

# React

by SURANART NIAMCOME - May 25, 2015

## Isomorphic คืออะไร ? + สอนวิธีทำด้วย React

**3.8K** Total shares

หากใครตามข่าวในวงการเว็บหน่อยก็คงจะเคยได้ยินคำว่า Isomorphic JavaScript หรือที่สมัยนี้เปลี่ยนมาเรียกว่า Universal JavaScript กันมาบ้างใช้มั้ยล่ะครับ หรือถ้าเพิ่งเคยได้ยิน ก็ขอบอกเลยว่ามันกำลังมาแรงมากๆ ในตอนนี้เช่น วันนี้ SiamHTML เลยจะมาเล่าสู่กันฟังนะ หน่อยว่าเจ้า Isomorphic นี้มันคืออะไร รับรองเลยว่ามันเจ๋งสุดๆ ครับ

### ทุกวันนี้เราทำเว็บกันอย่างไร ?

ก่อนอื่นเรา มาดูกันว่าตั้งแต่อดีตจนถึงปัจจุบัน เราทำเว็บกันอย่างไร ?

#### Server เป็นคน Render

ในยุคแรกๆ เราจะเอา business logic, route และ template ของหน้าต่างๆ ฝากไว้ที่ฝั่ง server ทั้งหมดครับ คือ server จะทำหน้าที่ render หน้าเว็บทั้งหน้าขึ้นมา แล้วถึงจะส่งหน้าเว็บที่สมบูรณ์แล้ว กลับไปยัง client ทั้งนี้ก็ เพราะว่าเครื่อง client สมัยก่อนยังไม่ค่อยแรงเท่าไร ภาระส่วนใหญ่จึงตกอยู่ที่เครื่อง server ครับ

#### Client เป็นคน Render

ในปัจจุบัน เครื่อง client เร็วขึ้นมากครับ เลยเกิดเทคนิคการทำเว็บรูปแบบใหม่ที่เรียกว่า Single-page application หรือ SPA ขึ้นมาโดยการทำเว็บแบบ SPA นี้ business logic, route และ template ของหน้าต่างๆ จะย้ายมาอยู่ฝั่ง client แทนครับ การ render หน้าเว็บก็จะทำที่ฝั่ง client เลย ไม่ต้องเสียเวลามาโหลดหน้าเว็บใหม่ทั้งหน้าเหมือนแต่ก่อน ส่วน server ก็จะเหลือหน้าที่เพียงอย่างเดียวนั่นก็คือการดึงข้อมูลจาก database มาให้ หรือพูดง่ายๆ ก็คือเอา server มาทำ API นั่นเองครับ

#### Server/Client ช่วยกัน Render

สุดท้ายมาดูเทคนิคใหม่ล่าสุดที่รวมเอาข้อดีของ 2 วิธีแรก มารวมกันครับ คือวิธีนี้ตอนโหลดมาที่แรกจะให้ server เป็นคน render หน้าเว็บทั้งหน้ามาให้ก่อน และหลังจากนั้น client จะเป็นคนรับหน้าที่ render ทั้งหมดแทนครับ ซึ่งการจะทำแบบนี้ได้ เราจะต้องอาศัยสิ่งที่เรียกว่า Isomorphic JavaScript เข้ามาช่วย

# Isomorphic JavaScript คืออะไร ?

เวลาเราจะเขียนโค้ดสำหรับดึงค่า cookie มาใช้งาน สมมติว่าที่ผ่าน server เราใช้ PHP และกัน เราจะเขียนโค้ดแบบนึง ส่วนผ่าน client เราใช้ JavaScript เราจะต้องเขียนโค้ดอีกแบบนึงถูกมั่ยครับ เพราะมันคนละภาษา กัน syntax ก็ยอมไม่เหมือนกันอยู่แล้ว ที่นี่ปัญหาคืออะไร ครับ ? เราจะต้องมาทำงาน 2 ครั้ง ถูกมั่ยครับ หรืออย่างน้อยเราก็จะต้องเรียนรู้ 2 ภาษา ที่มีวิธีการเขียนที่ไม่เหมือนกัน ปัญหาเหล่านี้จะหมดไปหากเราใช้ Isomorphic JavaScript ครับ

คำว่า Isomorphic JavaScript นั้นมาจากคำว่า Isomorphic ที่แปลว่า “รูปแบบเดียวกัน” ครับ ซึ่งถ้าจะแปลในบริบทของการทำเว็บแล้ว Isomorphic JavaScript ก็จะหมายถึงรูปแบบการเขียนโค้ด JavaScript ที่เหมือนกันไม่ว่าจะเขียนที่ผ่าน server หรือ ผ่าน client ก็ตามครับ

เอ๊ะ! JavaScript มันสามารถเขียนที่ผ่าน server ได้ด้วยหรอ ? ได้สิครับ ด้วยการใช้ Node.js นั่นเอง หากพูดให้เข้าใจง่ายๆ Isomorphic JavaScript ก็คือการใช้ Node.js ร่วมกับ template engine ที่สามารถ render ได้ทั้งที่ผ่าน server และผ่าน client ครับ วิธีนี้จะช่วยให้เราเขียน business logic, route รวมไปถึง template เพียงแค่ครั้งเดียวเท่านั้น ตอนโหลดครั้งแรก เราจะให้ server ทำหน้าที่ render หน้าเว็บให้ก่อน และหลังจากนั้น พ่อ user กดดูข้อมูลอะไร เราจะจะส่งไปต่อให้ client ทำหน้าที่ render ข้อมูลส่วนนั้นครับ

## ข้อดีของเว็บแบบ Isomorphic

ผมขออภัยด้วยนะครับว่า ผมนี้เป็นเพียงหนึ่งในจุดเด่นของ Isomorphic เลย เพราะข้อดีของมันนี้เรียกว่าตอบโจทย์เว็บสมัยนี้ได้ดีจริงๆ

### ใช้โค้ดร่วมกันทั้ง Server และ Client

อย่างที่บอกครับว่าสิ่งที่เราทำก็แค่เขียน JavaScript ที่เราคุ้นเคยเป็นอย่างดี ไม่ต้องไปแตะภาษาอื่นๆ ที่เราไม่ค่อยคุ้นเคยเลย (ขอบอกว่า Front-end Developer ทุกคนสามารถทำได้แน่นอนครับ)

### โหลดครั้งแรกไม่กระพริบ

ใครทำ SPA บ่อยๆ จะรู้ดีครับว่า ตอนโหลดหน้าเว็บครั้งแรกมันจะขอบกระพริบๆ เพราะมันไม่ได้ render มาตั้งแต่ผ่าน server ซึ่งอาการนี้จะไม่พบในเว็บแบบ Isomorphic แน่นอนครับ

### SEO

ประเด็นนี้สำคัญสุดๆ ครับ เพราะว่าเว็บแบบ SPA นั้น ไม่ค่อยดีต่อ SEO เท่าไร (จริงๆ ก็ทำให้ได้ แต่ค่อนข้างจะลำบาก) แต่ Isomorphic นั้นได้ SEO ไปเต็มๆ ครับ เพราะ server จะ render หน้าที่มีข้อมูลสมบูรณ์มาให้ตั้งแต่ตอนที่โหลดมาครั้งแรกแล้ว

มาถึงตรงนี้คาดว่าเพื่อนๆ คงอยากรู้แล้วใช่มั้ยครับว่าขั้นตอนการทำเว็บแบบ Isomorphic นั้น มันยากง่ายแค่ไหน ส่วนตัวมองว่ามันไม่ได้ยากมากนะครับ เพียงแต่ว่าการจะเขียน Isomorphic ได้นั้น เราจำเป็นจะต้องมีความรู้พื้นฐานเยอะซักนิดนึง

## พื้นฐานที่ควรรู้

น่าจะกันครับว่าก่อนที่จะเริ่มลงมือทำเว็บแบบ Isomorphic เราควรจะมีพื้นฐานอะไรมาก่อนบ้าง

### Node.js

อันนี้ขาดไม่ได้เลยครับ เพราะเราจะต้องอาศัยมันในการใช้ JavaScript เป็นภาษาของผ่าน server

### Express + Jade / Handlebars

ต่อมาเป็น framework ที่จะใช้กับ Node.js ครับ ซึ่งจริงๆ และจะใช้หรือไม่ใช้ก็ได้ สำหรับบทความนี้ ผมจะขอใช้ Express และกันส่วน template engine ที่จะใช้กับ Express นั้น จะใช้หรือไม่ใช้ก็แล้วแต่เลยครับ

### Browserify / Webpack

สิ่งสำคัญที่ขาดไม่ได้เลยก็คือ tool ที่จะใช้แปลง JavaScript Module แบบ **CommonJS** ให้สามารถรันบน web browser ได้ครับ สำหรับบทความนี้ผมจะขอใช้เป็น Webpack และกันนะจะ

## React

มาถึงพระเอกของงานนี้ครับ นั่นก็คือ template engine ที่สามารถ render ได้ทั้ง server และฝั่ง client นั่นเอง ซึ่งจริงๆ แล้วในปัจจุบันมีให้เลือกใช้หลายท่าเลย แต่ในบทความนี้ ผมจะขอเลือกใช้เป็น React และกันนะจะ (จริงๆ และ React นั้นไม่ใช่ template engine นะครับ เพียงแต่เราจะอาศัยความสามารถส่วนหนึ่งของมันมาเป็น template engine เท่านั้นเอง)

## ECMAScript 2015

อันนี้ถึงแม้ว่าจะไม่จำเป็น แต่ผมแนะนำให้เรียนรู้ไว้ครับ เพราะโคดของ project ต่างๆ ที่เกี่ยวกับ Isomorphic ส่วนใหญ่จะเขียนด้วย ECMAScript 2015 หมดเลยนะ

ถ้าใครเป็นทั้งหมดนี่มาก่อนแล้ว ผมว่าสบายแล้วล่ะครับ แต่ถ้าใครยังเป็นไม่ครบถ้วนไปอ่านบทความที่ผมทำลิ้งค์ไว้ แล้วค่อยๆ ทำความเข้าใจไปก็ได้จะ อ่ายใจร้อน เพราะขีนข้ามขันไปเล่น Isomorphic เลยนี่ ผมว่าคงจะงงจนอาจจะถึงขั้นท้อก่อนที่จะทำเป็นเลยก็ได้ครับ

## ตกลงกันก่อน !

มาถึงตรงนี้ หากใครยังไม่เคลียร์ในประเด็นไหน ผมขอแนะนำให้ย้อนกลับไปอ่านอีกทีก่อนนะครับ เพราะหลังจากนี้ เราจะเริ่มลงมือเขียนโคดกันจริงๆ แล้ว ซึ่งตอนนั้นผมจะถือว่าพื้นฐานของเราแน่นแล้ว รายละเอียดเล็กๆ น้อยๆ ผมอาจจะไม่ได้อธิบาย เพราะไม่อยากให้บทความนี้ยาวเกินไปครับ (นี่ขนาดไม่อยากแล้วนะ -\_-“)

เอาล่ะครับ เราจะมาเริ่มลงมือเขียนเว็บแบบ Isomorphic กัน ก่อนอื่นผมจะขอพูดถึงพระเอกของงานนี้อย่าง React ซักนิดนึง เพราะมันถือเป็นหัวใจสำคัญของ Isomorphic เลยทีเดียว

## React คืออะไร ?

ในเว็บของ React เขียนเอาไว้ว่า React เป็น **JavaScript Library** ที่เอาไว้สร้าง **User Interface** ครับ พึ่งดูแล้วเหมือน scope มันจะเล็กๆ เนอะ ว่ามันคือเอาไว้ทำแค่พวก component บนหน้าเว็บอะไรทำนองนี้อะหรอ ? คำตอบคือใช่เลยครับ บางคนอาจจะนิยาม Web Components หรือ Custom Directive ของ AngularJS แต่ขอบอกเลยว่า React มันเทพกว่านั้นเยอะ

## จุดเด่นของ React

มาตรฐานครับว่าสิ่งที่ทำให้ React ดูโดดเด่นกว่าช้าบ้านนั้นมีอะไรบ้าง

### สร้างโดย Facebook

ก่อนอื่นต้องขอบอกก่อนนะครับว่าคนทำ React นั้นไม่ใช่ใครที่ไหน Facebook นั่นเองครับ และทุกวันนี้ React ก็ยังคงถูกใช้อยู่ทั่วใน Facebook และ Instagram ก็ลองคิดกันเอาเองนะครับว่ามันจะสุดยอดขนาดไหน

### Render ได้ทั้ง 2 ฝั่ง

ข้อนี้เอง ที่ทำให้ React ดูดีกว่า AngularJS ในแง่ของ SEO ครับ เนื่องจาก AngularJS นั้นจะทำกับ DOM ได้เพียงอย่างเดียว crawler จึงไม่สามารถ index เนื้อหาที่โหลดเข้ามาทีหลังได้ครับ

### Virtual DOM

ในส่วนของการ render ที่ฝั่ง client นั้น React ก็ยังดูดีกว่า AngularJS ตรงที่ React จะไม่ทำอะไรกับ DOM โดยตรง แต่ React จะสร้างสิ่งที่เรียกว่า Virtual DOM ขึ้นมา และดูว่า DOM ที่มีอยู่เดิมกับ DOM ที่กำลังจะเปลี่ยนไปนั้นแตกต่างกันอย่างไร เมื่อ

หากความแตกต่างได้แล้ว React จะค่อยไปแก้ DOM จริงๆ ซึ่งขั้นตอนที่ว่านี้อยู่ใน memory ทั้งหมดครับ เรียกได้ว่าการใช้ React ทำ DOM Manipulation นั้น เร็วกว่าการใช้ JavaScript แบบเดียวๆ ชะอึก

## Unidirectional

ในปัจจุบัน เราจะพบว่า JavaScript framework ส่วนใหญ่จะเป็นแบบ Two-way Data Binding ซึ่งมันก็คือการที่ data ใน model และ view จะเปลี่ยนแปลงตามกันและกัน แต่ React นั้นตรงกันข้ามครับ คือจะเป็นแบบ Unidirectional หรือแบบทางเดียว นั่นเอง เพราะ React มองว่า Two-way นั้นอาจทำให้เกิดการอพเดทข้อมูลที่ไม่จำเป็นขึ้นมาได้ครับ

มาถึงตรงนี้ ผู้อ่านอาจจะมีคิดมองว่า “ผู้ใช้ React แล้ว หรือมองว่าผมไม่ชอบ AngularJS หรือเปล่า ? ตอบเลยว่าไม่ใช่ครับ ทั้ง React และ AngularJS ต่างก็เป็น tool ที่ดีด้วยกันทั้งคู่ และไม่ได้เกิดมาเพื่อทดแทนกันครับ หากสนใจสักว่า React นั้นมีข้อเสียอะไรบ้าง ? ขอให้ออดใจรอหนึ่งนาที เพราะเดียวเราจะได้เจอกับตัวตนที่เขียนโค้ดในบทความนี้แหล่ะครับ

## Syntax ของ React

เล่ามาตั้งนาน มาดูวิธีการเขียนโค้ดของ React กันดีกว่าครับ ก่อนอื่นเราจะต้องเข้าใจก่อนว่า แนวคิดของ React นั้นก็คือ การมองทุกสิ่งทุกอย่างบนหน้าเว็บเป็น component หากมองดีๆ จะเห็นว่าหน้าเว็บนั้นก็จะมี root element อันนึง ภายในก็จะประกอบไปด้วยกล่องต่างๆ และภายในกล่องต่างๆ นั้นก็อาจจะมีกล่องลูกๆ ซ้อนเข้าไปข้างในอีกที เจ้าพวกกล่องเหล่านี้แหล่ะ ที่เราจะเรียกมันว่า component

## Top-Level API

ก่อนจะเริ่มสร้าง component กัน เรามาดู api ที่เราจะต้องใช้กันแน่ๆ ครับ ว่ามันมีอะไรบ้าง

### React.createClass

เอาไว้สร้าง component ครับ เพราะจริงๆ แล้ว component มันก็คือ class ที่ประกอบไปด้วย method ต่างๆ นั่นเอง

### React.createElement

เอาไว้สร้าง ReactElement ขึ้นมาจากการ component ที่เราสร้างครับ

### ReactDOM.render

เอาไว้ render ReactElement เข้าไปใน DOM ครับ เราจะใช้ method นี้ ในการ render ทางฝั่ง client

### ReactDOMServer.renderToString

เอาไว้ render ReactElement ออกมายัง string ครับ เราจะใช้ method นี้ ในการ render ทางฝั่ง server

api พวกนี้ เรา秧งไม่ต้องไปท่องจำนะครับว่าอันไหนเอาไว้ใช้ทำอะไรบ้าง ผนอย่างให้เราเห็นภาพรวมคร่าวๆ ว่าเราจะต้องสร้าง component ของ react ขึ้นมาก่อน และค่อยบอกให้ react มัน render component เหล่านั้น เข้าไปใน DOM ก็เท่านั้นเองจะ

## วิธีสร้าง Component แบบง่ายๆ

สมมติว่าเรามี container เป็นๆ อันนึง ชื่อ `div#content` หากเราต้องการจะใส่ component ง่ายๆ อย่าง `p.greet` เข้าไปแบบนี้ เราจะต้องเขียนโค้ดอย่างไร ?

```
<body>
  <div id="content">
    <p class="greet">Hello, SiamHTML!</p>
  </div>
</body>
```

จากโจทย์ด้านบน เราสามารถเขียนโค้ดด้วย React ได้แบบนี้ครับ

```
// โหลด React มาสร้าง component
var React = require('react');

// โหลด ReactDOM มา render component ใส่ DOM
var ReactDOM = require('react-dom');

// สร้าง component ชื่อ Greet
var Greet = React.createClass({
  // component นี้ แสดงผลอย่างไร ?
  render: function() {
    return (
      // เป็น p.greet ที่มี text เป็น "Hello, SiamHTML!"
      React.createElement(
        'p',
        {className: "greet"}, "Hello, SiamHTML!"
      )
    );
  }
});

// แสดงผล component ชื่อ Greet ข้างใน #content
ReactDOM.render(
  React.createElement(Greet, null),
  document.getElementById('content')
);
```

จะเห็นว่าเราใช้ `React.createClass` ในการสร้าง component ขึ้นมาครับ ซึ่งภายในก็จะเป็น object ที่จะเอาไว้ใช้ในการอธิบายว่า component นี้มีลักษณะอย่างไร สังเกตนะครับว่าผมจำเป็นต้องใส่ method `render` เอาไว้ด้วย ไม่งั้น React มันจะไม่รู้ว่าจะแสดงผล component นี้ออกมาอย่างไร

แม่เจ้า! โจทย์ง่ายๆ แค่นี้ React ล่อไปกีบรหัสเนี่ย ใช้ jQuery เขียนบรรทัดเดียวจบเลยนะ! อาย่าเพิ่งบ่นไปครับ เพราะคนทำ React เด็กๆ เตรียมวิธีแก้ปัญหานี้มาแล้ว ด้วยสิ่งที่เรียกว่า JSX

## รู้จักกับ JSX Syntax

จากโจทย์เดิมครับ เราจะเปลี่ยนมาสร้าง component โดยใช้ syntax แบบใหม่ที่เรียกว่า JSX แทน ลองดูโค้ดด้านล่างนี้จะดีกว่า

```
var Greet = React.createClass({
  render: function() {
    return (
      // เขียนแบบ JSX
      <p className="greet">Hello, SiamHTML!</p>
    );
  }
});

// เขียนแบบ JSX
ReactDOM.render(<Greet />, document.getElementById('content'));
```

เห็นมั้ยครับว่าโค้ดของเราจะง่ายขึ้นเยอะเลย จะเห็นว่าการเขียนด้วย JavaScript ด้วย JSX นั้น คล้ายกับการเขียน XML มากเลยครับ จะต่างเล็กน้อยก็ตรงที่การระบุ `class` นั้น เราจะต้องเปลี่ยนมาใช้ `className` แทนเท่านั้นเอง

ที่นี่ลองมาดูตัวอย่าง component ที่ขับช้อนขึ้นอีกนิดนึงครับ สมนติเราจะสร้าง component ใหม่ที่ชื่อ `<Header />` และข้างในมี component ลูก อよู่ 2 ตัว คือ `<Menu />` และ `<Search />` เราจะจะเขียนโค้ดแบบนี้ครับ

```
var Menu = React.createClass({
  render: function() {
    return (
      <ul>
        <li>Menu 1</li>
        <li>Menu 2</li>
        <li>Menu 3</li>
      </ul>
    );
  }
});

var Search = React.createClass({
  render: function() {
    return (
      <form>
        <input type="text" value="Search" />
        <input type="submit" value="Submit" />
      </form>
    );
  }
});

var Header = React.createClass({
  render: function() {
    return (
      <header>
        <Menu />
        <Search />
      </header>
    );
  }
});

ReactDOM.render(<Header />, document.getElementById('content'));
```

JSX ช่วยให้อะไรๆ ง่ายขึ้นเยอะเลยใช่มั้ยล่ะครับ มาถึงตรงนี้ เราคงพอจะเห็นภาพแล้วว่า การใช้ React สร้าง UI ของหน้าเว็บ ก็คือการแบ่งของในหน้าเว็บออกเป็น component ย่อยๆ ก่อน และค่อยจัดกลุ่ม component ย่อยๆ เหล่านั้นให้กลายเป็น component ที่ใหญ่ขึ้นเมื่อ component ใหญ่ขึ้นเรื่อยๆ สุดท้ายแล้ว มันก็จะกลายเป็นหน้าเว็บที่สมบูรณ์ครับ

## ส่งต่อข้อมูลไปยัง Component วิ่งด้วย Props

ในการทำเว็บจริงๆ คงไม่มีใครใส่ข้อมูลลงไปใน template ตั้งแต่แรกกูมั่ยครับ เราแม้กจะใช้วิธี assign ข้อมูลเหล่านั้น เข้าไปใน template แทน การใช้ React ก็เหมือนกันครับ สมมติเราต้องการจะส่งค่าอะไรให้กับ component เราสามารถเขียนแบบนี้ได้เลยนะ

```
| ReactDOM.render(<Greet name="SiamHTML" />, document.getElementById('content'))
```

แบบนี้จะเป็นการส่ง `props` ชื่อ `name` ที่มีค่าเป็น “SiamHTML” ไปให้กับ component ที่ชื่อ `Greet` ครับ ที่นี่ หากเราอยากรับข้อมูลที่ได้รับ มาแสดง ให้เราใช้ `this.props` และตามด้วยชื่อ `props` แบบนี้ได้เลยครับ

```
var Greet = React.createClass({
  render: function() {
    return (
      // ดึงค่าของ property ชื่อ name มาใช้
      <p className="greet">Hello, {this.props.name}!</p>
    );
  }
});
```

ผลลัพธ์ที่ได้ก็จะออกมาเป็น “Hello. SiamHTML!” ครับ

## แชร์ข้อมูลภายใน Component ด้วย State

แต่บางที่ แค่ property อย่างเดียวอาจจะทำอะไรได้ไม่มากนักครับ เพราะพอเรา assign ค่าอะไรให้ component แล้ว พอมัน render เสร็จ มันก็จะอยู่นิ่งๆ แบบนั้นไปตลอด หากเราอยากระบุให้ component มี interaction ใดๆ เราจะต้องอาศัยสิ่งที่เรียกว่า **state** เข้ามาช่วย ครับ

สำหรับผมแล้ว **state** นี้ถือเป็นจุดขายของ React เลยนะครับ หน้าที่ของมันก็คือการเก็บ “สภาพ” ของ component ในขณะนั้นๆ เอาไว้ เมื่อไรก็ตามที่ **state** มีค่าเปลี่ยนไป เมื่อนั้น method **render** จะถูกสั่งให้ทำงานทันทีครับ พอดีๆ ก็คือ หากเราต้องการจะอัพเดท หน้าตาของ component ใหม่ ก็ให้เราไปเปลี่ยน **state** ของมันนั้นแหล่งครับ เพื่อให้เห็นภาพมากขึ้น ลองมาดูโค้ดสร้าง component ที่ เป็นปุ่ม Like กันครับ

```
var LikeButton = React.createClass({
  // method นี้ เอาไว้กำหนด state เริ่มต้น
  getInitialState: function() {
    return {liked: false};
  },
  // เพิ่ม custom method ชื่อ handleClick
  handleClick: function(event) {
    // เปลี่ยน state "liked" ให้มีค่าตรงข้ามกับค่า liked เดิม
    this.setState({liked: !this.state.liked});
  },
  render: function() {
    var text = this.state.liked ? 'Unlike' : 'Like';
    return (
      // เมื่อถูก click ให้รัน method handleClick
      <button onClick={this.handleClick}>
        {text}
      </button>
    );
  }
});
```

จะเห็นว่าที่ **handleClick** นั้น เราไม่ต้องไปสั่งให้มัน **render** ใหม่นะครับ เพราะเมื่อไรที่ **state** เปลี่ยน component ก็จะถูก **render** ใหม่โดยอัตโนมัติอยู่แล้วครับ

## Lifecycle Methods ของ Component

สุดท้ายเรามาดู Lifecycle ของ component กันหน่อยครับว่า ตั้งแต่ component ถูกสร้างขึ้นมา จนถึงตอนที่มันถูกถอนออกหน้าเว็บ นั้น มันเกิดอะไรขึ้นบ้าง

### **componentWillMount**

เกิดขึ้นก่อนที่ component จะถูก **render** ขึ้นมา

### **componentDidMount**

เกิดขึ้นหลังจากที่ component ได้ถูก **render** แล้ว

### **componentWillReceiveProps**

เกิดขึ้นหลังจากที่ component ได้รับ **props** ค่าใหม่

### **componentWillUpdate**

เกิดขึ้นก่อนที่ component จะถูก **render** ใหม่ หลังจากที่ได้รับ **props** ใหม่

### **componentDidUpdate**

เกิดขึ้นหลังที่จาก component ถูก **render** ใหม่

## componentWillUnmount

เกิดขึ้นก่อนที่ component จะถูกเอาออกจาก DOM

สาเหตุที่เราต้องรู้ Lifecycle ของ component ก็เพราะว่าเราสามารถนำ Lifecycle เหล่านี้มาใส่เป็น method ให้กับ component ได้ครับ ลองดูตัวอย่างโค้ดด้านล่างนี้

```
var Timer = React.createClass({
  // กำหนด state ชื่อ secondsElapsed ให้มีค่าเป็น 0
  getInitialState: function() {
    return {secondsElapsed: 0};
  },
  // custom method สำหรับอัพเดท state ชื่อ secondsElapsed
  tick: function() {
    this.setState({secondsElapsed: this.state.secondsElapsed + 1});
  },
  // หลังจาก render แล้ว ให้ทำอะไร ?
  componentDidMount: function() {
    this.interval = setInterval(this.tick, 1000);
  },
  // ก่อนจะลบออกจาก DOM ให้ทำอะไร ?
  componentWillUnmount: function() {
    clearInterval(this.interval);
  },
  // แสดงผลลอกมาอย่างไร ?
  render: function() {
    return (
      <div>Seconds Elapsed: {this.state.secondsElapsed}</div>
    );
  }
});
```

ก็น่าจะพอเห็นประโยชน์ของ Lifecycle methods กันบ้างแล้วใช่มั้ยครับ ที่นี่เราลองมาดูวิธีการนำ React ไปใช้งานจริงกันดีกว่าครับ

## วิธีนำ React มาใช้งาน

สำหรับการใช้งาน React นั้นจะแบ่งออกเป็น 2 วิธีด้วยกัน ดังนี้ครับ

### 1. ใช้ที่ฟัง Client

แบบแรกจะเป็นการเอา React มาใช้สร้าง UI ที่ฟัง client ครับ วิธีใช้ก็ง่ายๆ ให้เราไป [ดาวน์โหลด React](#) มา ก่อน หรือจะใช้จากเว็บของ facebook ไปก่อนก็ได้จะ จากนั้นก็ติด script ของ React เข้าไปแบบนี้ได้เลย

```
<!DOCTYPE html>
<html>
  <head>
    <script src="https://fb.me/react-0.14.3.min.js"></script>
    <script src="https://fb.me/react-dom-0.14.3.min.js"></script>
  </head>
  <body>
    <div id="content"></div>
  </body>
</html>
```

สังเกตนะครับว่าเราจะต้องติด script ของ React ถึง 2 ตัว ด้วยกัน คือตัวแรกจะเป็น core ของ React เลย ส่วนตัวที่สองจะเป็น script ที่ดูแลในส่วนของการ render ใส่ DOM ของหน้าเว็บครับ และถ้าเราเขียน React โดยใช้ JSX ด้วย เราอาจจะต้องเพิ่ม script ที่ชื่อ Babel ที่มีความสามารถในการแปลง JSX เข้ามาอีกด้วยครับ ไม่งั้นโค้ดที่เราเขียนจะพัง เพราะ web browser มันยังไม่รู้จัก JSX นั่นเอง

```
<head>
  <script src="https://fb.me/react-0.14.3.min.js"></script>
  <script src="https://fb.me/react-dom-0.14.3.min.js"></script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/babel-core/5.8.23/browser.min.js"></script>
</head>
```

ส่วนโค้ดหลักของเรา ก็แยกไปเขียนอีกไฟล์ก็ได้ครับ และค่อยใส่เข้ามาต่อท้าย container หลักของเราแบบนี้

```
<body>
  <div id="content"></div>
  <script type="text/babel" src="siamhtml.js"></script>
</body>
```

สังเกตนะครับว่า เราจะต้องใส่ type ของ script ให้เป็น **text/babel** ด้วย เพราะ Babel จะแปลงให้เฉพาะ script ที่เป็น type นี้เท่านั้น ครับ เพียงเท่านี้ เราก็พร้อมที่จะรันโค้ดที่เขียนด้วย React และล่ะครับ

## 2. ใช้กีฬ์ Server

ส่วนแบบที่สองจะเป็นการเอา React มาใช้สร้าง UI ด้วยภาษาฝั่ง server อย่าง Node.js ครับ วิธีนี้จะค่อนข้างซับซ้อนกว่าวิธีแรกนิดนึง เพราะเราจะต้องเปลี่ยนมาใช้ React แบบที่เป็นโมดูล CommonJS แทน

```
// ไฟล์ SiamHTML.jsx

// โหลด React ที่เป็นโมดูลแบบ CommonJS มาใช้งาน
var React = require('react');

// ใช้ React สร้าง component
var SiamHTML = React.createClass({
  render: function() {
    return (
      <p>SiamHTML</p>
    );
  }
});

// export component นี้ ออกเป็น module
module.exports = SiamHTML;
```

ส่วนฝั่ง server ก็ให้ใช้ **ReactDOMServer** ในการ render component ออกมาเป็น string ครับ

```
// ไฟล์ app.js

// โค้ดสำหรับสร้าง HTTP server
...

// โหลด React มาใช้งาน
var React = require('react');

// โหลด ReactDOMServer มา render component ให้เป็น String
var ReactDOMServer = require('react-dom/server');

// โหลด component ที่ชื่อ SiamHTML มาใช้
var SiamHTML = require('./components/SiamHTML.jsx');

// โค้ดสำหรับ render component ที่ใช้กับฝั่ง server
app.get('*', function(req, res) {
  res.send(ReactDOMServer.renderToString(React.createElement(SiamHTML)));
});
```

มาถึงตรงนี้ บางคนอาจจะสงสัยว่า ถ้าจะทำ Isomorphic นั้นก็ต้อง render ได้ทั้ง 2 ฝั่งเลยสิ แล้วเราจะใช้วิธีไหนล่ะ ?

## Workshop – Setup Isomorphic

เอาล่ะครับ เมื่อเรารู้พื้นฐานต่างๆ ครบแล้ว เรา มาเริ่มเขียน Isomorphic กันเลยดีกว่า

### เตรียมไฟล์และโฟลเดอร์

เริ่มด้วยการจัดโครงสร้างของเว็บแอปเปิลครับ ให้เราสร้างไฟล์และโฟลเดอร์ขึ้นมาตามนี้ได้เลย

```
myApp/
|-
|-- public/
|   |
|   |-- images/          # เก็บรูปต่างๆ ที่จะใช้ในเว็บ
|   |
|   `-- js/
|       |
|       `-- bundle.js    # ไฟล์ bundle ที่ได้จากการรัน webpack
|
|-- src/
|   |
|   |-- js/
|   |   |
|   |   |-- components/  # เก็บ component ทั้งหมด (.jsx)
|   |   |
|   |   |-- client.js    # โค้ดสำหรับ render ฝั่ง client (เป็น entry point ของ webpack)
|   |   |
|   |   |-- router.jsx   # middleware สำหรับทำ routing
|   |   |
|   |   |-- routes.jsx   # เก็บ routes ทั้งหมดของแอป
|   |   |
|   |   `-- server.js    # โค้ดสำหรับ render ฝั่ง server
|   |
|   `-- templates/        # เก็บ template ต่างๆ ของแอป
|
|-- .babelrc              # ไฟล์กำหนด option ของ Babel
|
|-- app.js                 # ไฟล์ entry point ของฝั่ง server (รันด้วย Node.js)
|
|-- package.json            # ระบุ dependency ที่จะใช้
|
`-- webpack.config.js      # ไฟล์ config ของ webpack
```

ก่อนไปต่อ ผมอยากรู้ว่า ทำความเข้าใจกับโครงสร้างนี้ให้ดีๆ นะครับ เพราะถ้าเราเข้าใจมันแล้ว ขั้นตอนต่อๆ ไปก็จะง่ายขึ้นเยอะเลยล่ะครับ

### ติดตั้ง Babel

เนื่องจากสมัยนี้ เราไม่ได้เขียน JavaScript กันแบบเพี้ยวๆ และ เราเลยจะต้องติดตั้ง Babel มาช่วยแปลงโค้ดของเราให้กลับมาเป็นโค้ด JavaScript แบบธรรมดากัน

```
| npm install babel babel-register --save-dev
```

ก่อนหน้านี้พอติดตั้ง Babel เสร็จแล้ว มันจะพร้อมใช้งานได้ทันทีเลย แต่ตอนนี้ไม่ได้แล้วครับ เพราะเราจะต้องบอก Babel ก่อนว่าอยากรู้ว่า ให้ Babel ช่วยแปลงอะไรบ้าง สำหรับโปรเจกต์นี้ เราจะมีการใช้ JSX และ ECMAScript 2015 ครับ ให้เราติดตั้ง preset ของ Babel ตามนี้ได้เลย

```
| npm install babel-preset-es2015 babel-preset-react --save-dev
```

จากนั้นให้เราไปสร้างไฟล์ `.babelrc` ที่จะเอาไว้กำหนด option ต่างๆ ให้กับ Babel และใส่ preset ที่เราโหลดมาเมื่อกี้ลงไป

```
{
  "presets": ["es2015", "react"]
}
```

เพียงเท่านี้ Babel ก็พร้อมใช้งานแล้วล่ะครับ ^0^

## สร้าง Entry point ของผู้จัดการ Server

เริ่มแรกให้เราเข้าไปดูที่ไฟล์ `app.js` ซึ่งจะเป็นไฟล์ที่เราเอาไว้ start app Node.js ของเราจะ ให้เราใส่โค้ดด้านล่างนี้ลงไป

```
// ให้ Babel ช่วยแปลง ECMAScript 2015 และ JSX
require('babel-register');

// โหลดไฟล์ที่เอาไว้ใช้ start server
require('./src/js/server.js');
```

จะเห็นว่าไฟล์นี้ เราแค่จะเอาไว้เรียก Babel มาใช้งานเท่านั้นเองจะ ที่นี่เราจะสามารถเขียนโค้ดด้วย JSX และ ECMAScript 2015 ได้แล้วล่ะครับ

## ใส่โค้ดสำหรับ Render ผู้จัดการ Server

ต่อมาไปดูที่ไฟล์ `server.js` ครับ ไฟล์นี้เราจะเอาไว้สร้าง http server ขึ้นมาเพื่อรับ request จาก client ให้เราใส่โค้ดสำหรับสร้าง http server ขึ้นมาตามปกติ โดยบทความนี้ ผมจะขอใช้ Express เป็น framework สำหรับ Node.js และใช้ Handlebars เป็น template engine ครับ ให้เราติดตั้ง dependency ที่จำเป็นตามด้านล่างนี้

```
| npm install express express-handlebars serve-favicon --save-dev
```

จากนั้นก็ใส่โค้ดสำหรับ create server ด้วย Express ลงไป

```
import path from 'path';
import express from 'express';
import exphbs from 'express-handlebars';
import favicon from 'serve-favicon';

var app = express();
app.use(favicon(path.join(__dirname, '../public/images/favicon.ico')));
app.use(express.static('public'));

app.engine('handlebars', exphbs());
app.set('view engine', 'handlebars');
app.set('views', path.join(__dirname, '../templates'));

var server = app.listen(3000, function() {
  var host = server.address().address;
  var port = server.address().port;
  console.log('listening at http://%s:%s', host, port);
});

app.get('*', function(req, res){
```

```

    res.render("index.handlebars", {
      markup: 'Hello, SiamHTML!'
    });
  });
}

```

จะเห็นว่าเป็นโค้ดธรรมดา ที่เราอาจจะคุ้นเคยกันดีครับ ที่นี่มาต่อ กันที่ไฟล์ `index.handlebars` ที่เราจะใช้เป็น template หลักกันบ้างครับ ให้เราใส่โค้ดด้านล่างนี้ลงไป

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Isomorphic with React Tutorial by SiamHTML</title>
</head>
<body>
  <div id="content">{{markup}}</div>
</body>
</html>

```

เสร็จแล้วก็ลองรันดูเลยครับ เราจะได้หน้าเว็บที่มีคำว่า “Hello, SiamHTML!” แสดงออกมา

ที่นี่เราจะเปลี่ยนการ assign ค่า markup จากการยัด string ลงไปตรงๆ มาเป็นการ render ด้วย component ของ React กันบ้างครับ ก่อนอื่นให้เราสร้าง component ที่ง่ายสุดๆ ขึ้นมาสักอัน แล้วตั้งชื่อไฟล์ว่า `App.jsx` ครับ

```

var React = require('react');

var App = React.createClass({
  render: function () {
    return (
      <div>
        Hello, SiamHTML!
      </div>
    );
  }
});

module.exports = App;

```

แต่อย่างที่บอกครับ สมัยนี้เค้านิยมเขียน ECMSScript 2015 กัน งั้นเราก็ตามเทรนด์ไปหน่อยละกันนะ เรายังจะได้โค้ดสำหรับสร้าง `<App />` ที่มีหน้าตาแบบนี้

```

import React from 'react';

class App extends React.Component {

  render() {
    return (
      <div>
        Hello, SiamHTML!
      </div>
    );
  }
}

export default App;

```

จะเห็นว่ามันแก้แค่นิดเดียวเองนะ เอาเป็นว่าต่อจากนี้ไป พอจะเขียน ECMAScript 2015 ตลอดเลยละกันนะครับ ใครยังไม่คุ้นกับ syntax ก็ลองไปอ่านทบทวนกันดูได้อะ

พอเราจะเปลี่ยนมาใช้ React เรา ก็จะต้องแก้โค้ดในส่วนของการ render ที่ไฟล์ **server.js** นิดนึงครับ แต่อย่าลืมนะครับว่าการ render ที่ฟัง server นั้น เราจะต้องอาศัย **ReactDOMServer** ด้วย ให้เราติดตั้ง dependency เพิ่ม ตามด้านล่างนี้จะ

```
| npm install react react-dom --save-dev
```

จากนั้นก็เข้าไปแก้ไฟล์ **server.js** ตรงส่วนของ routing นิดนึงครับ

```
// โค้ดสำหรับสร้าง HTTP server
...
import React from 'react';
import ReactDOMServer from 'react-dom/server';
import App from './components/App.jsx';

app.get('*', function(req, res){
  res.render("index.handlebars", {
    markup: ReactDOMServer.renderToString(React.createElement(App))
  });
});
```

เมื่อลองรันดูอีกที ก็จะพบว่าผลลัพธ์ยังคงเหมือนเดิมเลยครับ เพียงแต่ตอนนี้เราเปลี่ยนมา render ด้วย React แล้ว

## การทำ Routing

เมื่อ render ด้วย server ได้แล้ว ต่อมา ก็จะเป็นการทำ routing ครับ ซึ่งในการทำ Isomorphic นั้น เราจะไม่ใช้ routing ของ Express นะ ครับ เพราะว่ามันไม่สามารถแชร์โค้ดกับฝั่ง client ได้ วิธีแก้ก็คือการใช้โนดูลที่มีชื่อว่า **react-router** ครับ

```
| npm install react-router --save-dev
```

จากนั้นให้เราสร้างไฟล์ **src/js/routes.jsx** และใส่โค้ดด้านล่างนี้ลงไว้

```
import React from 'react';

// โหลดความสามารถของ react-router มาใช้งาน
import { Route, IndexRoute } from 'react-router';

// โหลด component ต่างๆ
import App from './components/App.jsx';
import Home from './components/Home.jsx';
import About from './components/About.jsx';

// ระบุว่า path นี้จะถูก handle ด้วย component ไหน
export default (
  <Route path="/" component={App}>
    <IndexRoute component={Home}/>
    <Route path="/about" component={About}/>
  </Route>
);
```

หากไม่ได้โค้ดดู ก็คงจะเข้าใจได้ไม่ยากครับ ข้อสังเกตก็คือ โค้ด JSX ในส่วนของการกำหนด route นั้นสามารถช้อนต่อกันไปได้เรื่อยๆ เลย นะครับ หาก **path** ของ **<Route />** ไหน match สำเร็จ **component** ของ **<Route />** เหล่านั้น ก็จะถูกนำมา **render** ตามตามระดับที่เรา ช้อนกันไว้ครับ จากโค้ดด้านบน หาก **path** เป็น **/about** สิ่งที่ render ออกมาก็จะเป็น **<App />** ที่มี **<About />** อยู่ข้างในครับ

ต่อมาให้เราเตรียม component ของแต่ละ route มาให้พร้อมแล้วครับ ไฟล์นั้นอยู่ที่ไฟล์ **App.jsx** ครับ เพราะมันเป็น component ระดับสูงสุดของเรา ให้เราลบโค้ดเดิมใน **App.jsx** ออกให้หมด และเปลี่ยนมาใช้โค้ดด้านล่างนี้แทนครับ

```
import React from 'react';

// โหลดความสามารถของ react-router มาใช้งาน
import { Link } from 'react-router';

class App extends React.Component {

    // ใส่ link ไปยังหน้า Home และ About
    render() {
        return (
            <div>
                <header>
                    <ul>
                        <li>
                            <a href="/">Home</a>
                        </li>
                        <li>
                            <a href="/about">About</a>
                        </li>
                    </ul>
                </header>
                {this.props.children}
            </div>
        );
    }
}

export default App;
```

สังเกตนะครับว่าที่ **<App />** จะมีการใส่ **props** ที่ชื่อ **children** เข้ามาด้วย ซึ่งเจ้า **props** นี้เอง ที่เราจะเอาไว้ render component ลูก ที่ match กับ route ที่เราได้กำหนดไว้จะ อย่างที่ได้บอกไปว่าถ้า **path** เป็น **/about** react มันก็จะ render **<About />** ลงไปยังตำแหน่ง ที่ **this.props.children** วางอยู่นั่นเองจะ

มาตรฐานๆ component ของหน้า **Home** กันบ้างครับ ให้เราสร้างเป็น component ง่ายๆ ไปก่อนแบบนี้จะ

```
import React from 'react';

class Home extends React.Component {
    render() {
        return (
            <div>
                Hello, SiamHTML!
            </div>
        );
    }
}

export default Home;
```

ส่วน component ของหน้า **About** ก็เหมือนกันครับ ให้เราสร้างขึ้นมาแบบง่ายๆ เอาแค่กำหนดข้อความที่จะแสดงให้ตรงกับชื่อ component ก็ได้ครับ

เมื่อสร้าง route และ component ต่างๆ เสร็จทั้งหมดแล้ว ให้เราสร้างไฟล์ **src/js/router.jsx** ขึ้นมาเพื่อเป็น middleware สำหรับทำ routing ผ่าน server ครับ ให้เราใส่โค้ดด้านล่างนี้ลงไป

```
import React from 'react';
import ReactDOM from 'react-dom/server';
import { RoutingContext, match } from 'react-router'
```

```

import routes from './routes.jsx';

export default function(req, res) {
  match({ routes, location:req.url }, (error, redirectLocation, renderProps) => {
    res.render("index.handlebars", {
      markup: ReactDOM.renderToString(<RoutingContext {...renderProps} />)
    });
  });
}

```

จากนั้นให้เรากรับไปที่ไฟล์ `server.js` เพื่อทดสอบโค้ด routing เดิม ที่ทำด้วย Express ออก แล้วเพิ่มโค้ดสำหรับเรียกใช้ middleware ที่เราเอาไว้คุยกับ `react-router` ลงไปแทน

```

// โค้ดสำหรับสร้าง HTTP server
...

import router from './router.jsx';
app.use(router);

```

เสร็จแล้วก็ลองรันดูเลยครับ ให้เราลองกดลิ้งค์ต่างๆ ดู ก็จะพบว่าหน้าเว็บสามารถ render ได้ตรงกับที่เราได้กำหนดไว้ใน `routes.jsx` แล้วล่ะครับ แคมพอกด view source ดูแล้วก็ยังพบว่าเนื้อหาของแต่ละ component นั้น ถูก render ออกมากัด้วย ซึ่งหมายความว่าการทำ SEO สำคัญมาก ขั้นตอนต่อไป เราจะมาแปลงเว็บแอปของเราราให้ได้ feel แบบ SPA ครับ

## ใส่โค้ดสำหรับ Render ผึ้ง Client

ที่นี้เราจะมาทำให้เว็บแอปของเราราให้สามารถ render ที่ผึ้ง client ได้ด้วยครับ พอดีง่ายๆ ก็คือ เวลาเรากดลิ้งค์ไปหน้าอื่นแล้ว มันจะไม่โหลดใหม่ทั้งหน้าแล้ว แต่จะโหลดมาเฉพาะเนื้อหาส่วนที่เปลี่ยนไปเท่านั้น วิธีทำก็ไม่ยากเลยครับ ให้เราสร้างไฟล์ `src/js/client.js` และใส่โค้ดด้านล่างนี้ลงไป

```

// โหลด React มาใช้งาน
import React from 'react';
import ReactDOM from 'react-dom';

// โหลดความสามารถของ react-router มาใช้งาน
import { Router } from 'react-router';
import createBrowserHistory from 'history/lib/createBrowserHistory';

// โหลด route ต่างๆ ที่เราได้กำหนดไว้
import routes from './routes.jsx';

// render ลงไปใน DOM ที่ #content
ReactDOM.render(
  <Router routes={routes} history={createBrowserHistory()} />,
  document.getElementById('content')
);

```

หากลองไล่โค้ดดูจะเห็นว่าหลักการจะคล้ายๆ กับการใช้ `react-router` ที่ทางผึ้ง server เลยครับ แต่สังเกตนะครับว่าเราจะใช้ `ReactDOM.render()` แทน ไม่ได้ใช้ `ReactDOMServer.renderToString` เมื่อฉันกับทางผึ้ง server แล้ว เนื่องจากครั้งนี้ เราจะ render เข้าไปใน DOM เลยครับ ส่วนสาเหตุที่ต้องมีการโหลด `createBrowserHistory` มาใช้งานด้วยก็ เพราะเราต้องการจะลบ `#` ออกจาก url เวลาเมื่อมีการเปลี่ยนหน้าหน้าจอ เช่น

จากนั้น ให้เราเข้าไปแก้ไฟล์ `src/js/components/App.jsx` นิดนึงครับ ในส่วนของ `<header>` ให้เราแก้เป็นแบบนี้

| `<ul>`

```
<li><Link to='/'>Home</Link></li>
<li><Link to='/about'>About</Link></li>
</ul>
```

จะเห็นว่าเราต้องแก้ลิงค์จากเดิมที่ใช้ `<a />` มาเป็น `<Link />` แทนครับ เพื่อเป็นการบอกว่าลิงค์นี้ เป็นลิงค์ที่เราต้องการจะให้ render ด้วยฝั่ง client นะ ส่วนจะลิงค์ไปไหนนั้น เราจะกำหนดด้วย `props` ที่ชื่อว่า `to` ซึ่งค่าที่กำหนดเข้าไปนั้นก็คือ `path` ของ `Route` ที่เราได้กำหนดเอาไว้ในไฟล์ `src/js/routes.jsx` นั้นเองนะ

## รวมไฟล์ JavaScript ด้วย Webpack

มาถึงตรงนี้ จะเห็นว่าการ render ของทั้ง 2 ฝั่ง นั้น มีการแข่งโค้ดกันถูกมั่ยครับ ทั้งส่วนของ route และส่วนของ component เลย ซึ่งการจะแข่งกันได้นั้น เราจะต้องเขียน JavaScript เป็นโมดูลแบบ CommonJS ครับ สำหรับฝั่ง server นั้นสบาย เพราะ Node.js รองรับโมดูลแบบนี้อยู่แล้ว แต่ฝั่ง client มันไม่รู้จักรอโค้ด 所以我们才需要使用工具如 webpack ในการรวมไฟล์ JavaScript ให้สามารถรันบน browser ได้ ซึ่งก่อน ซึ่ง tool ที่ผมชอบใช้นั้นจะเป็น webpack ครับ ให้เราติดตั้งโมดูลเพิ่มตามด้านล่างนี้

```
npm install webpack -g
npm install webpack babel-loader --save-dev
```

จากนั้นให้เราสร้างไฟล์ `webpack.config.js` ขึ้นมา และใส่โค้ดด้านล่างนี้ลงไป

```
var path = require('path');
var webpack = require('webpack');

module.exports = {

  // ให้ webpack เริ่มรวมโค้ดที่ไฟล์ client.js
  entry: path.resolve(__dirname, 'src/js/client.js'),

  // และตั้งชื่อไฟล์ output ว่า bundle.js
  output: {
    path: path.resolve(__dirname, 'public/js'),
    filename: 'bundle.js'
  },

  // อ่านไฟล์นามสกุล .js, .jsx ด้วย Babel
  module: {
    loaders: [
      {
        test: /\.jsx?$/,
        exclude: /node_modules/,
        loader: 'babel-loader'
      }
    ]
  }
};
```

สิ่งสำคัญก็คือการกำหนด `entry point` ครับ ให้เราเลือกไปที่ไฟล์ `src/js/client.js` เพราะไฟล์นี้ถือเป็นจุดเริ่มต้นของโค้ดสำหรับ render ฝั่ง client (เราจะกำหนด `entry point` ไปที่ไฟล์เดียว เนื่องจากไฟล์นั้นมันจะไป `import` โมดูลอื่นๆ ต่อ กันไปเป็นทอดๆ อยู่แล้ว)

จากนั้นก็เข้าไปเพิ่มโค้ดสำหรับโหลด `bundle.js` มาใช้ ที่ `src/templates/index.handlebars` ครับ

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
```

```
<title>Isomorphic with React Tutorial by SiamHTML</title>
</head>
<body>
  <div id="content">{{{{markup}}}}</div>
  <script src="/js/bundle.js"></script>
</body>
</html>
```

จากนั้นก็รัน webpack ได้เลยจะ

| webpack

เสร็จแล้วก็ลองรันหน้าเว็บดูอีกทีครับ เราจะพบว่าเวลากดลิงค์มันจะไม่โหลดใหม่ทั้งหน้าแล้ว ในที่สุดเราก็ทำสำเร็จครับ!

## แนวการการเขียน React

จาก workshop เมื่อกี้ สิ่งที่เราทำไปมันก็แค่โครงสร้างของเว็บแบบ Isomorphic เท่านั้นเองนะครับ T\_\_T ซึ่งขั้นตอนหลังจากนี้ ถึงจะเป็นการเริ่มลงมือสร้าง component ที่เราต้องการจะใช้จริงๆ ขึ้นมาด้วย React ครับ แต่ก่อนจะเริ่มเขียน React เรา มาดูวิธีคิดแบบใหม่ ที่เราจะต้องทำความเข้าใจให้ถ่องแท้กันดีกว่า

### 1. แบ่งหน้าเว็บออกเป็น Component ย่อยๆ ก่อน

ขั้นตอนแรก ให้เราพยายามมองหน้าเว็บออกเป็น component ย่อยๆ ให้ได้ก่อนครับ ส่วนจะใช้เกณฑ์อะไรในการดูว่าส่วนไหนเป็น component บ้าง ผมแนะนำให้แบ่งย่อยเข้าไปถึงระดับที่สิ่งนั้นสามารถอยู่ได้โดยอิสระ โดยที่ไม่ได้ขึ้นอยู่กับ component ใดๆ แล้วครับ หากเขียน React มาถึงจุดๆ นึง เราจะมองออกครับว่าจริงๆ และ เจ้า component มันก็คือ Class ดีๆ นี่เอง ตามหลักของ **Single Responsibility Principle** และ Class ควรจะมีหน้าที่เพียงหนึ่งเดียวเท่านั้นครับ ดังนั้น ให้เราซอย component ให้เล็กลงเรื่อยๆ จน แต่ละ component เหลือเพียงหน้าที่เดียวเท่านั้น

### 2. ประกอบ Component เหล่านั้น ให้แสดงผลออกมาได้

พอได้ component มาทั้งหมดแล้ว ให้เราเขียนโค้ด React เพื่อประกอบ component เหล่านั้นเข้าด้วยกันครับ ในขั้นตอนนี้ เราจะใช้แต่ **props** เท่านั้นในการส่งข้อมูลให้กับ component ที่เป็น parent และก็ให้มันส่งต่อไปยัง component ลูกไปเรื่อยๆ ส่วนข้อมูลที่จะส่งนั้น ขอให้ mock ขึ้นมาเองได้เลยครับ พอทำเสร็จแล้ว เราจะได้เว็บหน้าตาเหมือนจริงเลย เพียงแต่มันจะยังเป็นแค่ static version คือจะยังไม่มีการโต้ตอบใดๆ ครับ

### 3. ดูว่าควรเก็บอะไรไว้ใน State บ้าง

มาถึงขั้นตอนที่เราจะแยกที่สุดครับ นั่นก็คือการมองให้ออกว่าข้อมูลอะไรควรเก็บไว้ใน **state** บ้าง มีหลักง่ายๆ ก็คือ ถ้าข้อมูลมีลักษณะ แบบนี้ เราจะไม่เก็บไว้ใน **state** ครับ

- ถ้า parent สามารถส่งมาให้ได้
- ถ้าข้อมูลนั้นไม่ได้เปลี่ยนแปลงไปตามกาลเวลา
- ถ้าข้อมูลนั้นสามารถคำนวณได้จาก **state** หรือ **props** อีกๆ

บางครั้งอาจสงสัยว่าทำไมเราต้องมาคิดด้วยว่าจะไรมีความอยู่ใน **state** เก็บๆ ไปเลยไม่ได้หรอ ? คำตอบคือมันมีผลต่อ performance ของเรา ราคาระเก็บเฉพาะข้อมูลที่มีขนาดเล็กมากๆ เท่านั้น เพราะหน้าที่จริงๆ ของ **state** ก็คือการเอาไว้ switch UI ตาม event ต่างๆ ที่เกิดขึ้นเท่านั้นเองครับ

## 4. ดูว่าควรใช้ State ให้กับ Component ไหน

ตอนนี้เรารู้แล้วว่าข้อมูลอะไร ที่ต้องเก็บไว้ใน **state** บ้าง ต่อมาเราจะต้องมองให้ออกครับว่าเราควรจะกำหนด **state** ให้กับ component ไหน ขั้นตอนนี้ก็ยากมากเหมือนกับการรับหากเราเริ่มเขียน React ใหม่ๆ แต่หากมีหลักง่ายๆ ดังนี้ครับ

- ถ้ามี component ไหนบ้าง ที่การ **render** ของมันขึ้นอยู่กับ **state**
- หา component ที่เป็น parent ของ component เหล่านั้น และใส่ **state** ให้มัน
- หากไม่มี component ที่เป็น parent ของ component เหล่านั้นเลย ให้สร้าง component ใหม่มาครอบ component เหล่านั้น และค่อยใส่ **state** ให้มัน

พุดง่ายๆ ก็คือในการเขียน React นั้น เราจะใช้วิธีใส่ **state** เอาไว้ที่ตัวแม่ และให้มันส่งข้อมูลที่จำเป็นไปยังลูกๆ ผ่าน **props** ครับ ลอง

**3.8K** Total shares  

## แนวการศึกษาเพิ่มเติม

มาถึงตรงนี้ ผู้คิดว่าพื้นฐานของเราโอเคในระดับหนึ่งแล้วล่ะครับ ที่เหลือก็แค่ฝึกซ้อมจนเขียน React ได้คล่องเท่านั้นเอง เวลาเห็น design มาแล้วต้องรู้เลยว่าจะต้องช่วยอะไรออกมามาเป็น component บ้าง ต้องใส่ **state** ไว้ตรงไหน ตรงนี้ต้องมีช่วงบินสูงๆ ถึงจะทำได้ครับ หรือถ้าใครอยากเป็นเว็บ ผู้ขอแนะนำให้ลองติดตั้ง **React Developer Tools** ดูครับ และเราจะรู้เลยว่าแต่ละเว็บที่ใช้ React ทำนั้น เค้ามีการออกแบบ component อย่างไร ใส่ state อะไร เอาไว้ที่ ไหนบ้าง

ผู้ขอสารภาพให้เจ็บใจเล่นๆ nidneng นะครับว่า หากจะเอา Isomorphic ไปใช้กับ production จริงๆ และ เราจะไม่เขียนโค้ดในรูปแบบที่เราเขียนกันในบทความนี้หรือครับ เพราะมันเป็นแค่โค้ดพื้นฐานสุดๆ ที่ทำให้ Isomorphic ทำงานได้เท่านั้นเอง (อาจมี例外 เช่น PHP เพียงๆ ในช่วงหัดเขียนแรกๆ แต่สุดท้ายก็เปลี่ยนไปใช้ framework อื่น) ซึ่งถ้าผู้เขียนโค้ดที่เค้าใช้กันจริงๆ มาอธิบายด้วยบทความนี้ก็คงจะยากกว่านี้อีกเยอะเลยล่ะครับ ก็ถือจะว่าเนื้อหาในบทความนี้เป็นก้าวแรกในการศึกษาเรื่อง Isomorphic และกันนะครับ ^0^

หากใครจะมาเอาดีทางด้าน Isomorphic หรือจะลองนำไปใช้กับ production ดู ผู้ขอแนะนำให้ไปศึกษาเรื่องพวknี้ต่อเลยครับ

### ECMAScript 2015

project ที่เกี่ยวกับ Isomorphic ต่างๆ บน GitHub มักจะเขียนด้วย **ECMAScript 2015** ครับ ถ้าเราไม่เป็นเลย รับรองว่าไฟล์โค้ดคนอื่นลำบากแน่ๆ

### Flux

แล้วส่วนใหญ่เค้าก็จะใช้ **Flux** เป็น pattern ในการเขียน React กันครับ ลองไปหัดเล่นดูนะครับ และจะพบว่ามันทำให้การเขียน React นั้นเข้าใจง่ายขึ้นเยอะเลย

### Redux

เมื่อเข้าใจหลักการของ Flux และ ผู้ขอแนะนำให้ลองเล่น **Redux** ดูครับ เพราะมันเป็น library ที่นำ Flux ไปต่อยอดให้ดียิ่งขึ้น ซึ่งจริงๆ library ที่นำ Flux ไปต่อยอดนั้น มีเยอะมากๆ เลยนะครับ แต่ที่ผู้ขอแนะนำ Redux ก็เพราะว่า docs ของมันค่อนข้างจะโอเค

แล้ว แล้วคนที่ใช้ก็ถือว่า酵ะพอสมควรเลยครับ ในข้ามโน้นถือว่าเป็นตัวที่น่าเล่นที่สุดแล้ว

## ความรู้สึกหลังลองทำเว็บแบบ Isomorphic

ผมมองว่า Isomorphic คืออนาคตของเว็บนะครับ ข้อดีผมพูดไปหลายครั้งแล้ว ขอพูดถึงข้อเสียมั่งละกัน หลักๆ เลยก็คือ learning curve ที่แอบสูงนิดๆ ครับ อาจจะเป็นเพราะมันยังใหม่อุ่นด้วยมั้ง เราพยายามไม่ค่อยเจอบทความไหนที่อ่านแล้วเข้าใจง่ายๆ เท่าไร

ส่วนข้อเสียอีกเรื่องซึ่งผมเพิ่งเจอมากมาดๆ เลยก็คือ แต่ละส่วนในการทำ Isomorphic มันพัฒนาแยกกันอะ คือวันเดียวเราอัพเดท dependency ที่ เราอาจจะพบว่าโค้ดเดิมของเราใช้งานไม่ได้อีกแล้ว หรือบางที่เราพบว่าบางโนดล้ม bug และการจะแก้ bug นี้ได้ ต้อง อัพเดทเวอร์ชันใหม่ ที่ syntax มันเปลี่ยนยกเครื่องเลย นี่เป็นสิ่งที่เราพบเจอดีบอยๆ ในการทำ Isomorphic ณ วันนี้อะ ก็ซึ่งใจเอาเองนะ ครับว่างานของเราจะมีความเหมาะกับ tool ตัวไหน จะเป็นต้องแคร์เรื่อง SEO มั้ย ? ส่วนตัวผมก็ใช้เกณฑ์เหล่านี้แหละ เป็นตัวตัดสินใจว่าจะใช้ AngularJS หรือ React ดี

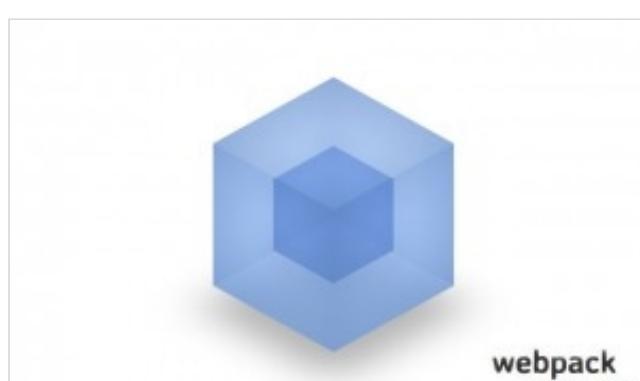
แต่อย่าเพิ่งคิดว่าโลกของเว็บหลังจากนี้ React จะผูกขาดนะครับ เพราะกระแสของ Angular 2 นั้นเริ่มจะมาแล้ว แणในเวอร์ชันใหม่นี้ยัง สามารถทำ server-rendering ได้แล้วด้วย ประเด็นเรื่อง SEO จึงตัดออกไปได้เลย อีกตัวที่ทำได้(นานแล้ว) ก็จะเป็น Meteor ครับ ก็ลองไปศึกษา กันต่อๆ กันดูนะครับ ว่า tool ตัวไหนมันเหมาะสมกับงานของเรา

ก็ต้องทำใจกันหน่อยนะครับ สำหรับชาว Front-end ทั้งหลาย ที่ต้องเคยเรียนรู้อะไรใหม่ๆ อุ่นๆ ตลอดเวลา บ่อยครั้งที่เรารู้สึกว่าพอใช้ตัวนี้ จนเก่งแล้ว ตัวใหม่ที่ดีกว่าดันออกแบบมาอีกละ เอาเป็นว่าผมจะพยายามช่วยกรองมาให้ชั้นนึงก่อนแล้วกันนะครับว่า tool ตัวไหนดูดี มีอนาคต SiamHTML จะรีบนำมาวิวิห์ได้อ่านกันก่อนใครแน่นอน และพอกันใหม่ๆ กับความหน้าครับผม

## RECOMMENDED



Flux คืออะไร ? + สอนวิธีนำไปใช้กับ React



Webpack คืออะไร ? + สอนวิธีใช้ร่วมกับ React



ECMAScript 6 คืออะไร ? + สอนวิธีใช้



Jade คืออะไร ? + สอนวิธีใช้ร่วมกับ Express