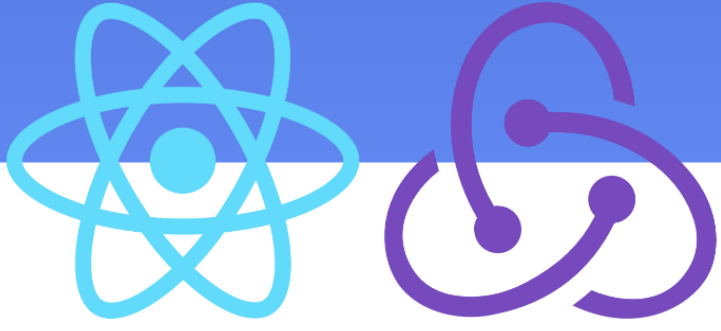
REACT AND REDUX FOR BEGINNERS

By Chai Phonbopit



React และ Redux ฉบับเริ่มต้น

devahoy

React and Redux for Beginners

Chai Phonbopit

This book is for sale at http://leanpub.com/react-and-redux-for-beginners

This version was published on 2018-07-15

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

สารบัญ

Profile

Author: Phonbopit Sahakitchatchawan (Chai)

About the Author

Phonbopit Sahakitchatchawan (พรบพิตร สหกิจชัชวาล) ชื่อเล่น ไช้ เป็นบล็อคเกอร์ที่ devahoy ชื่อใน Internet มักใช้ Chai Phonbopit เพราะว่ามันสั้นและเขียนง่ายกว่า

เคยเป็น Android Developer และ Web Developer โดยเขียนทั้ง Backend และ Frontend โดยใช้ JavaScript เป็นหลัก ทั้ง Node.js (Hapi) และ React.js

Contact Details

- Email: chai@phonbopit.com[1]
- Github : github.com/phonbopit[2]
- Facebook : facebook.com/phonbopit[3]
- Twitter : @phonbopit[4]
- Medium : @phonbopit[5]
- Website : devahoy.com[6]

---

[1] mailto:chai@phonbopit.com
[2] https://github.com/phonbopit
[3] https://facebook.com/phonbopit
[4] https://twitter.com/phonbopit
[5] https://medium.com/@phonbopit
[6] https://devahoy.com

Preface

หนังสือ React and Redux for Beginners หรือพื้นฐาน React และ Redux สำหรับมือใหม่ เป็นการนำเอาบทความที่เผยแพร่ผ่านทางบล็อค https://devahoy.com นำมารวบรวมเป็นฉบับ Ebook โดยเนื้อหาประกอบไปด้วย

- Episode I : พื้นฐาน React
- Episode II : React Router
- Episode III : React Redux

หนังสือเล่มนี้เหมาะสำหรับใคร?

เหมาะสำหรับนักเรียน นักศึกษา หรือผู้ที่สนใจการเขียนเว็บไซต์ ทำเว็บไซต์ และมีพื้นฐานการเขียนเว็บไซต์มาในระดับนึง รวมถึงพอรู้จัก Node.js เบื้องต้น หรือสามารถที่ใช้คำสั่งเบื้องต้นในการติดตั้ง Node Package ได้ (Terminal บน Unix หรือ Power Shell บน Windows) หนังสือเล่มนี้จะไม่สอน HTML/CSS หรือ JavaScript ว่าคืออะไร ฉะนั้นหากไม่เคยเขียนเว็บไซต์มาก่อน หรือไม่รู้จัก HTML/CSS แนะนำควรศึกษามาก่อนอ่านหนังสือเล่มนี้ครับ

เนื้อหา

Episode 1: Basic React

พื้นฐาน React + React คืออะไร? และเข้าใจ React มากขึ้น

- รู้จักกับ State และ Props
- รู้จกกับ Components

Episode 2: React Router

ทำ Single Page Application และเข้าใจการใช้งาน Routing ด้วย React Router v4

Episode 3: React Redux

รู้จักกับ Redux เข้าใจการทำงานของ Redux รู้จักกับ Action, Reducer และ Store รวมถึงการนำไปใช้งาน

Requirements

- Laptop หรือ Computer ที่ลง Mac OS, Linux หรือ Windows
- ติดตั้ง Node.js (v8.0.0 หรือ v10.0.0 ขึ้นไป)
- เตรียม Text Editor หรือ IDEs ให้พร้อม (แนะนำ Visual Studio Code)
- Modern Browser (Chrome หรือ Firefox)
- Git *(optional)*

Code Conventions

ตัวอย่าง Code ในหนังสือเล่มนี้ จะแสดงแบบด้านล่าง

```
function sayHi(name) {
    return 'Ahoy! ' + name
}
```

และแบบที่แสดงชื่อไฟล์ด้วย (เพื่อระบุว่า Code เป็นของไฟล์ไหน) จะเป็นรูปแบบด้านล่างนี้

src/utils/greeting.js

```
function sayHi(name) {
    return 'Ahoy! ' + name
}
```

Command Line จะอยู่ในรูปแบบด้านล่างนี้

```
$ yarn add react
$ yarn add react-router
```

โดยสัญลักษณ์ $ คือ command line input นะครับ ไม่จำเป็นต้องพิมพ์เวลาที่เราจะเรียกคำสั่ง command line

แล้วจะรออะไรอยู่ละ? มาเริ่มเขียน React กันดีกว่า...

Episode I : Basic React

ก่อนไปเริ่มรู้จักกับ React อย่างแรกเลยคือมาดูว่าเราต้องเตรียมตัว เตรียมความพร้อมอะไรบ้างก่อนเริ่มเขียน React กันครับ

สิ่งที่ต้องเตรียมตัวก่อนเขียน React

- ควรมีพื้นฐาน Node.js มาบ้าง (หากไม่รู้จักแนะนำ อ่านเพิ่มเติม Node.js คืออะไร ? + เริ่มต้นใช้งาน Node.js[7])
- ใช้ Command Line พื้นฐานได้ (หรือ Power Shell บน Windows) และพอรู้จัก NPM หรือ Yarn

เมื่อพร้อมแล้วมาเริ่มกันเลย!

React คืออะไร?

React เป็น JavaScript Library ที่พัฒนาโดย Facebook เอาไว้แสดงผล จัดการกับ User Interface (UI) ของเว็บไซต์ ซึ่ง React นั้นไม่ใช่ Framework อย่าง Angular แต่ตัว React นั้นเป็นเพียงแค่ **V** ใน MVC (Model View Controller) เท่านั้น หรือพูดง่ายๆคือเอาไว้ Render DOM Element ต่างๆ ในหน้า HTML

Create React App คืออะไร?

Create React App เป็น Command Line Tools ที่เอาไว้ให้เราสร้างโปรเจ็ค React ได้ง่ายๆ เพียงแค่การพิมพ์คำสั่งบรรทัดเดียว ซึ่งเจ้า Create React App นั้นถูกสร้างโดย Facebook (นำทีมโดย Dan Abramov

[7] https://devahoy.com/posts/getting-started-with-nodejs/

คนคิด Redux นั่นเอง) ซึ่งจริงๆทำผ่านตัว **react-scripts** ซึ่งถ้าหากเราไม่ใช่ตัว Create React App เราก็ต้องโหลด React มาติดตั้งเอง

ซึ่งเจ้า Create React App นั้น รวม Tools และ Config ค่าต่างๆ หลายๆอย่างไว้ให้เราแล้ว โดยที่เราเอาเวลาโฟกัสเรื่องพวกนี้ไปเขียน React จะดีกว่า

ด้านล่างนี้คือ Features ของ Create React App (สำหรับมือใหม่ ข้ามไป เริ่มสร้าง Project ได้เลยครับ)

- Config Webpack สำหรับทั้ง Development และ Production (Minify CSS, JS, Images)
- Babel : ถูก Setup มาให้เรียบร้อยแล้ว ซึ่งการเขียน React บาง Feature ยังจำเป็นต้องพึ่งตัว Babel อยู่ เช่น JSX, Object Rest/Spread
- ESLint : Config มาให้แล้ว เผื่อเอาไว้เช็ค Syntax ของโค๊ดเราว่าเขียนถูกต้องตาม Style Guide หรือไม่
- Jest : เป็นตัว Test Library ของทาง Facebook built in มาให้เรียบร้อย
- Autoprefixer CSS : ไม่ต้องไปนั่งใส่พวก Vendor prefix เช่น -moz-, -webkit-
- PWA (Progressive Web Application) : ใส่ Service Worker แล้วก็ Web Manifest มาให้เลย

เริ่มต้นเขียน React

เราจะเริ่มต้นเขียน React ด้วยการใช้งาน Create React App กัน วิธีนั้นง่ายมากๆ เปิด Terminal ขึ้นมา สำหรับ Tutorial นี้ผมตั้งชื่อ App ให้มันว่า hello-react

สำหรับ npm version 5.2+ ใช้คำสั่งนี้ :

```
npx create-react-app hello-react
```

และ สำหรับ npm เวอร์ชั่นต่ำกว่า

```
npm install create-react-app -g
create-react-app hello-react
```

npx เป็นเหมือน version upgrade ของ npm หากใช้ npm version 5.2+ จะมีติดมาอยู่แล้ว

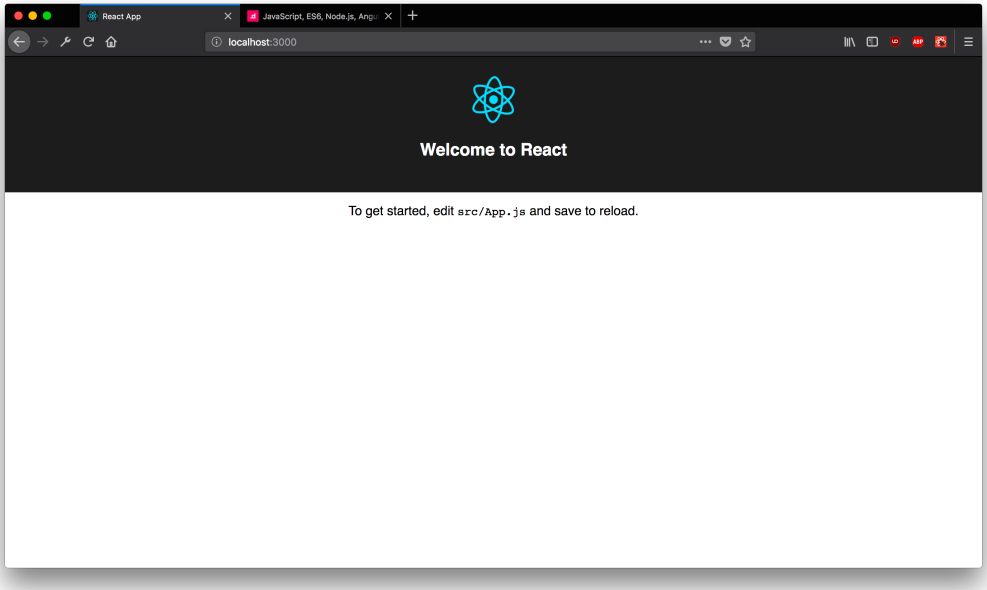
หลังจากนั้น เราก็จะได้โปรเจ็คที่เป็น React ขึ้นมา

```
cd hello-react
yarn start

# หรือหากใช้ npm :
npm start
```

เมื่อรันแล้ว มันจะเปิดหน้าเว็บเราให้อัตโนมัติ โดย default แล้วจะเป็น http://localhost:3000[8] หน้าตาเว็บเราก็จะเป็นแบบนี้

[8] http://localhost:3000



Hello React

ที่นี้กลับมาดูที่ตัวโปรเจ็คเรา ทำการเปิดด้วย Text Editor หรือ IDE ที่เราถนัดเลย ซึ่งไฟล์ที่สำคัญๆจะประกอบไปด้วย

- public/index.html : คือไฟล์ html หลักของเรา
- src/index.js : เป็นไฟล์หลักของ React
- src/App.js : ไฟล์สำหรับ Component ที่ชื่อว่า App

ซึ่งเมื่อเปิดดูไฟล์ App.js จะมีโค๊ดด้านล่างดังนี้

App.js

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

class App extends Component {
  render () {
    return (
      <div className='App'>
        <header className='App-header'>
          <img src={logo} className='App-logo' alt='logo' />
          <h1 className='App-title'>Welcome to React</h1>
        </header>
        <p className='App-intro'>
          To get started, edit <code>src/App.js</code> and save t\
o reload.
        </p>
      </div>
    )
  }
}

export default App
```

- import { Component } from 'react' : คือการ Import module จะมีทั้ง import default และ import module, See more : ES6 : import[9]
- export default App : คือการ export ตัว Component App เพื่อเอาไปใช้ที่ไฟล์อื่น See more : ES6 export[10]

[9] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/import
[10] https://developer.mozilla.org/en-US/docs/web/javascript/reference/statements/export

Note : สำหรับบทความผมใช้ Style Guide แบบ Standard[11] เลยไม่มี ; ต่อท้าย ไม่ต้องตกใจไปครับ

เราจะเห็นว่า ด้านบนมีส่วนที่คล้ายๆ HTML เลย นั่นก็คือตรงส่วนที่อยู่ใน render() ซึ่งมันก็คือการ render HTML นั่นแหละ เพียงแต่ render ในรูปแบบ JSX Syntax ที่สามารถใส่โค๊ด JavaScript ลงไปใน HTML ได้ เช่น

```
<img src={logo} className="App-logo" alt="logo" />
```

จะเห็นว่าเราสามารถใส่ script ลงไปใน HTML ได้ ผ่าน {}

และ App ก็คือ Component ของเราเอง ที่ extend มาจาก React.Component อีกที ซึ่งเจ้า Component กฎการสร้างก็มีเพียงแค่นี้ (คือการ extends Component และมี function render() เพื่อ return เป็น HTML Element กลับมา)

และดูที่ไฟล์ /src/index.js ซึ่งเป็นไฟล์ที่จะใช้รัน React

/src/index.js

```
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()
```

จะเห็นว่ามีส่วน ReactDOM.render(<App />, document.getElementById('root')) มันเป็นการบอกว่า หา element ที่มี id ชื่อว่า root จากนั้นก็ทำการ render React Component โดยในที่นี่ทำการ render ตัว App ญึ่ง id=root เราได้ประกาศไว้ในไฟล์ public/index.html นั่นเอง

ที่นี้เราลองมาดูตัวอย่างการใช้ Component ร่วมกับ HTML ดู สมมติเราสร้าง Component ใหม่ และตั้งชื่อว่า HelloApp.js

[11] https://standard.js

HelloApp.js

```
import React from 'react'

class HelloApp extends React.Component {
        render() {
                return <h1>Ahoy! React</h1>
        }
}

export default Hello
```

ทำการแก้ไขไฟล์ src/App.js โดยการเพิ่ม นี้ลงไป

src/App.js

```
import HelloApp from './HelloApp'
```

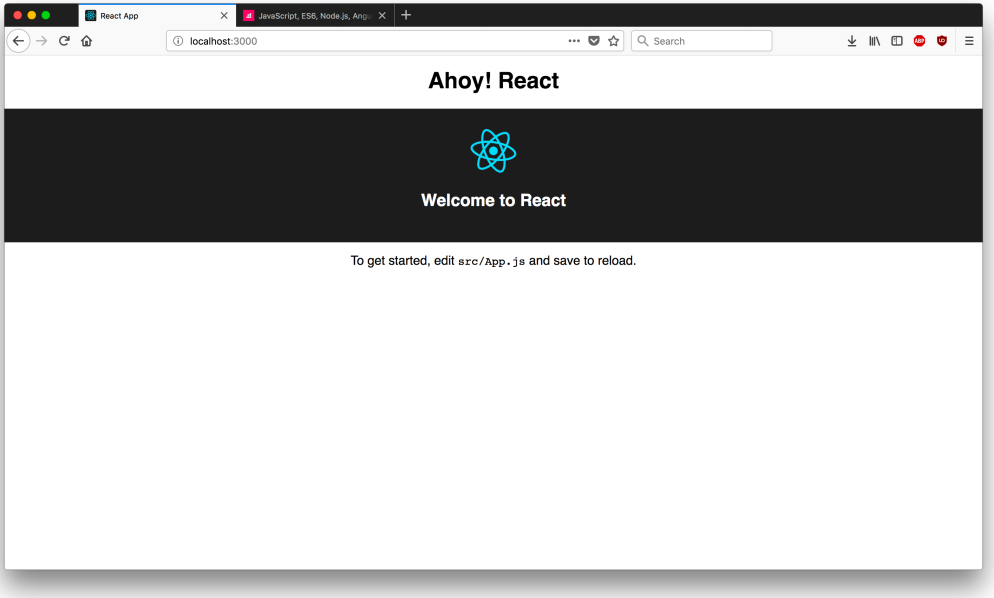
และตรงส่วน render() ก็ทำการเพิ่ม <HelloApp /> ลงไป

src/App.js

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

```
    </p>
   </div>
  )
}
```

ทีนี้เมื่อลองดูหน้าเว็บอีกครั้ง เราจะเห็น Ahoy! React และก็ไม่จำเป็นต้อง refresh browser ด้วย ตัว Create React App นั้น built in ทุกอย่างมาให้เราแล้ว สนใจแค่โค๊ดก็พอ :)



Hello React 2

รู้จักกับ Props & State

ต่อมา เรามาพูดถึงเรื่องถัดไปของ React กัน นั้นก็คือเรื่องของ Props และ States กัน หากเราเขียน React ก็หนีไม่พ้นเจ้าสองตัวนี้แน่นอน ยังไงก็ต้องเจอ มาเริ่มเลย

Props

Props หรือชื่อเต็มๆมันคือ Properties หากเปรียบกับ HTML แล้ว ตัว Props จะเป็นคล้ายๆ attributes ของ HTML ดัง เช่น src, href หรือ class

```
<img src="/path/to/image.png" />
<a href="#" class="my-link">Click!</a>
```

ซึ่ง Props ใน React ข้อดีคือ เราสามารถส่งข้อมูลจาก Component หนึ่งไปอีก Component ได้ด้วยการใช้ Props นี่แหละ

เราลองกลับมาที่ Component App.js ที่เราเรียก HelloApp เราลองทำการส่ง props ไปกับ Component ด้วยแบบนี้

```
<HelloApp message="This is message sent from App" />
```

และที่นี่ที่ไฟล์ HelloApp.js เราก็ต้องทำการเพิ่ม

HelloApp.js

```
import React from 'react'

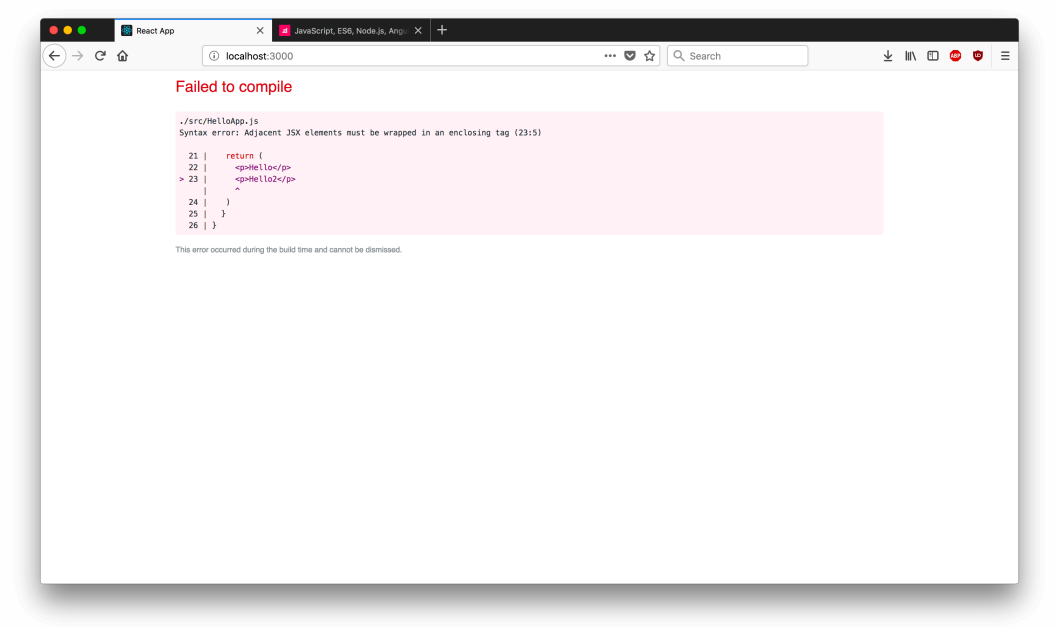
class HelloApp extends React.Component {
  render() {
    return (
      <div>
        <h1>Ahoy! React</h1>
        <p>{this.props.message}</p>
      </div>
    )
  }
```

```
}

export default HelloApp
```

ซึ่งจะเห็นได้ว่า render() นั้น return ค่าหลายบรรทัด เราจะใช้ () ซึ่งจำเป็นต้อง wrap element ด้วย ซึ่งถ้าเรา return แบบด้านล่าง จะไม่สามารถ run ได้ เนื่องจาก JSX จะมองไม่เห็น closing tag

```
return (
  <p>Hello</p>
  <p>Hello2</p>
)
```



React Compile Error

แต่ถ้าเราต้องการ render แบบนี้ละ ไม่อยากเอา div มาหุ้มอีกชั้นนึง? เราสามารถใช้ Fragment เข้ามาช่วยได้แบบนี้

```
return (
  <React.Fragment>
    <h1>Ahoy! React</h1>
    <p>{this.props.message}</p>
  </React.Fragment>
)
```

ก็จะไม่ error แล้ว

State

State เป็นค่าที่จะถูกใช้ ภายใน Component วิธีการใช้งาน State ทำการสร้าง Constructor ขึ้นมา จากนั้น set ค่า default state ด้วยคำสั่ง this.state = {}

```
import React from 'react'

class HelloApp extends React.Component {
  constructor(props) {
    super(props)

    this.state = {
      counter: 0
    }
  }

  render() {
    return (
      <React.Fragment>
        <h1>Ahoy! React</h1>
        <p>{this.props.message}</p>
        <buttonClick me!</button>
```

```
        <p>Total click : {this.state.counter}</p>
      </React.Fragment>
    )
  }
}

export default HelloApp
```

ด้านบน เราได้ทำการเปลี่ยนแปลง HelloApp.js ใหม่

- โดยให้ counter เก็บไว้ใน state มีค่าเริ่มต้น 0
- ทำการ render ค่า counter ด้วย {this.state.counter}

เมื่อเราต้องการจะ set ค่าให้กับ state เราสามารถเรียกใช้ this.setState() ได้แบบนี้

```
this.setState({
  key: value
})
```

ตัวอย่าง ผมจะให้ เมื่อเราทำการ Click button จะให้มันทำการเพิ่ม counter ทีละ 1 โดยการใช้ setState() นั่นเอง ที่ <button>Click me!</button> ทำการเพิ่ม onClick เข้าไป

```
<button onClick={this.handleClick}>
```

- เมื่อมี event click จะไปเรียก method handleClick ซึ่งเรายังไม่ได้สร้าง ต่อมาสร้าง method handleClick

```
handleClick() {
  this.setState({
    counter: this.state.counter + 1
  })
}
```

แต่! เราจะยังไม่สามารถกดได้ เนื่องจากตัว button เวลาถูก click มันไปเรียก handleClick แต่ว่า this ในฟังค์ชั่นมัน refer ไปที่ button ไม่ใช่ Class มัน เราจึงต้องทำการ bind ใน Constructor ก่อน

```
constructor(props) {
  this.handleClick = this.handleClick.bind(this)
}
```

สุดท้ายโค๊ด HelloApp.js จะเป็นแบบนี้

HelloApp.js

```
import React from 'react'

class HelloApp extends React.Component {
        constructor(props) {
    super(props)

    this.state = {
            counter: 0
    }

    this.handleClick = this.handleClick.bind(this)
        }

        handleClick() {
          this.setState({
```

```
            counter: this.state.counter + 1
        })
    }

    render() {
        return (
      <React.Fragment>
        <h1>Ahoy! React</h1>
        <p>{this.props.message}</p>
        <button onClick={this.handleClick}>Click me!</button>
        <p>Total click : {this.state.counter}</p>
      </React.Fragment>
        )
    }
}

export default HelloApp
```

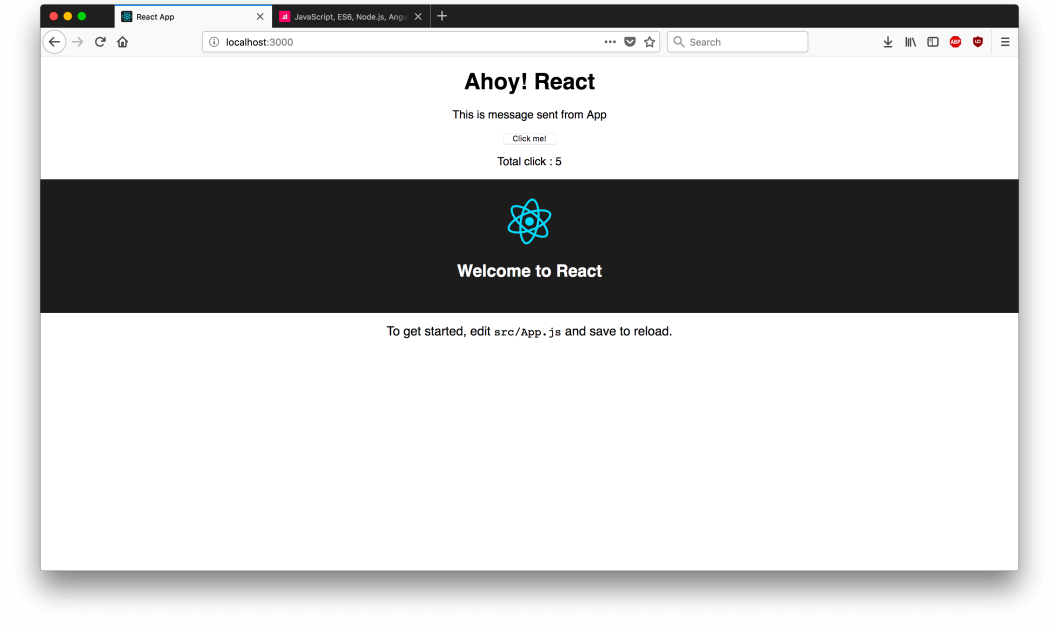
แต่ว่า (อีกแล้ว) เนื่องจากเวลาเรา setState() ถ้าเกิดว่ามีการอัพเดท state เดิม มันจะมี side effect บ้างในบางครั้ง ทาง React แนะนำว่า ให้ใช้การ setState() แบบ function แทน ซึ่ง function ก็มี 1 argument เป็น state ก่อนหน้า และ return เป็น new state เช่น

```
this.setState(function(prevState) {
  return {
    counter: prevState.counter + 1
  }
})
```

แปลงด้านบนเป็น ES6 ให้สวยๆดีกว่า

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

ที่นี้ลองกลับไปดูแล้วลอง คลิ๊ก Button จะเห็นว่า ค่าถูกอัพเดทตามที่เรากดเลย



Counter Clikc

สรุประหว่าง Props และ States

- Props : จะเป็นการส่งข้อมูลข้าม Component
- State : ข้อมูลจะถูกใช้ภายใน Component ตัวเอง
- State : เมื่อ State มีการเปลี่ยนแปลง Component จะทำการ render() ใหม่

มารู้จักกับ Stateless Component กัน

ต่อมามาดูการสร้าง Component อีกแบบนึง ที่เรียกว่า **Stateless Component** คือ Component ที่ไม่มี State

เอ๊ะ เมื่อกี้เพิ่งพูดถึง State, setState() ไป คราวนี้ไม่มี State แล้ว ?

จริงๆ อาจจะมีบาง Component ที่เราต้องการสร้าง ไม่จำเป็นต้องใช้ State ต้องการรับค่า Props มาแล้วก็ทำการ Render อย่างเดียว เราก็ไม่จำเป็นต้อง extends React.Component แถมพ่วงมาด้วย Life Cycle ต่างๆ ของ React มาด้วย

วิธีการสร้าง Stateless Component คือจริงๆ ให้มองว่ามันคือ Function ธรรมดาก็ได้ เช่น

```
function Hello(props) {
  return <h1>Ahoy! {props.name}</h1>
}
```

ด้านบน เป็น function ที่รับ prop มา แล้ว return เป็น JSX กลับไป ทีนี้เราลองมาสร้าง Stateless ใน HelloApp.js ดู เพิ่งฟังค์ชั่น ด้านบน เข้าไปก่อนบรรทัด class HelloApp

```
function Hello(props) {
  return <h1>Ahoy! {props.name}</h1>
}

class Hello App extends React.Component {
 ...
}
```

และส่วน render() ก็แค่เรียกใช้ Component Hello พร้อมส่ง props ที่ชื่อว่า name ไปด้วย แบบนี้

```
<Hello name="Chai" />
```

ข้อดีของ Stateless Component คือ

- ไม่ต้อง extends Class ให้ยุ่งยาก
- ไม่ต้องใช้ this (ไม่ต้อง bind ใน Constructor) แล้ว เช่น this.props.name ก็ใช้ props.name ได้เลย หรือ {onClikc={handleClikc}

ตัวอย่าง Stateless Component อีกอัน ลองสร้างขึ้นมาชื่อ MyStatelessComponent.js

src/MyStatelessComponent.js

```
import React from 'react'

const MyStatelessComponent = (props) => (
  <div>
    <h1>{props.title}</h1>
    <p>{props.message}</p>
  </div>
)

export default MyStatelessComponent
```

หรือสามารถ เขียน MyStatelessComponent ใหม่อีกแบบ ด้วย ES6 Destructuring ดังนี้

src/MyStatelessComponent.js

```
import React from 'react'

const MyStatelessComponent = ({ title, message }) => (
  <div>
    <h1>{title}</h1>
    <p>{message}</p>
  </div>
)

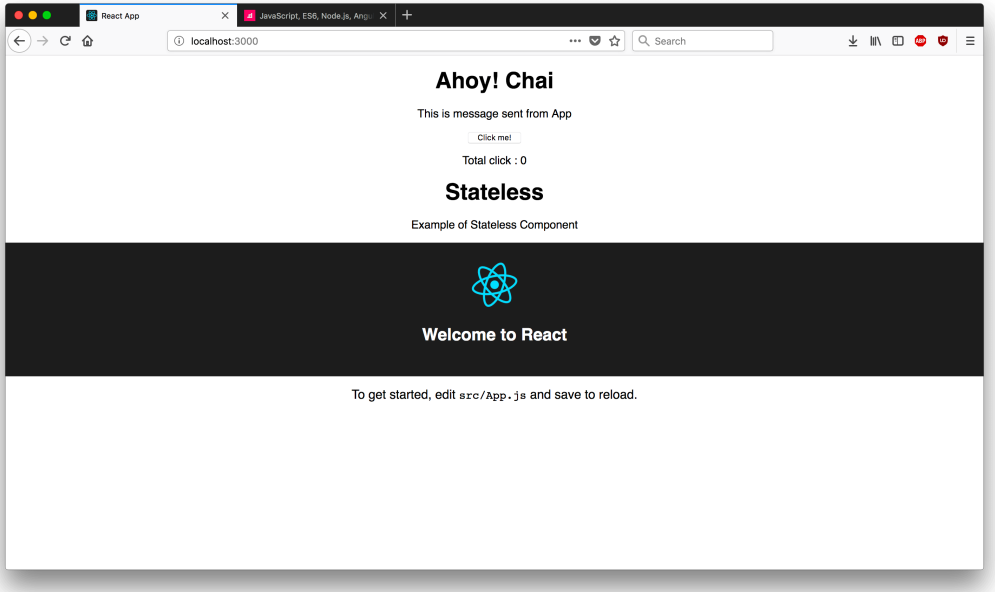
export default MyStatelessComponent
```

แล้วที่ไฟล์ App.js ก็ลองทำการเรียกใช้แบบนี้

src/App.js

```
import MyStatelessComponent from './MyStatelessComponent'


render() {
  <MyStatelessComponent
    title="Stateless"
    message="Example of Stateless Component"
  />
}
```



React Stateless

สรุป Episode I

สำหรับ Episode I ก็จะเป็น Guide แบบ Basic สำหรับผู้เริ่มต้นหัดเขียน React โดยจะใช้เจ้า Create React App เข้ามาช่วยทำให้เราสามารถขึ้นโปรเจ็คใหม่ได้ง่ายๆ ไม่ต้องไปเรียนรู้ Webpack, Babel ให้ยุ่งยาก (เอาไว้ให้เขียน React ได้คล่องแล้วค่อยไปหัดใช้ หัด Custom ก็ไม่สายครับ)

สุดท้าย Source Code ของ Episode I Source Code: hello-react[12]

[12] https://github.com/Devahoy/hello-react

Episode II : React Router

มาต่อด้วย Episode II กันเลย สำหรับ Episode นี้จะมาพูดถึงการทำ Routing ด้วยการใช้ React Router](https://reacttraining.com/react-router/) กัน เพื่อเอามาทำเว็บไซต์ที่มีหลายๆหน้า การ render แต่ละหน้า การ navigate ไปหน้าอื่นๆ นั้นทำยังไงบ้าง ซึ่งเนื้อหาทั้งหมด ก็ตามด้านล่างเลย

เริ่มต้นกับ React Router

เริ่มต้นสร้างโปรเจ็ค React ด้วย create-react-app ได้เลย

```
npx create-react-app basic-web-react-router

cd basic-web-react-router
yarn
```

สำหรับบทความพื้นฐาน React สามารถอ่านเพิ่มเติมได้ที่ : มาเริ่มต้นเขียน React ด้วย Create React App กันดีกว่า[13]

เมื่อได้โปรเจ็ค React มาแล้ว ต่อมาผมจะทำการ Format Code เป็น Standard Style (คือลบพวก comma) จาก Project ที่สร้างจาก create-react-app สำหรับใครที่ไม่ต้องการ format อยากได้แบบเดิม ก็ข้ามขั้นตอนนี้ได้เลยครับ

1. เพิ่มไฟล์ .eslintrc

[13] https://devahoy.com/posts/learn-react-with-create-react-app/

.eslintrc

```
{
  "extends": ["react-app", "standard"],
  "rules": {
    "space-before-function-paren": 0
  }
}
```

2. ทำการเพิ่ม dependencies ต่างๆ

```
yarn add prettier standard eslint eslint-config-react-app eslint-\
config-standard eslint-plugin-flowtype eslint-plugin-import eslin\
t-plugin-jsx-a11y eslint-plugin-node eslint-plugin-promise eslint\
-plugin-standard
```

3. Add script

ทำการเพิ่ม scripts ไปที่ไฟล์ package.json เพื่อเวลาสั่งรัน prettier จะให้มันทำการ format code โดยไม่ใช้ semicolon และ string แบบ single quote

package.json

```
scripts: {
  "prettier": "./node_modules/.bin/prettier --semi=false --single\
-quote=true --write src/*.js"
}
```

4. Run prettier

รันคำสั่ง Prettier ด้วย

yarn prettier

รู้จักกับ React Router

ขั้นตอนต่อมา ทำการติดตั้ง React Router กันเลย

yarn add react-router-dom

จากนั้นเปิดไฟล์ src/index.js จากโค๊ดด้านล่าง

src/index.js

```
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()
```

ทำการแก้ไขโค๊ดเป็นแบบนี้

src/index.js

```
import React from 'react'
import ReactDOM from 'react-dom'
import './index.css'
import App from './App'
import registerServiceWorker from './registerServiceWorker'
import { BrowserRouter } from 'react-router-dom'

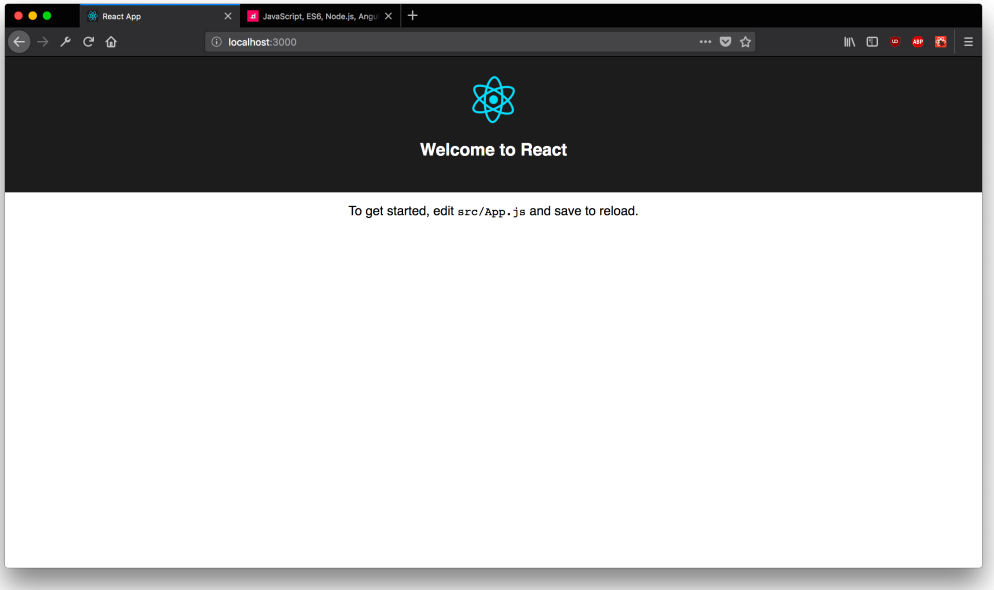
const AppWithRouter = () => (
  <BrowserRouter>
    <App />
  </BrowserRouter>
)

ReactDOM.render(<AppWithRouter />, document.getElementById('root'\
))
registerServiceWorker()
```

จากโค๊ด จะเห็นว่าเรามีการเพิ่ม BrowserRouter จาก react-router-dom และทำการใส่ไว้ On Top ของ Component ทำการหุ้ม Component App เราอีกชั้นนึงก่อนที่จะทำการ render

- ควรใช้ BrowserRouter : ถ้าเรามี Server เอาไว้ serve พวก url
- ควรใช้HashRouter : ถ้า Server เราเป็นแบบ serve static file เฉยๆ

สังเกตตอนนี้ Web เราก็ยังรันได้ปกติ อาจจะต้องมี stop แล้ว yarn start อีกครั้ง เพราะว่ามันอาจจะมองไม่เห็น react-router-dom ที่เพิ่งติดตั้งลงไป



Hello React

รู้จักกับ Route & Switch

มาถึงขั้นตอนการทำ Route นะครับ ตอนนี้ผมจะใส่ Bulma[14] เข้ามา เพราะว่าเป็น Stylesheet ที่จะใช้ในบทความนี้ ที่ไฟล์ public/index.html เพิ่ม Bulma และ Font เข้าไป

public/index.html

```
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/li\
bs/bulma/0.6.2/css/bulma.min.css" />
<link href="https://fonts.googleapis.com/css?family=Quicksand:300\
,700" rel="stylesheet">
```

เพิ่มไฟล์ src/index.scss

[14] bulma.io/

src/index.scss

```
body {
  margin: 0;
  padding: 0;
  font-family: 'Quicksand', sans-serif;
}

.App {
        display: flex;
        flex-direction: column;
        justify-content: center;
        align-items: center;
}
```

ต่อมาที่ไฟล์ src/App.js ทำการแก้ไขไฟล์เป็นแบบนี้

src/App.js

```
import React, { Component } from 'react'
import { Route } from 'react-router-dom'

const Home = () => <h1>Home</h1>
const About = () => <h1>About</h1>
const Post = () => <h1>Post</h1>
const Project = () => <h1>Project</h1>

class App extends Component {
  render() {
    return (
      <div className="App container">
        <Route path="/" component={Home} />
        <Route path="/about" component={About} />
```

```
        <Route path="/posts" component={Post} />
        <Route path="/projects" component={Project} />
      </div>
    )
  }
}

export default App
```

อธิบายเพิ่มเติม

```
const Home = () => <h1>Home</h1>
```

- เป็นการสร้าง Component ที่ชื่อ Home ทำการ render คำว่า Home นั่นเอง

```
const About = () => <h1>About</h1>
```

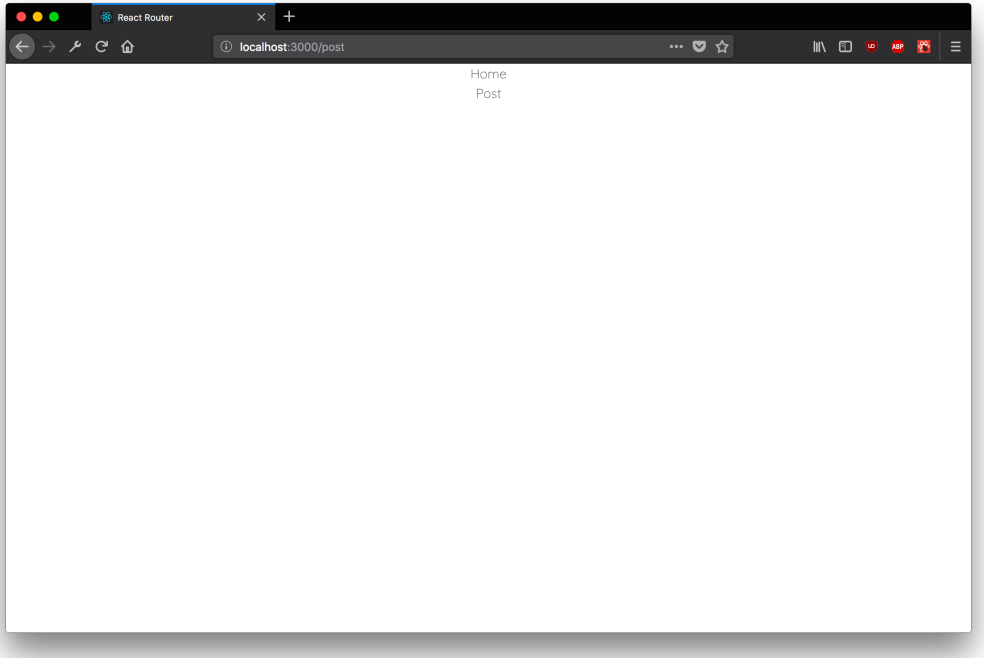
- สร้าง Component ชื่อ About และก็เหมือนกันกับ Post และ Project

และส่วน

```
<Route path="/projects" component={Project} />
```

- path : เป็นการบอกว่า ถ้า URL มี path = /project จะให้มัน render component ชื่อ Project

ทีนี้เราลองเข้าเว็บใหม่ ทีนี้ลองเข้าผ่าน URL http://localhost:3000/posts[15] หรือ http://localhost:3000/about[16]



React Router

ใช้งาน Route

อยากที่บ้านบนอธิบายไปแล้ว ตัว Route ถือเป็นส่วนนึงที่สำคัญของ React Router เลยก็ว่าได้ เป็นส่วนที่เราจะเอาไว้กำหนดว่า URL location ที่เราเข้ามาผ่านทาง Browser นั้นตรงกันกับที่เราประกาศไว้ใน Route หรือไม่ ถ้าตรง ก็จะทำการ render ตัว Component ที่เราได้ทำการกำหนดไว้

ซึ่งนอกจาก props path หรือ component แล้ว ตัว Route ก็ยังมี props ที่สำคัญอีก อันนึงคือ exact

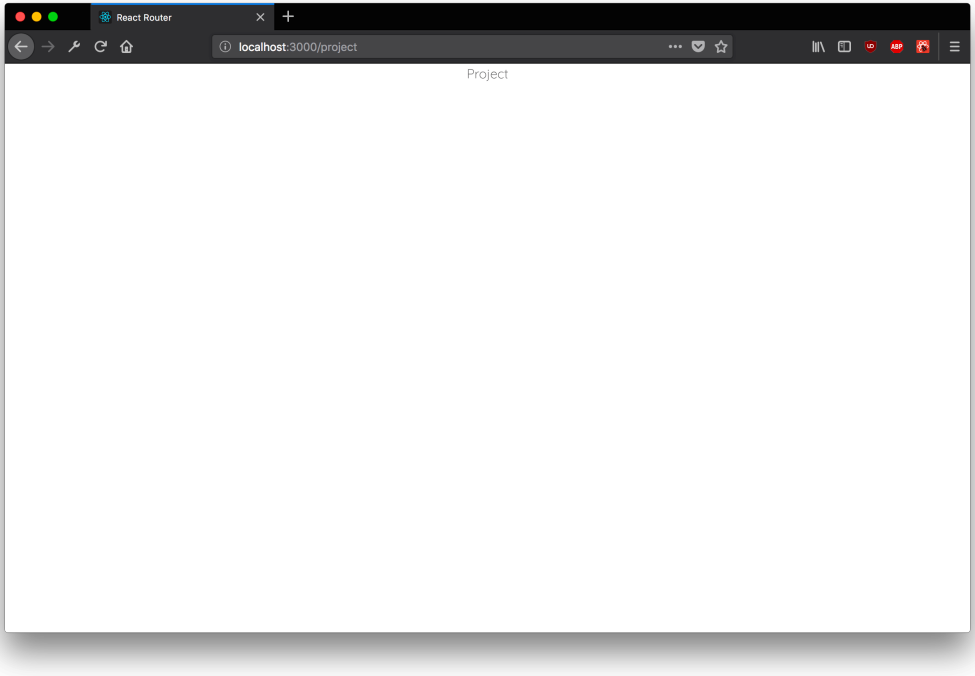
ซึ่งจากด้านบน จะเห็นว่า เวลาเราเข้า /project หรือว่า /about จะเห็นคำว่า **Project** และ **About** เวลาเข้าหน้าของมัน แต่ก็ดันมีคำว่า **Home** โผล่มาด้วย นั่นก็เพราะว่า <Route path="/"> นั้นมัน match ทั้ง 2 path ครับ (คือมี / ใน url ไม่สนว่าต่อท้ายเป็นอะไร มัน match อะ)

[15] http://localhost:3000/posts
[16] http://localhost:3000/about

วิธีการแก้ไขคือ ปรับเป็นแบบนี้

```
<Route exact path="/" component={Home} />
```

เป็นการบอกว่า ทีนี้ ต้อง match แบบว่า http://localhost:3000/ ยย่างเดียวนะ ถึงจะ render Home ถ้ามี เช่น /projects ถือว่ามันไม่ match ลองดูผลลัพธ์ที่หน้าเราอีกที



React Router

ใช้งาน Switch

ต่อมา Switch ตัวนี้จริงๆ ทำงานคล้ายๆ Route แต่ต่างกันที่ถ้าเป็น Switch เมื่อมัน match กับ location ไหนแล้ว มันก็จะ render ตัวนั้นไปเลย ตัวอื่นๆ จะไม่สน โดยไล่จากบนลงล่าง เช่น

```
import { Switch } from 'react-router-dom'

render() {
  <Switch>
    <Route path="/" component={Home} />
    <Route path="/about" component={About} />
    <Route path="/posts" component={Post} />
    <Route path="/projects" component={Project} />
  </Switch>
}
```

ซึ่งส่วนใหญ่แล้ว Switch ผมจะใช้เอาไว้ในบล็อค ที่มันต้อง render อย่างน้อย 1 Component เช่น บางทีจะวาง Route ที่เป็น Error 404 เอาไว้ เช่น ถ้าไม่เจอ location อะไรเลย ก็ให้มัน render ตัวนี้ไป เช่น

```
<Switch>
  <Route path="/" component={Home} />
  <Route path="/about" component={About} />
  <Route path="/posts" component={Post} />
  <Route component={NotFoundPage} />
</Switch>
```

การทำ Navigation

ต่อมาจะเห็นว่าเรากำหนด Route เรียบร้อยแล้ว ทีนี้จะเข้าไปแต่ละหน้า ต้องมานั่งพิมพ์ URL เอาหรอ? ไม่สะดวกแน่นอน ทำไมเราไม่ใช้ <a href="#"> ละ เพื่อจะ link ไปหน้า อื่นๆ

ก็ทำการเพิ่ม <a> กันเลย ที่ไฟล์ src/App.js ผมทำการเพิ่ม Navbar ของ Bulma ลงไปด้วย ก็เลยได้ประมาณนี้

src/App.js

```
import React, { Component } from 'react'
import { Route } from 'react-router-dom'

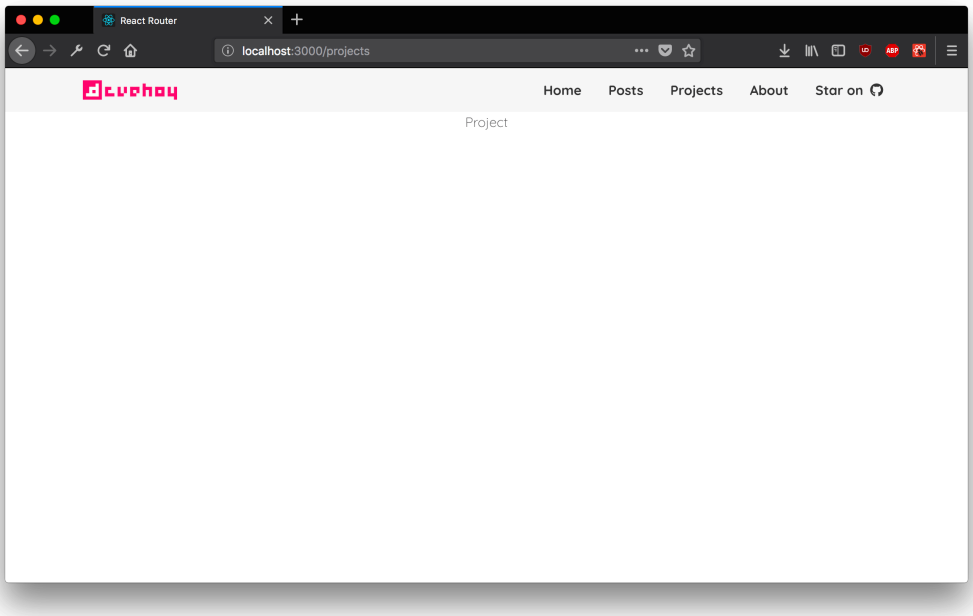
const Home = () => <h1>Home</h1>
const About = () => <h1>About</h1>
const Post = () => <h1>Post</h1>
const Project = () => <h1>Project</h1>

class App extends Component {
  render() {
    return (
      <div className="my-app">
        <nav className="navbar is-light" role="navigation" aria-l\
abel="main navigation">
          <div className="container">
            <div className="navbar-brand">
              <a className="navbar-item" href="https://devahoy.co\
m">
                <img src="https://devahoy.com/assets/images/devah\
oy-text-logo.png" alt="DEVAHOY LOGO" width="112" height="28" />
              </a>
            </div>
            <div className="navbar-menu">
              <div className="navbar-end">
                <a href="/" className="navbar-item">Home</a>
                <a href="/posts" className="navbar-item">Posts</a>
                <a href="/projects" className="navbar-item">Proje\
cts</a>
                <a href="/about" className="navbar-item">About</a>
                <a class="navbar-item" href="https://github.com/p\
honbopit" target="_blank">Star on <i className="fab fa-github"></\
```

```
i></a>
              </div>
            </div>
          </div>
        </nav>
        <div className="App container">
          <Route exact path="/" component={Home} />
          <Route path="/about" component={About} />
          <Route path="/post" component={Post} />
          <Route path="/project" component={Project} />
        </div>
      </div>
    )
  }
}

export default App
```

ทีนี้เวลาเรากด Link บน Navbar ก็จะ เปลี่ยนหน้า และ match location ที่เราต้องการได้



React Router

แต่!!

จะเห็นว่าทุกๆครั้งที่เรากด Link มันจะทำการ refresh หน้าใหม่ทุกครั้งเลย ไม่ใช่ตาม concept SPA ที่จะ route โดยไม่ต้อง refresh วิธีแก้ก็คือการใช้ Link ครับ

Link

ตัว Link จะเป็นเหมือนกับ <a> เพียงแต่ว่าจะเป็นของ React Router และเปลี่ยนจาก href เป็น to แทน เช่น

```
<Link to="/projects">Projects</Link>
```

เราลองมาเปลี่ยน <a href=""> ทั้งหมด เป็น Link ก็จะได้เป็นแบบนี้

```
<div className="navbar-end">
  <Link to="/" className="navbar-item">Home</Link>
  <Link to="/posts" className="navbar-item">Posts</Link>
  <Link to="/projects" className="navbar-item">Projects</Link>
  <Link to="/about" className="navbar-item">About</Link>
  <a className="navbar-item" href="https://github.com/phonbopit" \
target="_blank">Star on <i className="fab fa-github"></i></a>
</div>
```

ลองทำการกดใหม่ ทีนี้หน้าเราจะไม่ Refresh แล้ว เป็นอันเรียบร้อย

ใช้งาน NavLink

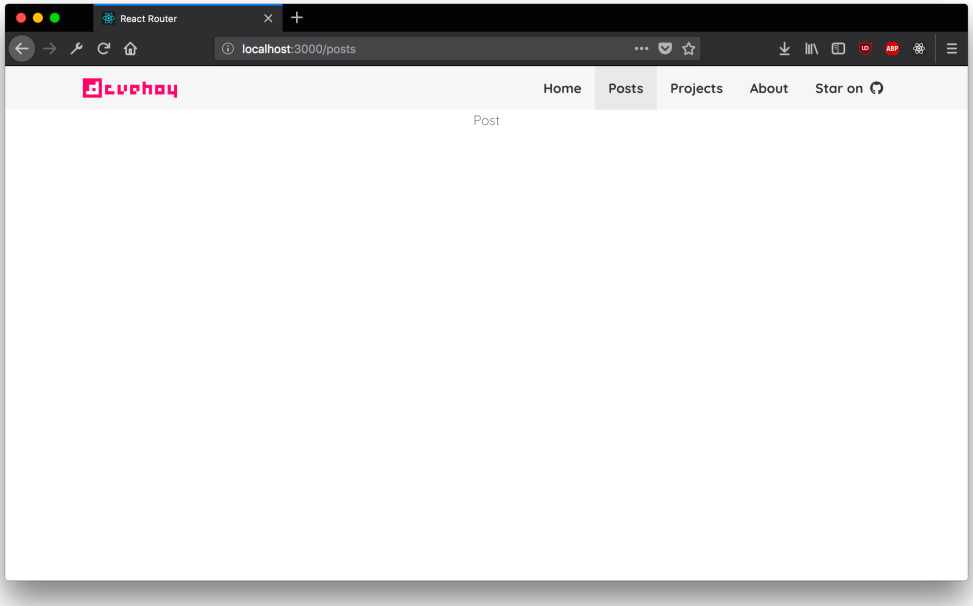
ต่อมา NavLink ตัวนี้จริงๆก็เหมือนกับ Link เลย เพียงแต่ว่ามันสามารถกำหนด active style, active class ให้กับ Link ได้ เช่น ถ้ามัน location match กัน ก็จะให้มันเป็น class is-active อะไรอย่างงี้

ตอนนี้ตัว Bulma[17] มันจะมี helper class ที่เอาไว้ highlight กรณีที่ active ก็คือคลาส is-active ผมก็เลยใส่กรณีถ้า match location เลยได้เป็นแบบนี้

```
<div className="navbar-end">
  <NavLink exact to="/" activeClassName="is-active" className="na\
vbar-item">Home</NavLink>
  <NavLink to="/posts" activeClassName="is-active" className="nav\
bar-item">Posts</NavLink>
  <NavLink to="/projects" activeClassName="is-active" className="\
navbar-item">Projects</NavLink>
  <NavLink to="/about" activeClassName="is-active" className="nav\
bar-item">About</NavLink>
  <a className="navbar-item" href="https://github.com/phonbopit" \
target="_blank">Star on <i className="fab fa-github"></i></a>
</div>
```

17

Note: <NavLink /> หรือ <Link /> เราสามารถใส่ exact ได้แบบเดียวกับ <Route />

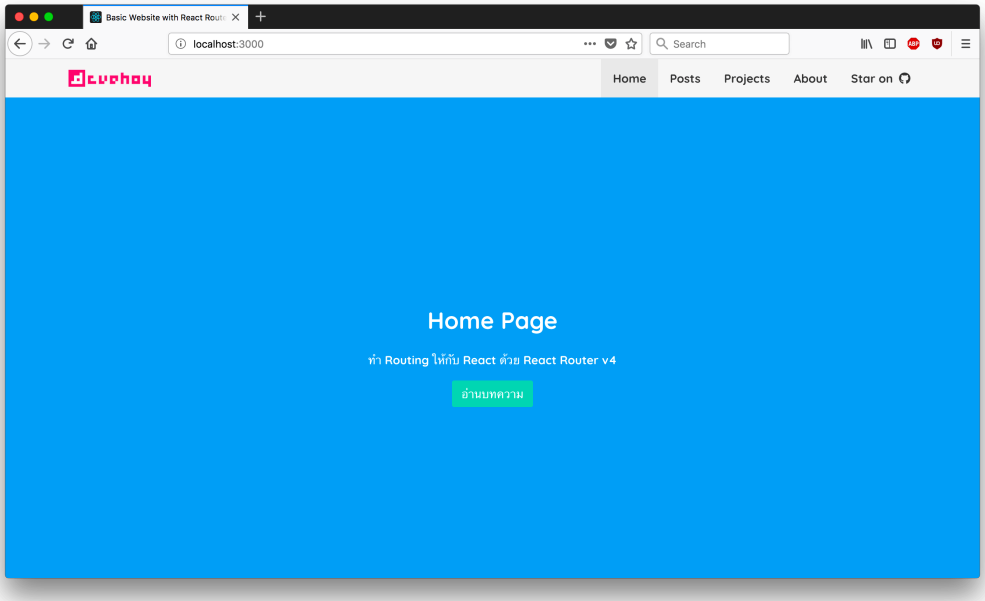


React Router step 4

สุดท้าย ผมทำการย้ายพวก Component ที่ประกาศไว้ใน src/App.js ไปแยกไว้แต่หน้าไฟล์ ได้เป็น

- src/pages/About/index.js
- src/pages/Project/index.js
- src/pages/Home/index.js
- src/pages/Post/index.js

และก็ได้เพิ่มเนื้อหาใส่ไปเล่นๆ ให้หน้าแต่ละหน้าดูมีอะไรนิดหน่อย สุดท้ายก็ได้เว็บเรียบๆง่ายๆ แบบนี้ครับ



React Router Final

สรุป Episode II

สรุปบทความนี้ก็เป็นตัวอย่างการใช้งาน React Router แบบง่ายๆ ยังไม่ได้ประยุกต์อะไรที่มันซับซ้อนมากนัก ลองนำไปใช้กันดูครับ ซึ่งก็ลองไปดูการทำ Route แบบ Authentication หรือว่าการ nested routing กันดู

สุดท้าย Source Code ของ Episode II Source Code: hello-react[18]

[18] https://github.com/Devahoy/basic-web-with-react-router

Episode III : React Redux

สำหรับ Episode สุดท้ายของ React ฉบับไตรภาค :) จะมาพูดถึง Redux กันครับ ซึ่งก่อนไปเริ่มต้นรู้จักกับ Redux ให้ทุกคนลองนึกภาพตามดูนะครับ ว่าปกติเราเขียน React ในการจัดการกับ State และ Props นั้นใน Application ขนาดใหญ่ เราจะมีวิธีจัดการพวกนี้ยังไง?

ถ้า App เรายังเล็กๆ มีไม่กี่ Component มันก็ยังไม่ยุ่งยากมาก แต่มันก็จะเริ่มเกิดปัญหาละ เมื่อเราอยากจะแชร์ State หรือส่งค่า State ข้ามไปมาระหว่าง 2 Components หรือทั้ง App เลยจะทำไง ต้องส่งผ่าน props ไปให้แต่ละ Component กันเลยหรอ ฟังดูยุ่งยากมากเลย ก็เลยทำให้เป็นที่มาว่าทำไม Redux จึงเกิดมา

Redux คืออะไร?

Redux ถ้าความหมายตามเว็บมันเลยคือ **“Predictable state container for JavaScript apps”** อ่านแล้วงงมั้ย □ จริงๆแล้ว แบบง่ายๆเลยมันคือ State Management (ตัวจัดการ State) นั่นเอง และมันก็ไม่จำเป็นว่าจะต้องใช้กับ React เท่านั้นนะครับ มันเป็น Concept ฉะนั้น มันสามารถประยุกต์ใช้ได้หมด เช่น Angular หรือ Vue

3 Principles of Redux

ก่อนอื่นเลย มารู้จักกับ 3 Principles ของ Redux กันก่อนว่ามีอะไรบ้าง?

1. Single Source of Truth

The state of your whole application is stored in an object tree within a single store.

กล่าวคือ State ของ Application เราต้องเก็บไว้ใน store เดียว โดยเป็น JavaScript Object ธรรมดานี่แหละ (ซึ่ง State ก็เปรียบเสมือน Data หรือ ก้อน Model ของเรานั่นเอง)

ตัวอย่าง state ที่เก็บอยู่ภายใน store

```
console.log(store.getState())

/*
{
  user: {
    id: 1,
    name: 'Chai'
  },
  posts: [{
    id: 1,
    title: 'Article #1'
  }, {
    id: 2,
    title: 'Article #2'
  }]
}
*/
```

2. State is read-only

ไม่ได้แปลว่า State จะอ่านค่าได้อย่างเดียว แต่ห้ามแก้ไขนะ แต่หมายถึงว่า เราจะอัพเดท State ได้ผ่านทาง Action ทางเดียวเท่านั้น

ตัวอย่างเช่น

```
store.dispatch({
  type: 'ADD_SCORE',
  score: 2
})

store.dispatch({
  type: 'RESET_SCORE'
})
```

dispatch เป็น function ที่ไว้บอก store ว่าเกิด action ขึ้นแล้วนะ ให้ state ทำการอัพเดทตาม type ที่กำหนดใน reducers ซะนะ

3. Changes are made with pure functions

การเปลี่ยนแปลง State ต้องเป็น Pure function เท่านั้น ก็คือ ใน reducers ของเราสามารถเปลี่ยน state ได้ แต่ไม่ใช่การแก้ไข state เดิม เป็นการส่งค่า state ใหม่กลับมาแทนครับ

ตัวอย่างเช่น

```
export default (state = 0, action) => {
  switch (action.type) {
    case 'INCREMENT':
      return state + 1
    case 'DECREMENT':
      return state - 1
    default:
      return state
  }
}
```

Reducers นี้จะทำการจัดการเกี่ยวกับ state สำหรับ ค่า โดยเริ่มต้นคือ 0 ซึ่งถ้ามี Action ที่มี type ว่า INCREMENT มันก็จะทำการเพิ่ม state ไปอีก 1 (เป็นค่า state ใหม่ไม่ได้แก้ไขค่าเดิม)

หรือถ้าหากว่า มันไม่เข้าเงื่อนไขทั้ง INCREMENT หรือ DECREMENT มันก็จะ return default state ซึ่งก็คือค่า 0 นั่นเอง

รู้จักกับ Actions

หน้าที่ของ Action คือ เป็นตัวบอกว่าข้อมูลนั้นคืออะไร จะทำอะไรกับมัน ซึ่งมอง Action เป็น JavaScript Object ธรรมดาเลย เพียงแต่มีข้อจำกัดนึงคือ ต้องมี key ชื่อว่า type ด้วย เช่น จะส่ง action สำหรับการสร้าง Post

```
const CREATE_POST = 'CREATE_POST'

store.dispatch(

 {
   type: CREATE_POST,
   payload: {
     title: 'Hello Redux',
     content: 'lorem ipsum'
   }
 }

)
```

รู้จักกับ Reducers

Reducers เปรียบเสมือนคล้ายๆ Controller ใน MVC คือจะเป็นตัวที่รับ Action มาเพื่อดูว่า Action นี้ต้องการจะทำอะไร จากนั้นก็ส่ง Action ไปหา Store นั่นเอง

- reducers : ต้องทำการ return Object ใหม่เท่านั้น จะไม่ทำการแก้ไข Object เดิม (เป็น Pure function)

ซึ่ง Reducer มันเป็นฟังค์ชัน ที่รับ previousState และ action แล้ว return newState กลับไป

```
(previousState, action) => newState
```

สิ่งที่ควรหลีกเลี่ยงใน Reducers

- อย่าเรียก non pure function เช่น new Date() หรือ Math.random()
- เปลี่ยนแปลงค่า argument ที่รับมาตรงๆ (mutation)

รู้จักกับ Store

Store เป็น Object ที่เอาไว้เก็บ State ของ Application และ 1 แอพมีเพียงแค่ 1 Store เท่านั้น ซึ่งเบื้องหลังของ Store นั้นมันทำอะไรบ้าง?

- เก็บ state ของแอพเราไว้ และสามารถเข้าถึงข้อมูลผ่าน getState() ได้
- สามารถ update state ได้ผ่าน store.dispatch(action)
- register listener ผ่านทาง subscribe(listener)

การสร้าง Store เราสามารถ สร้างผ่าน createStore() โดยใช้ reducers เป็น argument แบบนี้

```
import { createStore } from 'redux'
import reducers from './reducers'
const store = createStore(todoApp)
```

Implement Redux with react-redux

หลังจากพูดทฤษฎีกันไปแล้ว เอาจริงๆ เชื่อว่า 90% ไม่เข้าใจหรือยังจับต้นชนปลายไม่ถูกหรอก ว่า Redux มันคืออะไร ช่วยแก้ปัญหาอะไร ถ้ายังไม่ได้ลองลงมือทำ ลองเล่น ลองปรับๆแก้ๆดู ฉะนั้นมาลงมือทำ Redux จริงๆกันเลยดีกว่า

Step 1 : Setup & Installation

เริ่มจากสร้าง React Project ขึ้นนมาด้วย create-react-app ละกัน

```
npx create-react-app hello-react-redux
```

จากนั้นทำการ Add redux dependencies ครับ

```
yarn add redux react-redux
```

ซึ่งโปรเจ็คนี้จะเป็น Example ง่ายๆในการใช้ Redux โดยเป็น App Counter เพิ่ม ลบ ค่า

Step 2 : Add Provider & Store

ที่ไฟล์ index.js จาก

index.js

```
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();
```

ทำการเพิ่ม Provider เป็นแบบนี้ (เนื่องจากต้องหุ้มด้วย Provider เพื่อให้เราสามารถเข้าถึงค่า store ได้)

index.js

```
import { Provider } from 'react-redux'

const MyApp = () => (
  <Provider>
    <App />
  </Provider>
)

ReactDOM.render(<MyApp />, document.getElementById('root'))
```

ต่อมา ตัว Provider ต้องมี props store ด้วย ซึ่งเราจะได้มาจาก createStore ของ redux โดยใช้ reducers ของเราครับ

```
import { createStore } from 'redux'
import rootReducer from './reducers'

const store = createStore(rootReducer)

const MyApp = () => (
  <Provider store={store}>
    <App />
  </Provider>
)
```

ถึงตอนนี้ยังไม่สามารถรัน App ได้นะครับ เนื่องจากเรายังไม่ได้สร้าง reducers และ actions ให้แอพเราเลย

Step 3 : Reducers & Action

ต่อมาทำการสร้าง Reducers ของเราขึ้นมาครับ ชื่อ reducers/counters.js (ฟังค์ชันของ reducers จะรับ state และ action เป็น arguments)

reducers/counters.js

```
export default (state = 0, action) => {
  switch (action.type) {
    case 'INCREMENT':
      return state + action.score
    case 'DECREMENT':
      return state - action.score
    default:
      return state
  }
}
```

ต่อมาตรงส่วน rootReducers ให้ทำการสร้างไฟล์ reducers/index.js ขึ้นมาแบบนี้

reducers/index.js

```
import { combineReducers } from 'redux'
import counters from './counters'

export default combineReducers({
  counters
})
```

- combineReducers() : เป็นการรวม Reducers หลายๆตัวให้เป็น Reducers เดียวกัน

โดยที่ reducers/index.js จะเป็นเหมือนการรวม reducers ทั้งหมด ให้ state อยู่ใน store เดียวกันครับ ซึ่ง reducers 1 อันจะยังไม่เห็นผล แต่ถ้าใน Application จริงๆ เราจะได้ rootReducers แนวๆนี้ครับ

```
export default combineReducers({
  form,
  sessions,
  users,
  posts
})
```

ซึ่งใน State tree ก็จะเก็บเป็นก้อน Object ได้ลักษณะนี้ (ขึ้นอยู่กับ Reducers)

```
{
  form: {},
  sessions: {
    isAuthentication: false,
    currentUser: {}
  },
  users: [{...}],
  posts: [{...}]
}
```

และต่อมาเพิ่ม Action ที่ไฟล์ actions.js แบบนี้

actions.js

```
export const increment = (score = 1) => ({
  type: 'INCREMENT',
  score
})

export const decrement = (score = -1) => ({
  type: 'DECREMENT',
  score
})
```

Step 4 : Add Component

มาที่ไฟล์ App.js ทำการเปลี่ยนหน้าตาให้มันซักนิด โดยหน้านี้จะมีแค่โชว์ score และก็ Button ในการเพิ่ม/ลบ score ทีละ 1, 2 และ 3 ตามลำดับ

App.js

```
import React from 'react'
import { connect } from 'react-redux'

const App = ({ message, counter }) => (
  <div className="container">
    <div className="columns column is-12">
      <h1>Counter : {counter}</h1>
    </div>

    <div className="buttons">
      <button className="button is-primary">+1</button>
      <button className="button is-link">+2</button>
      <button className="button is-info">+3</button>
    </div>

    <div className="buttons">
      <button className="button is-primary">-1</button>
      <button className="button is-link">-2</button>
      <button className="button is-info">-3</button>
    </div>
  </div>
)

const mapStateToProps = function(state) {
  return {
    message: 'This is message from mapStateToProps',
```

```
    counter: state.counters || 0
  }
}

const AppWithConnect = connect(mapStateToProps)(App)
export default AppWithConnect
````
```

ใส่ stylesheet ด้วย Bulma ที่ไฟล์ `public/index.html` ซักนิด

{title=public/index.html, lang=html}
```html
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/li\
bs/bulma/0.7.1/css/bulma.min.css">
```

เราจะเห็นโค๊ดประหลาดๆอยู่บ้าง กำลังจะอธิบายในหัวข้อถัดไปครับ

connect()

ถ้าเราดูที่ไฟล์ App.js เราจะเห็นทั้ง connect() ทั้ง mapStateToProps แล้วมันคืออะไรล่ะ?

จริงๆแล้ว connect เป็น utility function ที่ช่วยให้เราสามารถเรียกใช้ dispatch() จาก store ได้เลย รวมถึง การ bind หรือ map ค่า state กับ props ต่างๆ ด้วย mapStateToProps และ mapDispatchToProps

ซึ่ง connect() เป็น HOC (Higher Order Component) : ฟังค์ชันที่รับ Component แล้ว return เป็น Component ที่มีดีกว่าเดิม เรียกว่าเป็นการ upgrade component ก็ได้

โดยเมื่อเราใช้ connect ในรูปแบบนี้
```

```
const MyComp = () => <h1>Hello World</h1>

connect()(MyComp)
```

หรือ ส่ง mapStateToProps ไปใน connect() แบบ App.js คือการเปลี่ยนจาก state ของ Application เป็น props ให้เอาไว้ใช้ภายใน App

ย้อนกลับไปเรื่อง Props & State เนาะ ปกติ เวลาเราส่งข้อมูล State ไปให้ Component นึง เราก็ต้องโยนผ่าน Props พอเป็น Redux ก็เหมือนกัน แต่ต่างกันที่ State จะไม่ได้ถูกเก็บไว้ใน Component ของใครของมันละ เป็น State กลางแทน แล้ว Component ไหนจะเข้าถึง State นั้นๆ ก็แค่ไปเรียก connect(mapStateToProps) เอา

ซึ่ง mapStateToProps มันก็เป็นเพียงแค่ function ที่รับค่า state ปัจจุบัน และ return เป็นก้อน Object ที่เราต้องการจะให้มันส่งไปหา Component ตัวอย่างง่ายๆ คือ

```
const mapStateToProps = function(state) {
  return {
    message: 'This is message from mapStateToProps',
    counter: state.counters || 0
  }
}

// หรือเขียนแบบ Arrow function
// const mapStateToProps = state => ({
//   message: 'This is message from mapStateToProps',
//   counter: state.counters || 0
// })

const App = ({ message }) => <h1>{message}</h1>
export connect(mapStateToProps)(App)
```

เมื่อเรียก Component

```
<App />
```

จะสามารถแสดงผลเป็น

```
<h1>This is message from mapStateToProps</h1>
```

โดยที่มีค่าเท่ากับเราส่ง Props แบบนี้

```
<App message="This is message from props" />
```

Step 5 : Dispatch an action

ต่อมาที่ปุ่ม Button ต่างๆ เราจะให้มัน store.dispatch(action) สิ่งที่เราต้องการ เช่น ถ้ากด +1 ให้มันส่ง action

```
{
  type: 'INCREMENT',
  score: 1
}
```

และถ้า +2 ก็จะเป็น

```
{
  type: 'INCREMENT',
  score: 2
}
```

และปกติ การ dispatch ก็แค่ dispatch object ด้านบนไปได้เลย แต่ที่นิยมทำกันคือสร้างเป็น actionCreator ขึ้นมามากกว่า ก็เหมือนอย่างที่เราได้สร้าง function ไว้ที่ไฟล์ actions.js แล้วครับ

```
export const increment = (score = 1) => ({
  type: 'INCREMENT',
  score
})
```

ทีนี้เวลาเรา store.dispatch() จาก

```
store.dispatch({
  type: 'INCREMENT',
  score: 1
})
```

ก็ได้เป็น

```
store.dispatch(increment(1))
```

กลับมาที่ไฟล์ App.js ทำการเพิ่ม dispatch() ให้กับ onClick() ของ Button กันแบบนี้

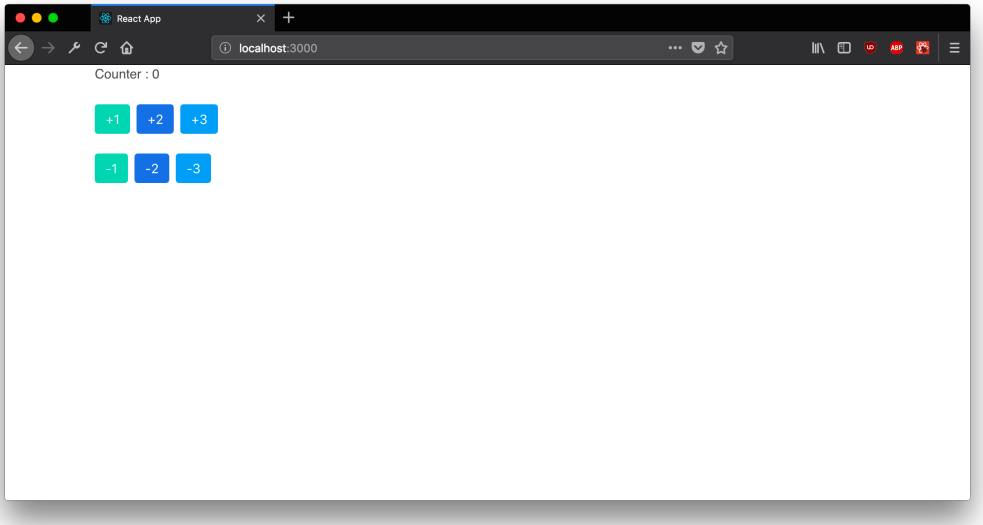
```
const App = ({ message, counter, dispatch }) => (
  <div className="buttons">
    <button onClick={() => dispatch(increment(1))} className="but\
ton is-primary">
      +1
    </button>
    <button onClick={() => dispatch(increment(2))} className="but\
ton is-link">
      +2
    </button>
    <button onClick={() => dispatch(increment(3))} className="but\
ton is-info">
```

```
      +3
    </button>
  </div>

  <div className="buttons">
    <button onClick={() => dispatch(decrement(1))} className="but\
ton is-primary">
      -1
    </button>
    <button onClick={() => dispatch(decrement(2))} className="but\
ton is-link">
      -2
    </button>
    <button onClick={() => dispatch(decrement(3))} className="but\
ton is-info">
      -3
    </button>
  </div>
)
```

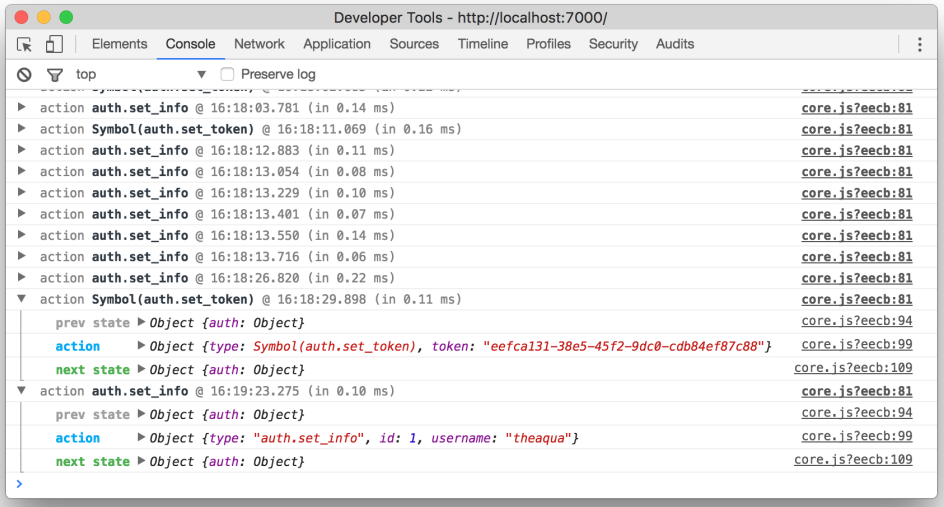
- dispatch : props ที่เราสามารถเข้าถึงได้ มีค่าเท่ากับ store.dispatch() เนื่องจากว่าเราทำการ implement Provider ไว้นั่นเอง
- onClick() จะรับ argument เป็น function นะครับ ซึ่ง ในตัวอย่างเป็น function ที่ return dispatch(action)

หน้าเว็บเราก็สสามารถกดปุ่ม เพิ่ม/ลบ ค่าได้แล้ว รวมถึง แสดงผลลัพธ์ counter ได้ถูกต้อง



Final Result

Step 6 : Redux Logger



Redux Logger

ขั้นตอนสุดท้าย เป็นสิ่งที่เราควรจะเอาไว้ log ดูว่า ตอนนี้ Application เรามี State อะไรบ้าง เราจะได้สามารถ map state ได้ถูกต้อง (กรณีที่บางคนดู reducers หรือ combineReducers แล้วไม่เข้าใจ)

ตัว library มันมีชื่อว่า Redux Logger ก็ install ง่ายๆ เลย

yarn add redux-logger

ต่อมาที่ไฟล์index.js ทำการเพิ่มโค๊ดนี้

index.js

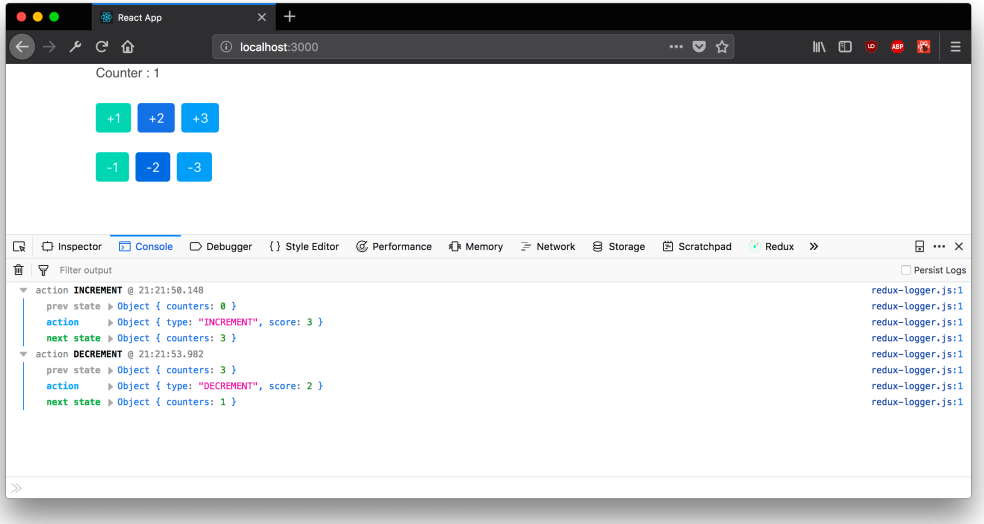
```
import { createStore, applyMiddleware } from 'redux'
import logger from 'redux-logger'

const store = createStore(rootReducer, applyMiddleware(logger))
```

ซึ่งการใช้ redux logger เราจะใช้ applyMiddleware เข้ามาช่วย โดย

- applyMiddleware() รับ logger เป็น argument
- createStore() รับ applyMiddleware เป็น argument ที่ 2 (ก่อนหน้านี้ มีแค่ rootReducer)

แค่นี้ เราก็สามารถดู Redux Log ของเราได้แล้ว ตอนที่มี Action เกิดขึ้น ลองเปิด Developer Tools (Chrome หรือ Firefox ก็ได้) แล้วดูที่ช่อง Console จากนั้นกด +1, +2 จะเห็นมี Action log เกิดขึ้น รวมถึงเราสามารถดูรายละเอียด ว่า dispatch action อะไร? state ถัดไปคืออะไร state ปัจจุบันเป็นยังไง? เรียกได้ว่ามประโยชน์ในการทำ Application แน่นอน ลองไปเล่นกันดูเนอะ



with Redux Logger

สรุป

สำหรับบทความนี้ก็เป็นอีกหนึ่งบทความที่พูดถึง Redux ก็หวังว่าผู้อ่านจะได้ไอเดีย ได้รู้ว่า Redux คืออะไร จะเอาไปใช้ยังไง แก้ปัญหาอะไรได้บ้าง ซึ่งหากใครที่ยังอ่านไม่ค่อยเข้าใจ ก็ยังมีแหล่งเรียนรู้ให้ติดตาม ให้อ่านเพิ่มเติมมากมายครับ ซึ่งผู้เขียนก็ไม่ได้หวังว่าผู้อ่านจะอ่านแค่ที่นี่อย่างเดียวและเข้าใจทั้งหมด มันไม่มีทางเป็นไปได้แน่นอน

ฉะนั้นผมก็เลยแนบ Link บางตัวเผื่อเอาไว้ท่านได้ศึกษาต่อยอดกันครับ (ส่วนใหญ่ก็มีแนะนำไว้ใน Docs ของ Redux อยู่แล้วครับ)

- Redux Official Docs[19]
- Getting Started with Redux[20]
- Building React Applications with Idiomatic Redux[21]
- Redux Example[22]
- The Complete Redux Book (2nd edition)[23]

---

[19] https://redux.js.org/
[20] https://egghead.io/courses/getting-started-with-redux
[21] https://egghead.io/courses/building-react-applications-with-idiomatic-redux
[22] https://redux.js.org/introduction/examples
[23] https://leanpub.com/redux-book

Conclusion

สรุปเนื้อหาทั้งหมดของหนังสือเล่มนี้ก็เอามาจากบล็อคที่ผมทำการ Publish ไว้ หากว่ามีการ Publish เพิ่มเติม ผมก็จะนำมาใส่ใน Ebook เล่มนี้ต่อไป และแน่นอน เล่มนี้ผมก็ยังเปิดให้ Download ฟรีเหมือนเดิม แต่หากใครอยากซื้อหรือช่วย Support ผมยินดีและขอบคุณมากๆครับ

หากอยากช่วย Support หนังสือเล่มนี้ สามารถช่วยเหลือได้ผ่านช่องทางนี้ครับ

SCB ธนาคารไทยพาณิชย์

เลขที่บัญชี 404-780-9150

พรบพิตร สหกิจชัชวาล

Phonbopit Sahakitchatchawan

ขอบคุณครับ