    default:

      return state;

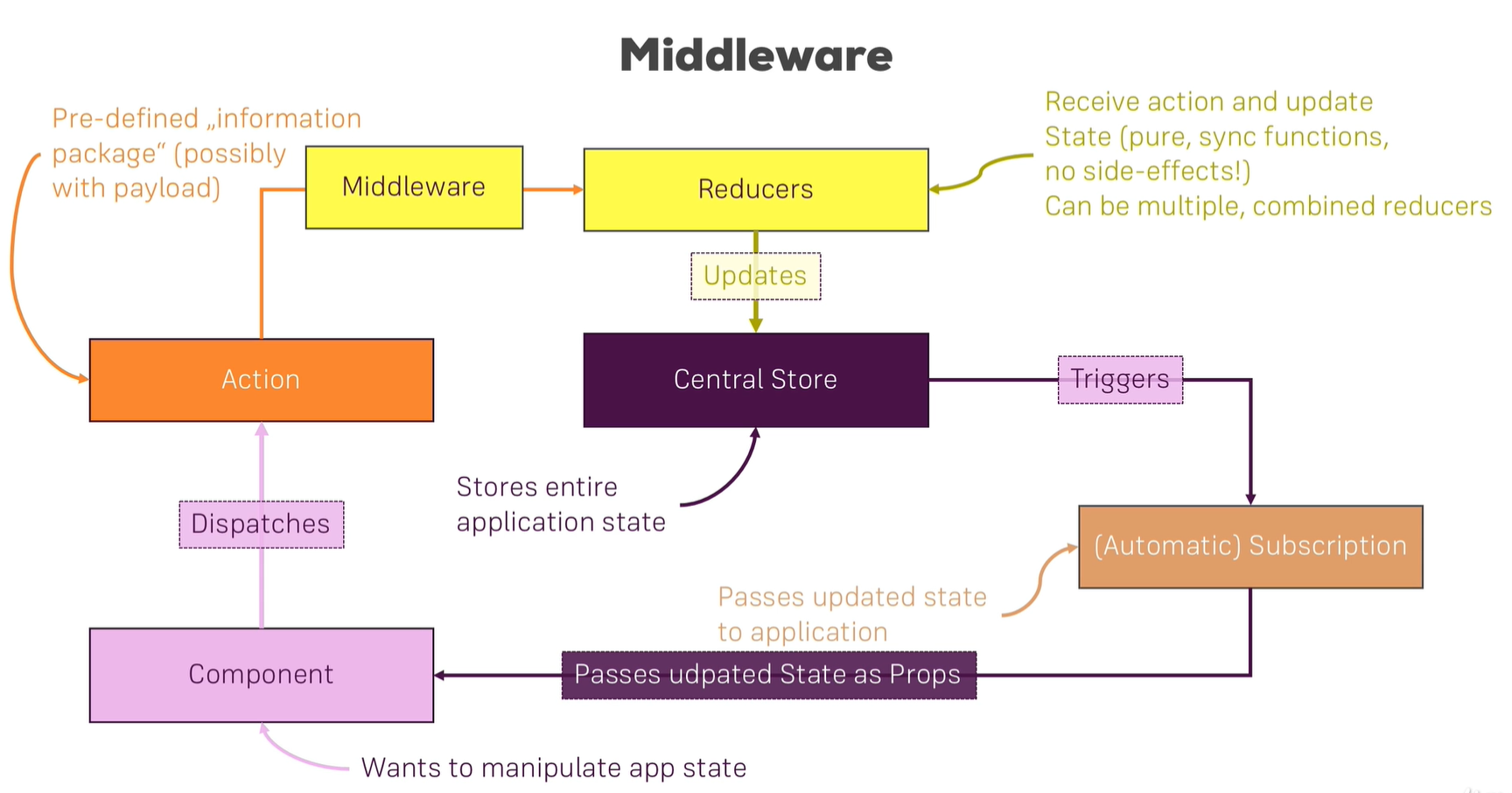
  }

}

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

**Redux Advanced**

**Adding Middleware**

****

* **Middleware** là một concept quan trọng của Redux itself (không liên quan gì đến React)
  + Đây là một term được dùng cho fuonction(s) hay code được hook vào process sẽ được executed mà không làm stop process này.
  + Để implement middleware chúng ta sẽ làm như sau:

**src\index.js**

import React from 'react';

import ReactDOM from 'react-dom';

import './index.css';

import App from './App';

import registerServiceWorker from './registerServiceWorker';

import { createStore, combineReducers, applyMiddleware } from 'redux'; // combineReducers là func nhận vào js object mapping các reducers thành các slices của state như là các input và sẽ merges tất cả thành một reducer và 1 state cho chúng ta (như chúng ta làm ban đầu). applyMiddleware để apply middleware

import { Provider } from 'react-redux';

import counterReducer from './store/reducers/counter';

import resultReducer from './store/reducers/result';

const rootReducer = combineReducers({

  ctr: counterReducer,

  res: resultReducer

});

const logger = store => {

  return next => {

    return action => { // action chúng ta dispatch. Ở đây chúng ta có thể access đến store và next() function

      console.log('[Middleware] Dispatching', action);

      const result = next(action); // let the action continue to the reducer, và ở đây chúng ta có thể change the action

      console.log('[Middleware] next state', store.getState());

      return result;

    }

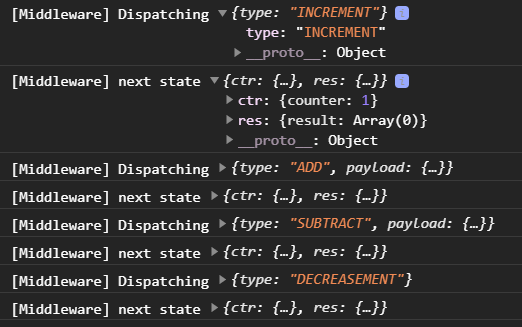
  }

};

const store = createStore(rootReducer, applyMiddleware(logger)); // có thể là một list các middleware, ở đây chúng ta chỉ apply 1 middleware

ReactDOM.render(<Provider store={store}><App /></Provider>, document.getElementById('root')); // đây là một helper component để inject store vào React component

registerServiceWorker();

****

**Using the Redux devtools**

**Executing Async code**

* Chúng ta sẽ giả sử sau khi bấm store result, sau đó result sẽ được sending to server (fake API using setTimeout)
  + Như đã biết chúng ta sẽ không dùng các async code bên trong reducer
  + Sử dụng một công cụ khác của redux, đó là **action creator**
  + Action creator đơn giản là một function trả về một action hoặc tạo ra một action
  + Action creatorr rất hữu dụng khi xử lý async code, tuy nhiên trước hết chúng ta sẽ sử dụng action creator cho sync code trước.
  + Đây là cleaner way để tạo ra các action

**src\store\actions\actions.js**

**const** INCREMENT = 'INCREMENT';

**const** DECREASEMENT = 'DECREASEMENT';

**const** ADD = 'ADD';

**const** SUBTRACT = 'SUBTRACT';

**const** STORE\_RESULT = 'STORE\_RESULT';

**const** DELETE\_RESULT = 'DELETE\_RESULT';

export **const** increment = () **=>** {

  return {

    type: INCREMENT

  }

}

export **const** decreasement = () **=>** {

  return {

    type: DECREASEMENT

  }

}

export **const** add = (val) **=>** {

  return {

    type: ADD,

    payload: {

      value: val

    }

  }

}

export **const** subtract = (val) **=>** {

  return {

    type: SUBTRACT,

    payload: {

      value: val

    }

  }

}

export **const** storeDefault = counter **=>** {

  return {

    type: STORE\_RESULT,

    payload: {

      counter: counter

    }

  }

}

export **const** deleteResult = id **=>** {

  return {

    type: DELETE\_RESULT,

    payload: {

      id: id

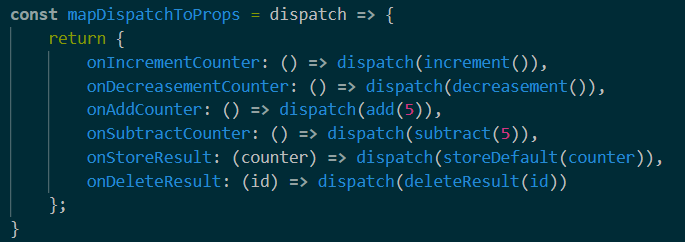
    }

  }

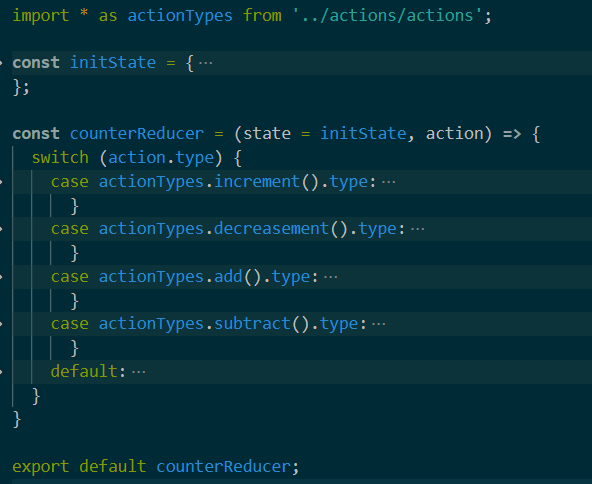
}

**src\containers\Counter\Counter.js**





**src\store\reducers\counter.js**



**Handling Asynchronous Code**

**Tham khảo:** [**Redux-thunk Viblo**](https://viblo.asia/p/tim-hieu-ve-redux-thunk-Qbq5Qkm4ZD8)

* Cho tới thời điểm hiện tạichúng ta mới chỉ handle sync action creators, đối với async code, sẽ có chút khác biệt, chúng ta sẽ cần một special middleware, là một 3rd là **redux-thunk**
  + **npm install --save redux-thunk**
  + Đây là một library sẽ giúp thêm vào một middleware vào project, cho phép actions không return về một action là một function thay vì chỉ là một PJO (plain js object) như chúng ta vẫn thường làm đối với sync code ở trên.
  + Redux-thunk sẽ cung cấp một middleware (thunk) được đặt trước thời điểm reducer nhận được request để nhận biết các action có return một PJO hay không
    - Nếu là một PJO, thunk sẽ pass action đó đến reducer như thường lệ
    - Nếu là một function, redux thunk sẽ block action đó lại và wait cho đến khi một lệnh asynchronous nào đó trong func hoàn tất và trả về kết quả như là một PJO và thunk sẽ đưa action này đến reducer như bình thường.
    - Thông thường:

****

* + - Khi sử dụng với middleware của redux-thunk:

****

* + Với trick này, chúng ta có thể run các async code vì dispatched sau cùng có thể run async
  + Có thể hiểu redux-thunk là một middleware cho phép viết các action return về một function thay vì một PJO bằng các trì hoãn việc đưa action đến reducer

**src\index.js**

import thunk from 'redux-thunk';

**const** store = createStore(rootReducer, applyMiddleware(logger, thunk));

**src\store\actions\actions.js**

export **const** saveResult = counter **=>** {

  return {

    type: STORE\_RESULT,

    payload: {

      counter: counter

    }

  }

}

export **const** storeDefault = counter **=>** {

  return dispatch **=>** { *// func này nhận vào một dispatch là một arg, và chúng ta get this dispatch due to redux-thunk. Có thể nói middware chạy giữa khoảng thời gian dispatching một action và thời điểm action reaches the reducer*

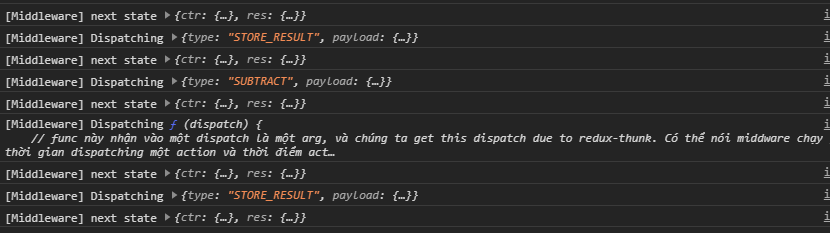
    setTimeout(() **=>** {

      dispatch(saveResult(counter)); *// execute saveResult(counter) để return về action*

    }, 2000);

  }

}



**Restructuring Actions**

**src\store\actions\actionTypes.js**

export **const** INCREMENT = 'INCREMENT';

export **const** DECREASEMENT = 'DECREASEMENT';

export **const** ADD = 'ADD';

export **const** SUBTRACT = 'SUBTRACT';

export **const** STORE\_RESULT = 'STORE\_RESULT';

export **const** DELETE\_RESULT = 'DELETE\_RESULT';

**src\store\actions\counter.js**

import \* as actionTypes from './actionTypes';

export **const** increment = () **=>** {

  return {

    type: actionTypes.INCREMENT

  }

}

export **const** decreasement = () **=>** {

  return {

    type: actionTypes.DECREASEMENT

  }

}

export **const** add = (val) **=>** {

  return {

    type: actionTypes.ADD,

    payload: {

      value: val

    }

  }

}

export **const** subtract = (val) **=>** {

  return {

    type: actionTypes.SUBTRACT,

    payload: {

      value: val

    }

  }

}

**src\store\actions\result.js**

import \* as actionTypes from './actionTypes';

**const** saveResult = counter **=>** {

  return {

    type: actionTypes.STORE\_RESULT,

    payload: {

      counter: counter

    }

  }

}

export **const** storeDefault = counter **=>** {

  return dispatch **=>** { *// func này nhận vào một dispatch là một arg, và chúng ta get this dispatch due to redux-thunk. Có thể nói middware chạy giữa khoảng thời gian dispatching một action và thời điểm action reaches the reducer*

    setTimeout(() **=>** {

      dispatch(saveResult(counter)); *// execute saveResult(counter) để return về action*

    }, 2000);

  }

}

export **const** deleteResult = id **=>** {

  return {

    type: actionTypes.DELETE\_RESULT,

    payload: {

      id: id

    }

  }

}

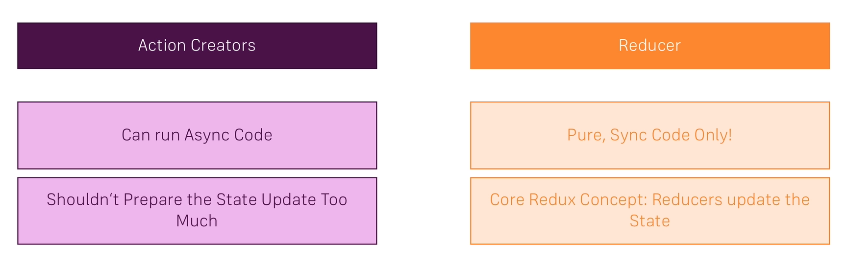
**src\store\actions\index.js**

export { add, decreasement, subtract, increment } from './counter';

export { deleteResult, storeDefault } from './result';

**Where to Put Data Transforming Logic**

* Có 2 chỗ chúng ta có thể dùng làm nơi transform data logic:
  + action fle
  + reducer

****

**Using action creators & Get State**

* Nếu như chúng ta lựa chọn cách put transforming data logic bên trong action creator, đây là một utility method mà chúng ta nên biết khi làm việc với thunk dùng để get current state bên trong action creator
  + Không nên quá overuse getState()

**src\store\actions\result.js**

export **const** storeDefault = counter **=>** {

  return (dispatch, getState) **=>** { *// method để get current state*

    setTimeout(() **=>** {

**const** oldCounter = getState().ctr.counter; *// getvalue of old state.*

      console.log('Old counter:', oldCounter);

      dispatch(saveResult(counter)); *// execute saveResult(counter) để return về action*

    }, 2000);

  }

}

**Using Utility Functions**

* Mặc dù chúng ta đã tách reducer ra thành 2 file để xử lý các action, tuy nhiên số lượng switch-case vẫn khá nhiều, và chúng ta vẫn có thể cleanup thêm nữa.
  + Đây là optional, không bắt buộc
  + Áp dụng tuy thuộc vào project

**src\store\utility.js**

export **const** updateObject = (oldObj, updatedValue) **=>** {

  return {

    ...oldObj,

    ...updatedValue

  }

}

**src\store\utility.js**

import { updateObject } from '../utility';

**const** counterReducer = (state = initState, action) **=>** {

  switch (action.type) {

    case actionTypes.INCREMENT:

      return updateObject(state, { counter: state.counter + 1 });

    case actionTypes.DECREASEMENT:

      return updateObject(state, { counter: state.counter - 1 });

    case actionTypes.ADD:

      return updateObject(state, { counter: state.counter + action.payload.value });

    case actionTypes.SUBTRACT:

      return updateObject(state, { counter: state.counter - action.payload.value });

    default:

      return state;

  }

}

**src\store\reducers\result.js**

**const** resultReducer = (state = initState, action) **=>** {

  switch (action.type) {

    case actionTypes.STORE\_RESULT:

**const** newState = Object.assign({}, state); *// clone the old object, mục đích để update state immutably*

      return updateObject(state, { result: state.result.concat({id: (new Date()).toISOString(), value: action.payload.counter}) });

    case actionTypes.DELETE\_RESULT:

**const** id = action.payload.id;

**const** newArr = [...state.result];

**const** updatedArr = newArr.filter(item **=>** item.id !== id);

      return updateObject(state, { result: updatedArr });

    default:

      return state;

  }

}

**A leaner switch case statement**

* Vẫn tiếp tục là việc làm sao để code trong reducer cleaner hơn.

**src\store\reducers\result.js**

**const** deleteResult = (state, action) **=>** {

**const** id = action.payload.id;

**const** newArr = [...state.result];

**const** updatedArr = newArr.filter(item **=>** item.id !== id);

  return updateObject(state, updatedArr);

}

**const** resultReducer = (state = initState, action) **=>** {

  switch (action.type) {

    case actionTypes.STORE\_RESULT:

**const** newState = Object.assign({}, state); *// clone the old object, mục đích để update state immutably*

      return updateObject(state, { result: state.result.concat({id: (new Date()).toISOString(), value: action.payload.counter}) });

    case actionTypes.DELETE\_RESULT:

      return deleteResult(state, action);

    default:

      return state;

  }

}

**An alternative folder structure**

* Khi mà project của chúng ta lớn hơn, chúng ta có thể có nhiều reducer hơn, và mỗi một container component lại quan tâm đến những reducer nhất định
  + Do đó new approach là chúng ta sẽ tạo ra các store folder tương ứng bên trong các container folder.