Nhóm trưởng: Phan Sông Núi

1212283@student.hcmus.edu.vn

GV: Thầy Hoàng Anh Tú

NHÓM: N2AX

ỨNG DỤNG PHÂN TÁN

BÁO CÁO SEMINAR CHỦ ĐỀ: REACT-ALT

**THÔNG TIN NHÓM SINH VIÊN THỰC HIỆN**

|  |  |  |  |
| --- | --- | --- | --- |
| Nhóm:N2AX | | Số lượng thành viên: 4 | |
| MSSV | Họ tên | Email | Điện thoại |
| 1212015 | Phan Long Ánh | [1212015@student.hcmus.edu.vn](mailto:1212015@student.hcmus.edu.vn) |  |
| 1212283 | Phan Sông Núi | [1212283@student.hcmus.edu.vn](mailto:1212283@student.hcmus.edu.vn) | 01672522416 |
| 1312379 | Nguyễn Thị Bích Ngân | [1313379@student.hcmus.edu.vn](mailto:1313379@student.hcmus.edu.vn) |  |
| 1212535 | Bùi Thị Thanh Xuân | [1212535@student.hcmus.edu.vn](mailto:1212535@student.hcmus.edu.vn) |  |

**BẢNG PHÂN CÔNG CÔNG VIỆC VÀ ĐÁNH GIÁ THÀNH VIÊN NHÓM**

|  |  |  |  |
| --- | --- | --- | --- |
| Công việc thực hiện | Người thực hiện | Mức độ hoàn thành | Đánh giá của nhóm |
| Tìm hiểu thông tin giới thiệu sơ lược về React, JSX, thực hiện demo | 1312379 – Nguyễn Thị Bích Ngân | 70% | 7/10 |
| Tìm hiểu về props, state, thực hiện demo | 1212535 – Bùi Thị Thanh Xuân | 70% | 7/10 |
| Tìm hiểu về flux, thực hiện demo | 1212283 – Phan Sông Núi | 60% | 6/10 |
| Tìm hiểu về Alt, thực hiện demo | 1212015 – Phan Long Ánh | 70% | 7/10 |

**MỤC LỤC**

[I. Sơ lược về React 3](#_Toc452153604)

[1. React là gì? 3](#_Toc452153605)

[a. Ưu điểm: 3](#_Toc452153606)

[b. Nhược điểm 4](#_Toc452153607)

[2. Có nên dùng React? 4](#_Toc452153608)

[II. Các khái niệm trong React 5](#_Toc452153609)

[1. JSX: 5](#_Toc452153610)

[a. JSX không phải một ngôn ngữ template: 5](#_Toc452153611)

[b. JSX hoàn toàn không bắt buộc, nhưng lại thực sự hữu ích: 6](#_Toc452153612)

[2. Props: 7](#_Toc452153613)

[3. State: 9](#_Toc452153614)

[4. Mối quan hệ giữa props và state: 9](#_Toc452153615)

[5. Flux 9](#_Toc452153616)

[6. Alt 14](#_Toc452153617)

[III. Các tài liệu tham khảo của báo cáo: 20](#_Toc452153618)

**NỘI DUNG**

# Sơ lược về React

## React là gì?

* React đơn thuần chỉ phục vụ cho tầng View (V trong mô hình MVC)
* React cung cấp cho bạn một ngôn ngữ tạo template và các function nhằm render ra các mã HTML.

Tuy nhiên ta hoàn toàn không thể xây dựng một ứng dụng đầy đủ nếu chỉ sử dụng mỗi React.

### Ưu điểm:

1. **Bạn sẽ luôn luôn biết được component của bạn sẽ sinh ra mã HTML như thế nào chỉ bằng việc nhìn vào một file source.**

Nếu không sử dụng bất kỳ Javascript MVC framework nào, bạn có thể sẽ làm nó như sau: HTML

*Markup:*

<header >

<div class="name"></div>

</header>

*Mã javascript:*

$.post('/login', credentials, function( user ) {

// Modify the DOM here

$('header .name').show().text( user.name );

});

*Với yêu cầu như trên, bạn có thể viết nó với React như sau:*

render: function() {

return <header >

{ this.state.name ? this.state.name : null }

</header >;

}

Chúng ta có thể biết ngay được component trên sẽ render ra mã HTML như thế nào

1. **Tích hợp giữa javascript và HTML vào trong JSX làm cho các component dễ hiểu hơn**

Theo truyền thống, chúng ta thường tách các file HTML và các function Javascript thành các file riêng rẽ, điều này dẫn đến các file Javascript sẽ chứa tất cả function cho một "page", do đó để hiểu được luồng làm việc của chương trình bạn sẽ phải đọc code từ JS > HTML > JS > HTML ... Mất thời gian rất nhiều.

Việc chèn các đoạn mã js trực tiếp vào các file markup và sau đó đóng gói chúng lại dưới dạng "portable". Những đoạn mã js khi đó sẽ xử lý trực tiếp với file markup mà nó được tích hợp vào và bạn không phải mất công lần mò những thứ không liên quan.

1. **Bạn có thể render React phía server**

Việc chỉ render phía client lý giải tại sao các website như Soundcloud rất chậm, trong khi đó Stack Overflow (chỉ render phía server)lại rất nhanh.

### Nhược điểm

React chỉ phục vụ cho tầng view:

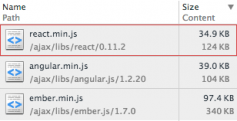
1. **Vậy nên React sẽ không cung cấp cho bạn:**

* Hệ thống event (ngoại trừ các event thuần của DOM)
* Các giải pháp về AJAX
* Các quy chuẩn về tầng dữ liệu
* Promises
* Phần lớn những thứ có trong một framework
* Hoặc các ý tưởng thực thi các điều ở trên

Nếu chỉ dùng duy nhất React, nó vô dụng đối với các yêu cầu thực tế.

1. **Tài liệu nghèo nàn:**   
   Offical document của React gây khó khăn với cả những người biết và đã làm việc với React một thời gian. Khá là khó để tìm kiếm các vấn đề mình gặp phải.
2. **React khá nặng nếu so với các framework khác:**

React có kích thước tương tương với Angular (Khoảng 35kb so với 39kb của Angular). Trong khi đó Angular là một framework hoàn chỉnh.



## Có nên dùng React?

***Đây là những lý do tại sao bạn nên sử dụng React:***

* Hiệu quả khi làm việc cùng team, áp dụng triệt để các pattern UI và workflow
* Code UI dễ đọc và bảo trì
* Component hóa UI là tương lai của phát triển web, và bạn cần bắt đầu nó từ ngay bây giờ.

**Đây là những lý do tại sao bạn nên nghĩ kỹ trước khi chuyển sang React:**

* Bạn sẽ cần thời gian ban đầu để làm quen với React như tìm hiểu về: props,state và component communication, chúng không đơn giản trong khi đó tài liệu thì sơ sài.
* React không hỗ trợ bất cứ trình duyệt nào có version thấp hơn IE8, và sẽ không bao giờ.
* Nếu ứng dụng của bạn không có nhiều page động, bạn sẽ phải implement rất nhiều thứ chỉ để phục vụ cho những nhu cầu nhỏ.

React là một thư viện mới và vì vậy việc xử lý sự kiện, giao tiếp giữa các component không theo một chuẩn nào cả. Dó đó bạn sẽ phải build lại các component từ đầu (dropdown, modal, lightboxes ...) - hoặc chờ người ta build để bạn sử dụng lại.

# Các khái niệm trong React

## JSX:

### JSX không phải một ngôn ngữ template:

Điều đầu tiên dễ dàng nhận thấy là JSX không phải một ngôn ngữ template. JSX chỉ đơn giản là một cú pháp thay thế JavaScript, mặc dù cú pháp đó là đặc trưng của React (ít nhất ở thời điểm hiện tại).

Ví dụ trên được viết lại như sau:

React.render(

React.createElement('div', {}, 'Hello, world!'),

document.body

);

Khi sử dụng React, bạn thường sẽ không làm việc trực tiếp với DOM. Thay vào đó, bạn tạo các thành phần Virtual DOM.

Virtual DOM thực ra là các JSON objects. Chúng là thể hiện của một cấu trúc bên dưới DOM nhưng không mang bất cứ đặc tính nào của các DOM thật. React sẽ convert các thành phần này thành các thành phần DOM thật (actual/real DOM) khi cần thiết.

React.createElement là một phương thức được sử dụng để tạo ra các JSON objects này. Cú pháp như sau:

React.createElement(elementNameOrClass, props, children...);

JSX (đại khái là) thực hiện việc chuyển đổi đoạn code trên như sau:

1. Nếu tham số đầu tiên là một chuỗi ánh xạ của một thẻ HTML nào đó thì sử dụng component class như là thẻ HTML đó.
2. Nếu tag name không phải là một HTML element, cứ coi như nó là một biến cục bộ có tên trỏ vào một component class tùy chọn.
3. Các thuộc tính được convert thành một đối tượng và được truyền vào như tham số thứ hai.
4. Các thành phần con được truyền vào như các tham số còn lại.

Ví dụ về JSX:

<div id="greeting-container" className="container">

<Greeting name="World"/>

</div>

Nó có thể được convert thành

React.createElement("div",

{

id: "greeting-container",

className: "container"

},

React.createElement(Greeting, {name: "World"})

)

JSX chỉ convert các đánh dấu XML thành code JavaScript. Nó tương tự như là bạn đang gõ code Javascript vậy.

Bạn có thể sử dụng tối đa sức mạnh của JavaScript và tất cả mọi thứ bạn đã học về nó như các biểu thức, cấu trúc hàm, hay bất kỳ các tính năng nào khác trong views của bạn. Bạn không cần viết các helpers đặc trưng cho template language của mình mà chỉ cần sử dụng Javascript

var names = ['Moe', 'Larry', 'Curly'];

React.render(

<div>

{

names.map(function (name) {

return <div>Hello, {name}!</div>

})

}

</div>,

document.body

);

Dấu ngoặc {} cho phép chúng ta sử dụng Javascript thuần, nó không phải là một cơ chế escape dữ liệu. Chú ý là bên trong map function, chúng ta trả về một markup là một thẻ div, bên trong markup đó chúng ta lại có một biểu thức Javascipt. Đây là một ví dụ rõ ràng về việc JSX không phải là một template language.

### JSX hoàn toàn không bắt buộc, nhưng lại thực sự hữu ích:

JSX thực sự hữu ích vì hai lý do:

* Nó cung cấp các biểu thức viết tắt (shorthand) hữu ích cho một số React boilerplate mà bạn bắt buộc phải sử dụng dù muốn hay không.
* Nó làm HTML markup của bạn trông giống như HTML markup.

Lý do đầu tiên đã khá rõ ràng qua các ví dụ trên. Các ví dụ về JSX ngắn hơn so với các ví dụ về JS tương ứng. Hãy xem xét ví dụ sau đây:

var R = React.DOM;

Greeting = React.createFactory(GreetingClass);

React.render(

R.div({id: "greeting-container", className: "container"},

Greeting({name: "World"})

),

document.body

);

React.DOM cung cấp một tập các factories cho các thẻ HTML. React.createFactory chỉ là một helper ràng buộc component class của bạn với React.createElement để bạn có thể tạo các factories cho riêng mình. Theo quan điểm của tôi, vấn đề khá lớn ở đây là code Javascript trộn trong các markup. Hãy xem markup sau:

**<p>**

**The live example is the same. The only difference is that we render to <code>mountNode</code>, which is just the DOM node for the example.**

**</p>**

Khi chuyển sang Javascript sẽ rất khó nhìn

R.p({},

'The live example is the same. The only difference is that we render to ',

R.code({},

'mountNode'

),

', ' +

'which is just the DOM node for the example.'

)

## Props:

* Props là một attribute của Component, có thể sử dụng props như một Object hoặc một Function.
* Props chứa giá trị được chuyển từ bên ngoài vào trong Component
* PropTypes: sử dụng khi người dùng muốn validate props

var Avatar = React.createClass({

propTypes: {

name: React.PropTypes.string.isRequired,

id: React.PropTypes.number.isRequired,

width: React.PropTypes.number.isRequired,

height: React.PropTypes.number.isRequired,

alt: React.PropTypes.string

},

render() {

var src = `/img/avatar/${this.props.id}.png`;

return (

<div>

<img src={src} width={this.props.width} height={this.props.height} alt={this.props.alt} />

<span>{this.props.name}</span>

</div>

);

}

});

<Avatar name="foo" width={100} height={100} />

* Default Prop value

var Hello = React.createClass({

getDefaultProps() {

return {

name: "React"

};

},

render() {

return <div>Hello {this.props.name}</div>

}

});

* **setProps và replaceProps**

<!DOCTYPE html>

<html>

<head>

<meta http-equiv='Content-type' content='text/html; charset=utf-8'>

<title>Basic Example Props</title>

</head>

<body>

<script

src="https://cdnjs.cloudflare.com/ajax/libs/react/0.13.3/react.js"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/react/0.13.3/JSXTransformer.js"></script>

<script type="text/jsx">

var Avatar = React.createClass({

propTypes: {

name: React.PropTypes.string.isRequired,

width: React.PropTypes.number.isRequired,

height: React.PropTypes.number.isRequired

},

render() {

var src='http://canime.files.wordpress.com/2010/05/mask-dtb.jpg';

return (

<div>

<img src={src} width={this.props.width}

height={this.props.height} alt="alt" />

<span>{this.props.name}</span>

</div>

);

}

});

var AvatarEl = <Avatar name="Foo" width={100} height={100}/>;

var AvatarComponent = React.render(AvatarEl, document.body);

</script>

</body>

</html>

setProps(): merge property nào mà nó xét

AvatarComponent.setProps({name: "Bar"}, function(){

alert("AvatarComponent setProps Done!");

});

replaceProps(): delete các props tồn tại trước đó và thay bởi các properties mới.

AvatarComponent.replaceProps({name: "Bar-Foo"}, function(){

alert("AvatarComponent replaceProps Done!");

});

## State:

Như là một biến **private** của Component, dùng cho việc thay đổi nội bộ trong Component.

Về cơ bản cấu trúc một Component có state như sau:

method render()

method getIntialState(): initial giá trị state cho Component.

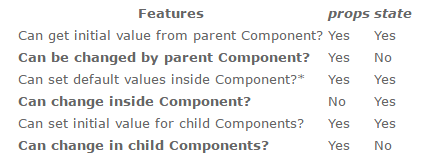
method handle Event: đây là method dùng để handle sự kiện trong Component. Hay cũng chính là callback function của event.

this.setState: được gọi trong method onClick, sẽ thay đổi data trong Component -> trigger render -> rerender View của Component.

## Mối quan hệ giữa props và state:

*props* và *state* đều là **plain JS objects**

*props* và *state* đều trigger **render update** khi thay đổi

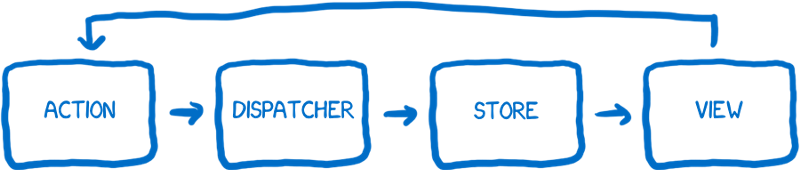


## Flux

Flux và ReactJS cùng được tạo ra bởi Facebook để giải quyết một số những vấn đề rất đặc thù của bản thân Facebook, Tuy vậy 2 công cụ này có thể được sử dụng hoàn toàn riêng biệt.

Facebook thời gian đầu có một notification bug rất phổ biến. Khi bạn login vào Facebook, bạn thấy một notification ở phần message. Khi click vào tin nhắn thì notification biến mất nhưng không hề có một tin nhắn mới nào. Sau một vài phút lướt Facebook thì hiện tượng trên lại lặp lại: notification lại hiện ra và click xong thì lại không có tin nhắn nào....

Kiến trúc Flux:



**The Action Creator:**

Thành phần đầu tiên gọi là The Action Creator, giữ nhiệm vụ tạo ra các action, là bước đầu tiên trong luồng mà các thay đổi và tương tác đều đi qua. Bất cứ khi nào trạng thái của web app hay là render của view thay đổi thì đầu tiên là một hành động sẽ được tạo ra.

**The dispatcher:**

The dispatcher về cơ bản là một tập hợp rất nhiều các callbacks, luôn biết trước một danh sách các store để gửi action đến. Khi một action được gửi đến The dispatcher, sẽ gửi nó đến store tương ứng theo quy tắc đồng bộ.

Flux dispatcher khác với dispatcher ở rất nhiều các kiến trúc khác. Action được gửi đến toàn bộ các store đăng ký với dispatcher không kể type là gì. Mỗi store sẽ nhận "nghe" từ tất cả các action và tự filter để xử lý.

**The store:**

Và cuối cùng là The store, giữ toàn bộ các trạng thái và logic chuyển trạng thái của app. Tất cả mọi thay đổi trạng thái đều được thực thi trực tiếp ở đây. Mỗi khi bạn muốn thay đổi một trạng thái, bạn cần phải tạo một action, submit vào The action creator, đi qua The dispatcher rồi mới được The store xử lý.

Như nói ở phần trên thì một store sẽ nhận rất nhiều các action, và trong store thường sẽ có một cấu trúc switch để quyết định xem có cần phải quan tâm đến action hay không.

**The controller view and the view**

The view có nhiệm vụ thu nhận lệnh thay đổi trạng thái và render hiển thị, cũng như nhận input từ người dùng.

Một view như là một nhân viên trình bày. Anh ta chỉ biết đến dữ liệu và biểu diễn dữ liệu đó (thành HTML) cho người dùng. Controller view như là một nhà quản lý cấp nhỏ đứng giữa store và view, nhận thông báo khi trạng thái thay đổi, tổng hợp những nội dung cần thay đổi và truyền đến những view trực thuộc.

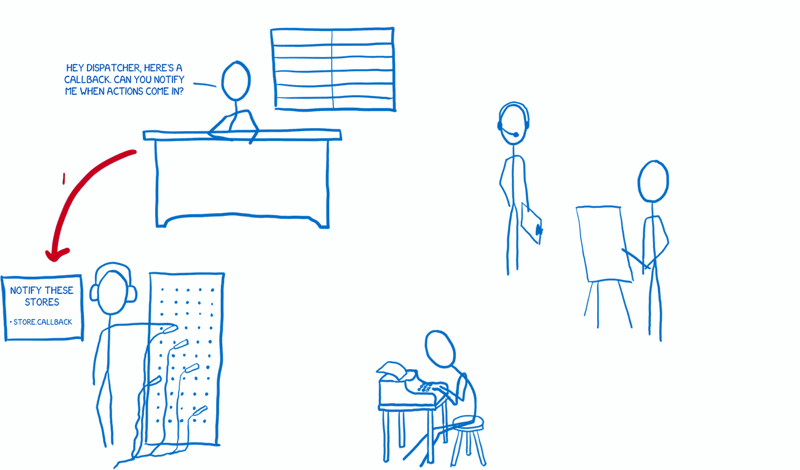
**Cơ chế hoạt động của Flux:**

Setup:

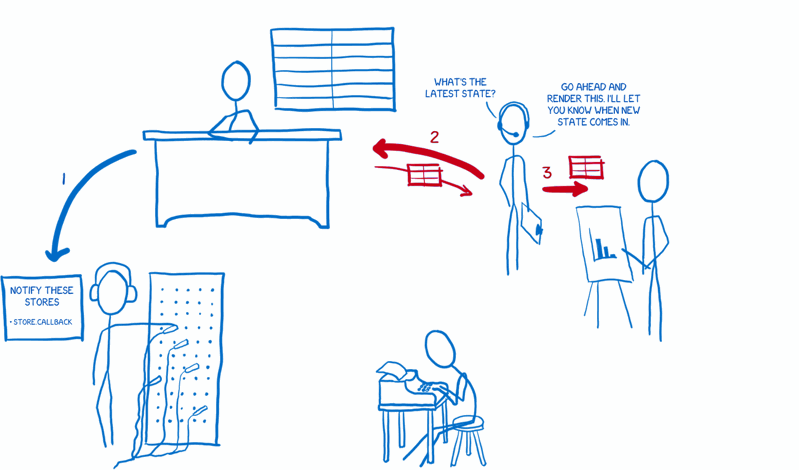
Sau đây là phần setup chỉ diễn ra 1 lần lúc lần đầu vào app.

1. Store liên lạc với dispatcher

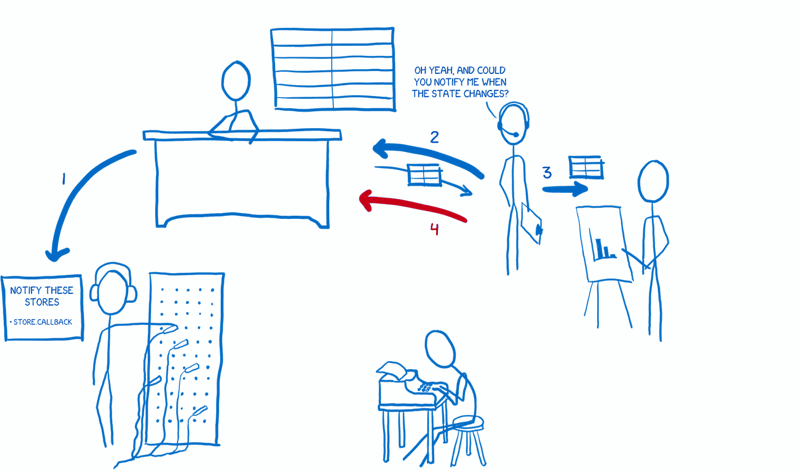
Đây là callback khi nào có action thì gửi tín hiệu.



2) Controller hỏi Store về trạng thái cuối.  
3) Sau khi Store đưa trạng thái cuối cho Controller, Controller đưa cho View để render hiển thị.

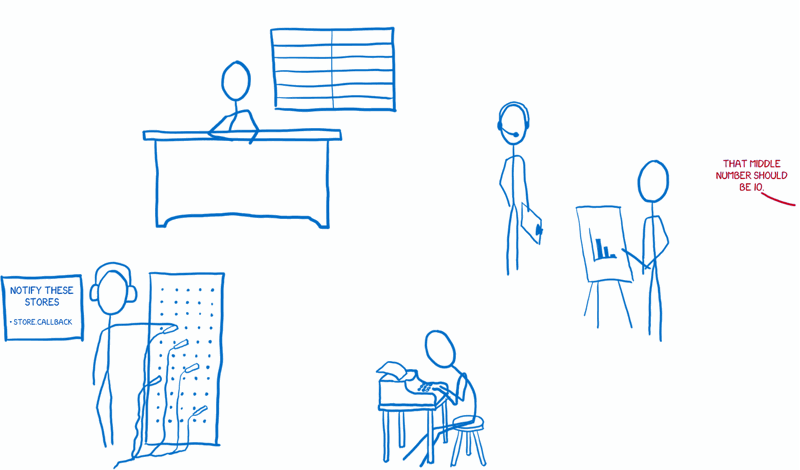


4) Controller và View cũng giao tiếp với Store:

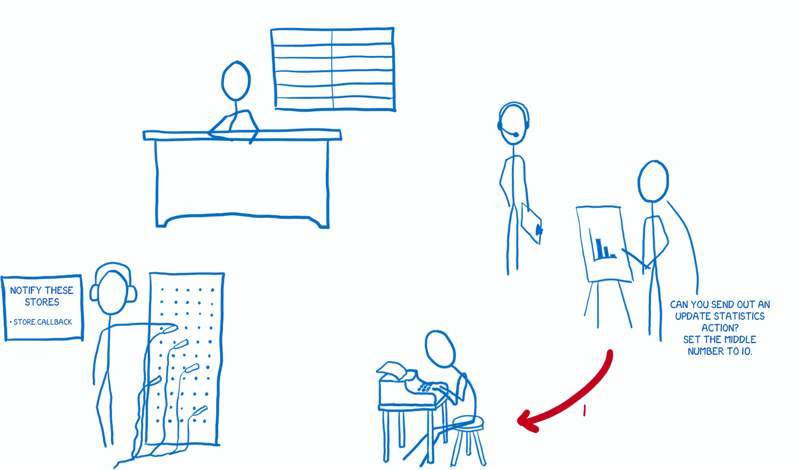


**Luồng dữ liệu**

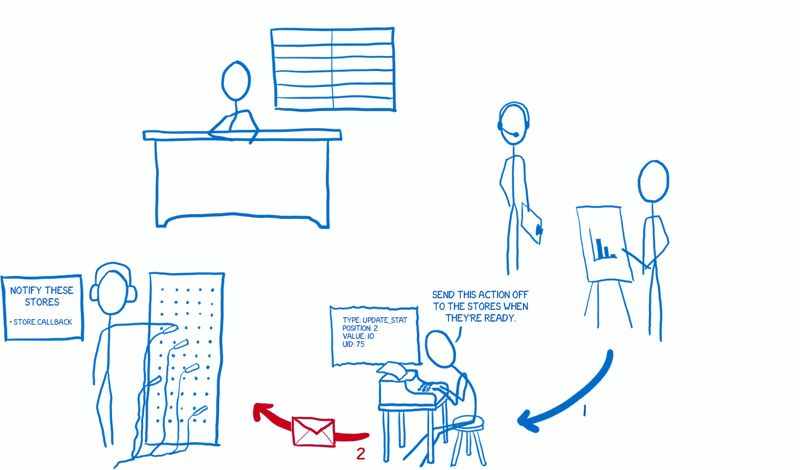
Sau phần setup thì web app đã sẵn sàng nhận input từ người dùng. Nếu khi này trigger một action bằng cách để người dùng tạo ra một sự thay đổi.



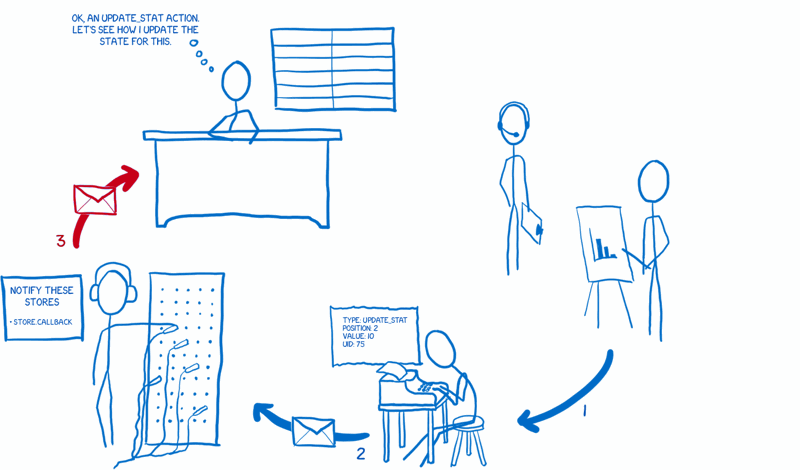
1. View nói với Action Creator tạo ra một action



1. Action Creator tạo ra một action và chuyển qua dispatcher

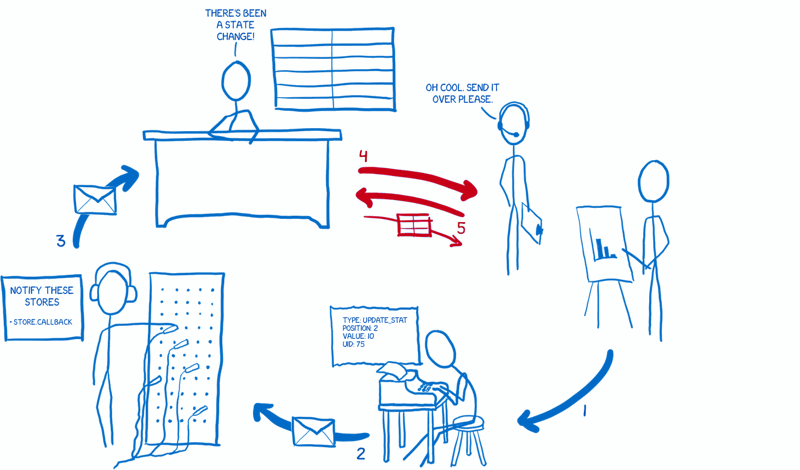


1. Action dispatcher gửi lại action cho tất cả các Store liên quan. Mỗi Store nhận thông báo xong tự quyết định xem có thay đổi trạng thái nào dựa vào action hay không.

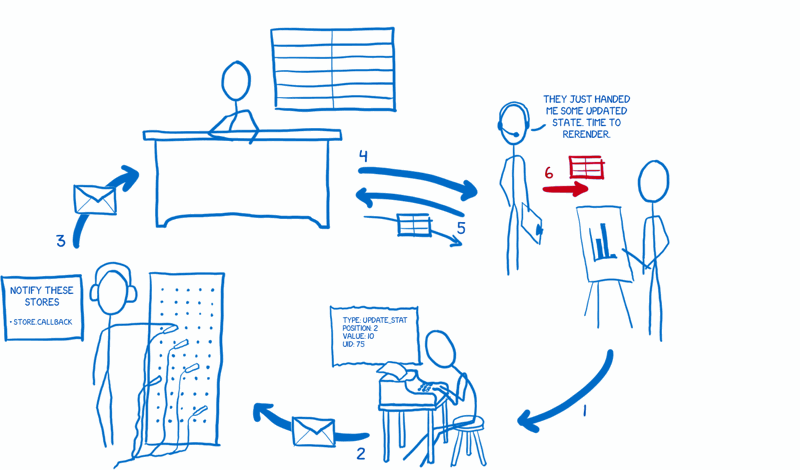


4) Sau khi thay đổi trạng thái, Store lại thông báo tiếp cho View controller đã đăng ký.

5) View controller hỏi lại Store cụ thể về trạng thái mới



6) Sau khi Store đưa lại thông tin cụ thể. View controller sẽ nói với các View của mình render lại hiển thị.



## Alt

Alt là gì:

* Alt là một thư viện hổ trợ cho cho việc điều khiển trạng thái của các ứng dụng Java Script. Được mô hình hóa sau flux

1. Cài đặt alt

“npm install alt”

1. Cấu trúc điển hình

|  |
| --- |
| your\_project  |--actions/  | |--MyActions.js  |--stores/  | |--MyStore.js  |--components/  | |--MyComponent.jsx  |--alt.js  |--app.js |

1. Create alt

* Gọi alt.js

|  |
| --- |
| var Alt = require('alt');  var alt = new Alt();  module.exports = alt; |

1. Tạo action

Tạo một action trong một class.

Sử dụng “*this.dispatch*” thông qua Dispatcher và store

Tạo action sử dụng *alt.createActions*

|  |
| --- |
| var alt = require('../alt');  class LocationActions {  updateLocations(locations) {  return locations;  }  }  module.exports = alt.createActions(LocationActions); |

1. Tạo store
2. Tương tự như actions, cú pháp store có thể thiết kế tùy ý, có thể sử dụng thiết kế mẫu

|  |
| --- |
| class LocationStore {  constructor() {  }  } |

1. Tạo phương thức

this.locations = [];

1. Định nghĩa các phương thức xử lý actions

|  |
| --- |
| handleUpdateLocations(locations) {  this.locations = locations;  // optionally return false to suppress the store change event  } |

1. Cài đặt các ràng buộc cho các action

|  |
| --- |
| this.bindListeners({  handleUpdateLocations: LocationActions.UPDATE\_LOCATIONS  }); |

1. Gọi store

module.exports = alt.createStore(LocationStore, 'LocationStore');

* store hoàn chỉnh stores/LocationStore.js

|  |
| --- |
| var alt = require('../alt');  var LocationActions = require('../actions/LocationActions');  class LocationStore {  constructor() {  this.locations = [];  this.bindListeners({  handleUpdateLocations: LocationActions.UPDATE\_LOCATIONS  });  }  handleUpdateLocations(locations) {  this.locations = locations;  }  }  module.exports = alt.createStore(LocationStore,'LocationStore'); |

1. Sử dụng View

components/Locations.jsx

|  |
| --- |
| var React = require('react');  var LocationStore = require('../stores/LocationStore');  var Locations = React.createClass({  getInitialState() {  return LocationStore.getState();  }, // lắng nghe store và lấy giá trị  componentDidMount() {  LocationStore.listen(this.onChange);  }, lắng nge sự thay đỗi của một trạng thái trong store  componentWillUnmount() {  LocationStore.unlisten(this.onChange);  }, // xóa sự kiện nghe  onChange(state) {  this.setState(state);  },  render() {  return (  <ul>  {this.state.locations.map((location) => {  return (  <li>{location.name}</li>  );  })}  </ul>  );  }  });  module.exports = Locations; |

1. Nạp dữ liệu

Để lấy dữ liệu từ máy chủ và xử lý lỗi. Sử dụng setTimeout và Promise giống như là tạo một yêu cầu để sử dụng fetch API

1. Tạo LocationStores sources/LocationSource.js

|  |
| --- |
| var LocationSource = {  fetch: function () {  // returning a Promise because that is what fetch does.  return new Promise(function (resolve, reject) {  // simulate an asynchronous action where data is fetched on  // a remote server somewhere.  setTimeout(function () {  // resolve with some mock data  resolve(mockData);  }, 250);  });  }  }; |

1. Thêm action fetchLocations trong fetch và gọi updateLocations

|  |
| --- |
| fetchLocations() {  return (dispatch) => {  // sự kiện để ta gọi dữ liệu  dispatch();  LocationSource.fetch()  .then((locations) => {  // điều khiển các actions  this.updateLocations(locations);  })  .catch((errorMessage) => {  this.locationsFailed(errorMessage);  });  }  }  locationsFailed(errorMessage) {  return errorMessage;  }// lỗi nếu không có sử liệu khả dụng |

1. Thêm action xử lý bindListeners để xử lý các lỗi có thể xãy ra

stores/LocationStore.js

|  |
| --- |
| class LocationStore {  constructor() {  this.locations = [];  this.errorMessage = null;  this.bindListeners({  handleUpdateLocations: LocationActions.UPDATE\_LOCATIONS,  handleFetchLocations: LocationActions.FETCH\_LOCATIONS,  handleLocationsFailed: LocationActions.LOCATIONS\_FAILED  });  }  handleUpdateLocations(locations) {  this.locations = locations;  this.errorMessage = null;  }  handleFetchLocations() {  // reset the array while we're fetching new locations so React can  // be smart and render a spinner for us since the data is empty.  this.locations = [];  }  handleLocationsFailed(errorMessage) {  this.errorMessage = errorMessage;  }  } |

1. Hiển thị thông báo lỗi và hiện liên kết nếu lấy dữ liệu thành công

|  |
| --- |
| componentDidMount() {  LocationStore.listen(this.onChange);  LocationActions.fetchLocations();  },  render() {  if (this.state.errorMessage) {  return (  <div>Something is wrong</div>  );  }  if (!this.state.locations.length) {  return (  <div>  <img src="/my-cool-spinner.gif" />  </div>  )  }  return (  <ul>  {this.state.locations.map((location) => {  return (  <li>{location.name}</li>  );  })}  </ul>  );  } |

1. Phụ thuộc dữ liệu
2. Thêm actions vào LocationActions actions/LocationActions.js

|  |
| --- |
| favoriteLocation(locationId) {  this.dispatch(locationId);  } |

1. Tạo một FavoriesStore

stores/FavoritesStore.js

|  |
| --- |
| var alt = require('../alt');  var LocationActions = require('../actions/LocationActions');  class FavoritesStore {  constructor() {  this.locations = [];  this.bindListeners({  addFavoriteLocation: LocationActions.FAVORITE\_LOCATION  });  }  addFavoriteLocation(location) {  this.locations.push(location);  }  }  module.exports = alt.createStore(FavoritesStore, 'FavoritesStore'); |

1. Thiết lập liên kết waitFor trong LocationStore
   1. Tạo action xử lý trong Store

|  |
| --- |
| this.bindListeners({  handleUpdateLocations: LocationActions.UPDATE\_LOCATIONS,  handleFetchLocations: LocationActions.FETCH\_LOCATIONS,  handleLocationsFailed: LocationActions.LOCATIONS\_FAILED,  setFavorites: LocationActions.FAVORITE\_LOCATION  }); |

* 1. Tạo các action với waitFor

|  |
| --- |
| resetAllFavorites() {  this.locations = this.locations.map((location) => {  return {  id: location.id,  name: location.name,  has\_favorite: false  };  });  }  setFavorites(location) {  this.waitFor(FavoritesStore);  var favoritedLocations = FavoritesStore.getState().locations;  this.resetAllFavorites();  favoritedLocations.forEach((location) => {  // find each location in the array  for (var i = 0; i < this.locations.length; i += 1) {  // set has\_favorite to true  if (this.locations[i].id === location.id) {  this.locations[i].has\_favorite = true;  break;  }  }  });  } |

# Các tài liệu tham khảo của báo cáo:

* Thông tin từ trang chủ của React, Alt
* [ReactJS For Stupid People](http://blog.andrewray.me/reactjs-for-stupid-people/) Nguồn: <http://fsd14.com>  
  Tài liệu khai thác dữ liệu và ứng dụng của khoa công nghệ thông tin DHKHTN TP.HCM.
* <http://coddingdaily.blogspot.com/2015/06/reactjs-props-la-gi.html>
* <http://coddingdaily.blogspot.com/2015/06/reactjs-bai-5-state-la-gi.html>
* <http://coddingdaily.blogspot.com/2015/06/reactjs-bai-6-moi-quan-he-giua-props-va.html>
* <http://kipalog.com/posts/Huong-dan-va-giai-thich-Flux-bang-hinh-ve>
* Các nguồn khác từ internet.