Mục lục

[1. Giới thiệu 2](#_Toc58155621)

[2. Lý thuyết 2](#_Toc58155622)

[2.1 Nodejs là gì? 2](#_Toc58155623)

[2.2 NPM 3](#_Toc58155624)

[2.3 Express là gì? 5](#_Toc58155625)

[2.4 ReactJS là gì? 12](#_Toc58155626)

[2.4.1 Component 12](#_Toc58155627)

[2.4.2 JSX — Javascript Syntax Extension 13](#_Toc58155628)

[2.4.3 Multiple components 13](#_Toc58155629)

[2.4.4 Props & State là gì? 14](#_Toc58155630)

[2.5 Redux là gì? 19](#_Toc58155631)

[2.5.1 Tại sao phải có redux 19](#_Toc58155632)

[2.5.2 Hiểu cách redux làm việc 21](#_Toc58155633)

[2.5.3 Nguyên lý vận hành 24](#_Toc58155634)

1. Giới thiệu

Bối cảnh: Hiện nay đại học Thủy Lợi thực hiện điểm danh sinh viên trên giảng đường hoàn toàn thủ công, gây mất thời gian giảng dạy, thời gian nghỉ giữa giờ.

Bài toán đặt ra: xây dựng hệ thống điểm danh tự động để không cần điểm danh thủ công

Hướng giải quyết: Sinh viên sẽ bật bluetooth trên điện thoại của mình, đặt tên bluetooth theo mã sinh viên. Giảng viên đến giờ vào lớp sẽ quét bluetooth tất cả các thiết bị xung quanh, và dựa vào mã sinh viên để điểm danh.

Ưu điểm: Hoàn toàn tự động, nhanh chóng, có thể ghi lại chính xác thời gian điểm danh nên ghi lại được cả các trường hợp đi muộn. Phát hiện được điểm danh hộ nếu 1 máy điểm danh 2 lần với 2 mã sinh viên khác nhau

Nhược điểm: sinh viên phải có điện thoại có bluetooth, nếu không sẽ phải điểm danh thủ công.

Công nghệ sử dụng: Hệ thống sẽ gồm 3 phần:

* Backend restful api: viết bằng nodejs-express, cơ sở dữ liệu mysql
* Frontend web để quản lý giảng viên, sinh viên, giờ học, lịch học… viết bằng reactjs
* Android app cho giảng viên thực hiện điểm danh (sinh viên chỉ cần phát bluetooth, không cần cài app), viết bằng java

1. Lý thuyết
   1. Nodejs là gì?

* Nodejs là một **nền tảng** (Platform) phát triển độc lập được xây dựng ở trên Javascript Runtime của Chrome mà chúng ta có thể xây dựng được các ứng dụng mạng một cách nhanh chóng và dễ dàng mở rộng.
* Nodejs được xây dựng và phát triển từ năm 2009, bảo trợ bởi công ty Joyent, trụ sở tại California, Hoa Kỳ. Dù sao thì chúng ta cũng nên biết qua một chút chút lịch sử của thứ mà chúng ta đang học một chút chứ nhỉ? =))
* Phần Core bên dưới của Nodejs được viết hầu hết bằng C++ nên cho tốc độ xử lý và hiệu năng khá cao.
* Nodejs tạo ra được các ứng dụng có tốc độ xử lý nhanh, realtime thời gian thực.
* Nodejs áp dụng cho các sản phẩm có lượng truy cập lớn, cần mở rộng nhanh, cần đổi mới công nghệ, hoặc tạo ra các dự án Startup nhanh nhất có thể.
  1. NPM

NMP là viết tắt của Node package manager là một công cụ tạo và quản lý các thư viện lập trình Javascript cho [Node.js](https://nodejs.org/). Trong cộng đồng Javascript, các lập trình viên chia sẻ hàng trăm nghìn các thư viện với các đoạn code đã thực hiện sẵn một chức năng nào đó. Nó giúp cho các dự án mới tránh phải viết lại các thành phần cơ bản, các thư viện lập trình hay thậm chí cả các [framework](https://topdev.vn/blog/framework-la-gi/).

Với npm , công việc sẽ đơn giản đi rất nhiều, chúng giúp bạn thực hiện việc quản lý đơn giản hơn rất nhiều. Các thư viện đều có sẵn trên npm, bạn chạy một dòng lệnh để tải về và dễ dàng include chúng hơn.

Mỗi đoạn code này có thể phụ thuộc vào rất nhiều các mã nguồn mở khác, thật may mắn khi các công cụ quản lý thư viện ra đời, nếu không sẽ mất rất nhiều công sức trong việc quản lý các thư viện này.

Cộng đồng sử dụng npm rất lớn, hàng nghìn các thư viện được phát hành, hỗ trợ Javascript ES6, React, Express, Grunt, Duo… Hiện nay cũng đã xuất hiện thêm Yarn một công cụ tương tự npm, được Facebook phát triển với nhiều tính năng vượt trội có khả năng sẽ thay thế npm.

Nếu như bạn từng code Php thì sẽ biết [Composer](https://topdev.vn/blog/composer-la-gi/) là công cụ quản lý thư viện của nó, tương tự như NPM là công cụ quản lý thư viện Javascript.

**Cài đặt NPM**:

npm có sẵn khi bạn tải NodeJS về. Để kiểm tra xem trên hệ thống của bạn đã được cài npm chưa chúng ta sử dụng lệnh npm -v, nếu một phiên bản hiện ra thì hệ thống của bạn đã được cài đặt npm.

Vì **NPM** là một phần mềm cài đặt trên máy tính của bạn nên bạn có thể sử dụng nó để cài đặt các thư viện **Javascript** từ trên **Internet**. Để cài đặt một thư viện nào đó, chỉ cần mở cửa sổ **Terminal (hoặc CMD)** và thực thi lệnh giống dưới đây:

npm install package-name

VD như mình thử tải Vuejs về sử dụng sẽ dùng lệnh:

npm install vue

Khi đó muốn sử dụng Vue.js chúng ta chỉ cần sử dụng lệnh require():

var Vue = require('vue');

**Cài đặt global và cài đặt local**

**Có hai cách để cài đặt một gói bằng npm:**

* **Local:** sẽ tạo ra thư mục node\_modules nếu chưa có trong project hoặc nếu có rồi nó sẽ lấy code của gói cần cài đặt đưa vào đây, tức chỉ hiện diện trong thư mục của project hiện tại. Khi cần sử dụng bạn có thể sử dụng lệnh require().
* **Global:** sẽ lưu trữ code của gói trong các file hệ thống cố định trong máy, chỉ có thể dùng các package này thông qua các hàm CLI (Command Line Interface) ví dụ như gulp. Không thể dùng package thông qua require().

**Mặc định thì các package khi cài đặt đều sẽ là cài trên project của bạn.**

Các package thư viện đưa vào project của bạn có thể liên tục có update mới. Thực hiện npm update để thực hiện cập nhật tất cả các gói liên quan. Nếu bạn chỉ muốn cập nhật một gói cụ thể có thể sử dụng cú pháp:

npm update package\_name

**Package.json**

Để quản lý các gói cài đặt cục bộ bằng npm thì cách tốt nhất là thông qua file package.json, chính là file nằm trong thư mục gốc của project. File [JSON](https://topdev.vn/blog/json-la-gi/) này chứa các nội dung:

* Các gói thư viện lập trình mà project sử dụng.
* Cho phép xác định phiên bản chính xác của các gói thư viện lập trình được sử dụng.
* Các gói bạn xây dựng có thể chia sẻ dễ dàng với các lập trình viên khác trên toàn cầu thông qua npm.

Lệnh npm init –yes sẽ tạo ra file package.json mẫu.

npm init --yes

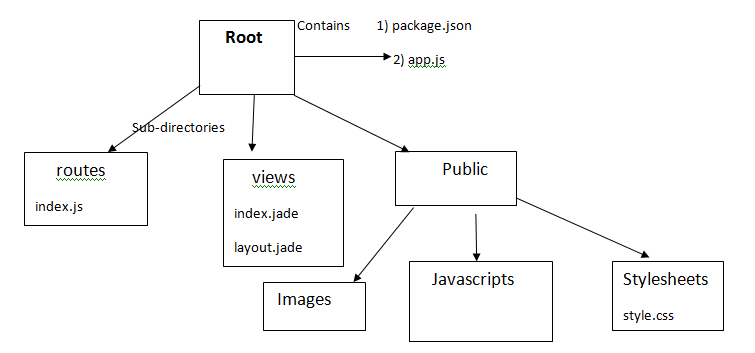
Có một số các thuộc tính trong package.json, chúng ta cùng điểm qua:

* + - **name**: tên gói thư viện
    - **version**: phiên bản gói
    - **description**: phần mô tả về gói thư viện
    - **homepage**: trang chủ của gói
    - **author**: tác giả
    - **contributors**: tên người đóng góp cho package
    - **dependencies**: danh sách các gói phụ thuộc, tự động được cài theo.
    - **repository**: loại repository và url của package, thông thường là git (Xem thêm [Git là gì](https://topdev.vn/blog/git-la-gi/)?)
    - **main**: index.js
    - **keywords**: các từ khóa

Nếu muốn thêm các entry vào thuộc tính dependencies khi cài đặt gói sử dụng thêm flag –save, còn với thuộc tính devDependencies thì sử dụng flag –save-dev.

* 1. Express là gì?

**Expressjs** là một framework được xây dựng trên nền tảng của **Nodejs**. Nó cung cấp các tính năng mạnh mẽ để phát triển web hoặc mobile. **Expressjs** hỗ trợ các method HTTP và midleware tạo ra API vô cùng mạnh mẽ và dễ sử dụng.



*Tổng hợp một số chức năng chính của Expressjs như sau:*

* Thiết lập các lớp trung gian để trả về các HTTP request.
* Define router cho phép sử dụng với các hành động khác nhau dựa trên phương thức HTTP và URL.
* Cho phép trả về các trang HTML dựa vào các tham số.

Dưới đây là một ví dụng để bạn sẽ hiểu được chức năng của mỗi phần trong 1 ứng dụng Express cơ bản.

**Bước 1: thuê Manager (require statements)**

Nếu muốn nhà hàng hoạt động hiệu quả, bạn sẽ cần 1 người giúp nhân viên làm việc hiệu quả ở mức tối đa. [Express](http://blog.topdev.vn/) sẽ là “nhân vật” Manager như thế.

Cũng giống như bất kì package NPM nào khác, bạn cần phải npm install module express, sau đó sử dụng lệnh **require**để load module.

const express = reqire(‘express’);

const app = express();

**Bước 2: đưa ra các quyết định quản lý ở nhà hàng (middleware)**

**Theo bạn, đâu là hoạt động thường diễn ra ở mọi nhà hàng? Đó sẽ là 3 hoạt động sau:**

* + - Mời khách đến ghế ngồi
    - Lấy order
    - Tính tiền

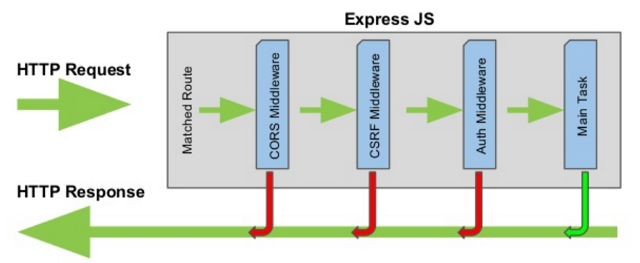
**Với mỗi hoạt động như vậy, sẽ có 1 loạt thứ phải chạy trước khi thực hiện được hoạt động đó. Ví dụ, trước khi mời khách vào ngồi, bạn cần biết:**

* + - Họ có đang mặc sơ mi và mang giày không? Nếu không thì sẽ không được ngồi.
    - Nếu họ muốn ngồi ở quán bar thì đã đủ 21 tuổi chưa? (nếu bạn đang ở Mỹ)

Tương tự với code, bạn sẽ cần phải xác nhận rằng các requests đảm bảo 1 chuẩn mực nào đó trước khi tiếp tục. Ví dụ, nếu ai đó cố gắng đăng nhập vào site thì:

* Họ có tài khoản chưa?
* Họ nhập password đúng chưa?

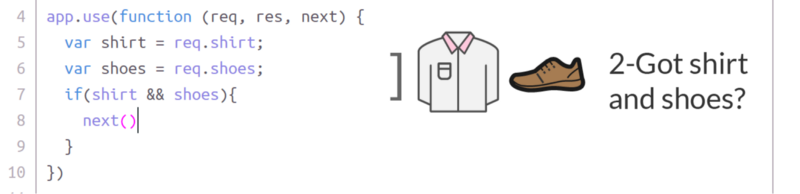
Đây là lý do concept **middleware** đã xuất hiện. Các hàm Middleware cho phép chúng ta thực hiện hành động trước request bất kì và modify nó trước khi gửi lại response.



Trong nhà hàng của bạn, cần hàng loạt các nguyên tắc để đưa ra quyết định xem liệu bạn có nên mời khách vào ghế hay không. Nguyên tắc ở đây là: họ có mang áo sơ mi và giày không?

Đầu tiên, hãy bắt đầu với [app.use()](http://expressjs.com/en/api.html#app.use). Đây là những rules đơn giản cần phải áp dụng để thực hiện các bước tiếp theo. Những rule này không phải GET, POST, PUT hay DELETE.

Ở line 4, có 1 hàm ẩn danh với các thông số req, res và next. Xét trong code block này, bạn chỉ đang kiểm tra request (req) để xem liệu có áo sơ mi & giày hay không.



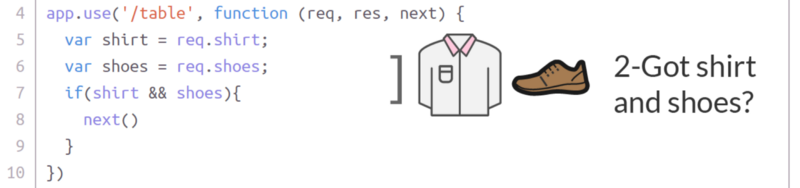
Cuối cùng, bạn cũng cần phải sử dụng hàm next() vì ở đây bạn chỉ đang xác nhận trang phục. Các bước tiếp theo, bạn sẽ cho phép khách được thực sự vào bàn ăn.

Ở line 5 & 6, bạn sẽ kiểm tra liệu các khách hàng đó có mặc áo và mang giày không.

Và ở line 7–9, bạn chỉ tiếp tục nếu các vị khách có cả 2 là mặc áo & mang giày.

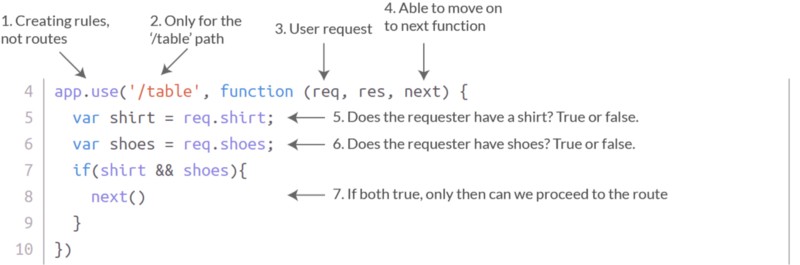
Code block ở trên bị thiếu 1 thứ quan trọng là **path**. Đây là string riêng đi chung với request. Và bởi vì thiếu path, nó sẽ chạy theo mỗi single request.

Bạn có thể tưởng tượng được không? Khi khách vào nhà hàng, gọi món, tính tiền, nhân viên sẽ phải nhìn họ trên dưới để đảm bảo là họ có mặc đồ! Nghe thật vô lý & trái với nguyên tắc kinh doanh.



Vì vậy, chúng ta đã thay đổi line 4 trong ví dụ ở trên. Bây giờ, chúng ta chỉ chạy đoạn code này khi 1 user truy vấn cùng route ‘/table’.

Chi tiết có thể xem hình bên dưới:



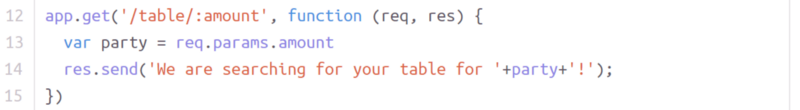
**Bước 3: thực hiện các routines (routing)**

Tiếp tục với ví dụ về việc mời vào chỗ ngồi. Cho đến lúc này, chúng ta chỉ biết được cách để xác thực liệu ai đó có nên được ngồi vào ghế hay không. Nhưng tôi thực sự không biết cách nào để đưa họ vào bàn và mời họ ngồi.

Chính lúc này, **routes** đã xuất hiện. Routes cho phép chúng ta soạn thảo (script) các hành động dựa trên **path**. Các options sẽ là GET, POST, PUT & DELETE, nhưng hiện tại chúng ta sẽ tập trung vào GET và POST.

Trong bối cảnh nhà hàng, chúng ta cần phải tạo 1 request GET để chọn 1 bàn ăn cụ thể và mời khách ngồi. GETs không modify hay thêm vào database của bạn. Chúng chỉ lấy thông tin dựa trên các thông số cụ thể.

Trong trường hợp này, bạn sẽ cần tạo 1 quy trình để sắp xếp chỗ ngồi cho 1 buổi tiệc với 2 khách. Số 2 xuất phát từ **request**của khách hàng.



Đúng vậy, đây chỉ là hành động gửi đi 1 tin nhắn vào lúc cuối. Nó vẫn chưa thực sự tìm được 1 bàn cụ thể để mời khách ngồi. Tôi sẽ cần phải tìm 1 array cho 1 bàn ăn còn trống và xử lý tình huống bên lề – tình huống nằm ngoài tầm giải quyết của tutorial này.

Ở line 12, chúng ta define quy trình này để tìm kiếm bàn ăn khi 1 vị khách **requests**cùng **route** ‘/table’. Cũng giống như ví dụ về middleware ở trên, chúng ta có sẵn các thông số request & response, có 1 **parameter**mà trong ví dụ này là hai.

Trên thực tế, mọi thứ đằng sau function declaration ở line 12 về mặt kỹ thuật là **middleware** vì nó modify user request. Bạn sẽ thấy điều đó trong biểu đồ ở dưới.

Ở line 13, chúng ta tiếp cận 1 số lượng người trong bữa tiệc từ các **parameters**của request object. Nó không được declare ở bất kì đâu vì request đến từ user và chúng ta không có dòng code frontend nào. Vì vậy nếu đây là 1 ứng dụng thực sự thì request sẽ trông như thế này:

req = {

params: {

amount: 2;

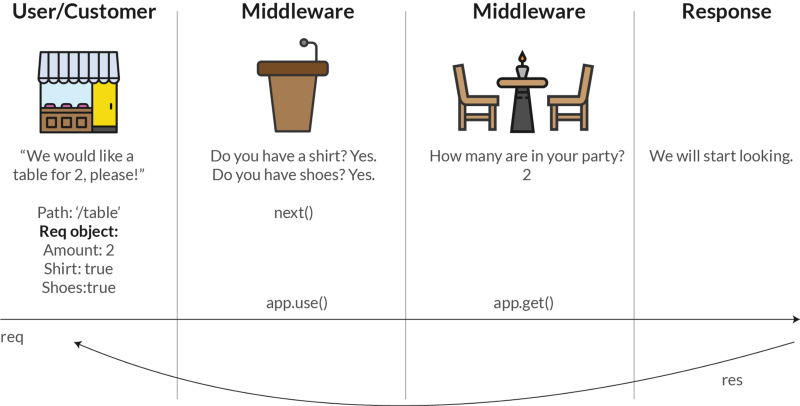
}

}

ở line 13, biến party tiếp cận **property** của **object** params trong **request**ở trên.

Cuối cùng ở line 14, chúng ta gửi 1 **response** trở lại với khách hàng: chúng ta đang tìm kiếm bàn ăn có kích cỡ vừa phải.

Đây là biểu đồ tóm gọn lại mọi thứ:



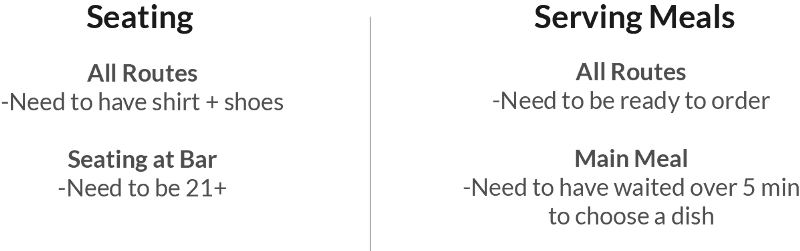
**Bước 3.5: khai thác tối đa nhà hàng (router)**

Bây giờ, bạn có thể truy dấu vết của toàn bộ đường đi từ request đến response. Nhưng khi app của bạn tăng về kích cỡ, bạn sẽ không muốn code những nguyên tắc này độc lập cho mỗi route. Bạn sẽ nhận ra 1 vài routes chia sẻ cùng các rules, vì vậy cần phải tìm cách để ứng dụng 1 set các rules vào nhiều routes.

Nói riêng việc mời khách vào chỗ ngồi, bạn có thể mời khách vào quầy bar hoặc vào bàn ăn. Chúng đều có những nguyên tắc chung như áo sơ mi + giày, nhưng muốn ngồi ở quầy bar thì các thành viên của bữa tiệc phải đủ 21 tuổi.

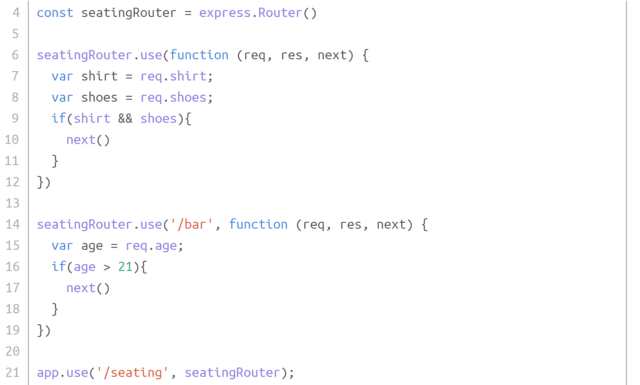
Và, xét trên câu chuyện phục vụ khách hàng, bạn sẽ cần phải dùng đến 1 quy trình khác đi 1 chút như phục vụ món khai vị, món chính và yến tiệc. Tuy nhiên, 3 routes đó đều có nhiều điểm giống nhau.

Lúc này, **router**sẽ giúp bạn bằng cách nhóm các routes để từ đây bạn có thể tạo các nguyên tắc chung.



Chúng ta cần phải tạo middleware để giải quyết hết mỗi trường hợp này. Bạn sẽ chỉ cần xử lý các trường hợp mời chỗ ngồi vì nó sẽ overwrite đoạn code ở trên.

Trích dẫn toàn bộ code ở trên:



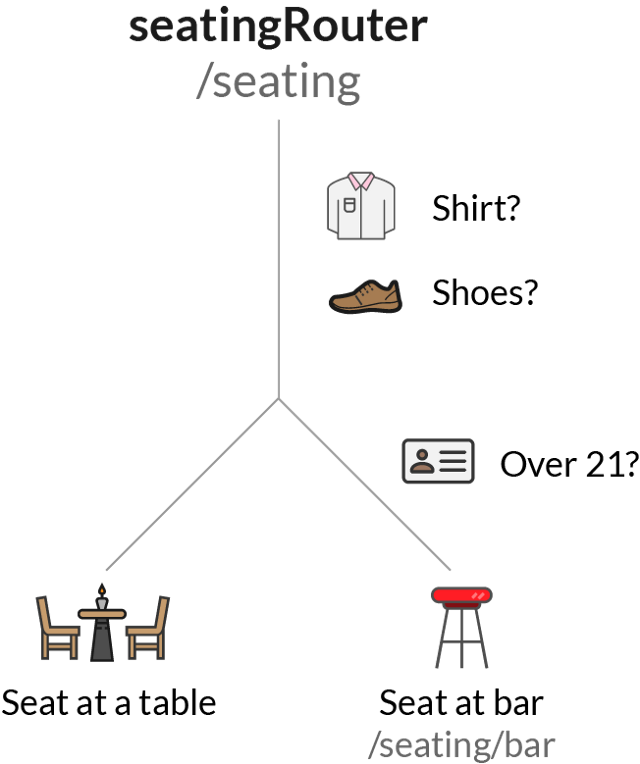
Tôi sẽ giải thích riêng biệt mỗi phần.

Ở line 4, chúng ta declare router.

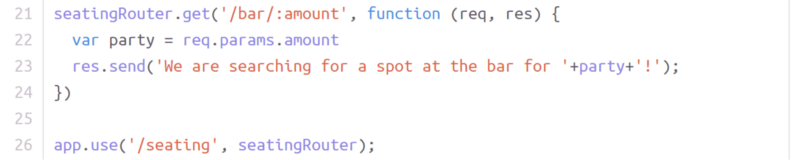
Ở lines 6 & 14, chúng ta hiện có seatingRouter.use() thay thế cho app.use() để chỉ ra rằng **middleware** này chỉ liên quan đến các routes seatingRouter.

Cuối cùng, ở line 21, chúng ta thêm middleware, cho thấy rằng mỗi seatingRouter route bắt đầu với ‘/seating’. Vì vậy, nếu ai đó yêu cầu 1 ghế ngồi ở quầy bar, toàn bộ path sẽ là ‘/seating/bar.’ Nghe có vẻ lệch quy trình nhưng bạn có thể đang mong là path sẽ được define khi bạn tạo router ở line 4.

Bình thường thôi, hãy xem form biểu đồ sau:



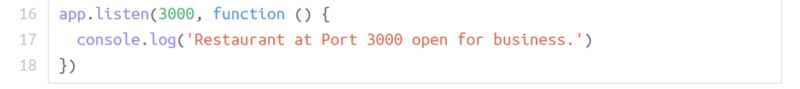
Và, khi thêm 1 route GET, nó sẽ ở trên statement cuối – nơi bạn chỉ định các routes đến router.



**Bước 4: bắt đầu kinh doanh (ports)**

Bạn đã thuê manager, xác định những gì phải làm trước khi chấp nhận customer requests, và xác định những gì phải làm với các customer request cụ thể khi chúng xuất hiện. Bây giờ, chúng ta chỉ cần xác định chính xác địa chỉ cho địa điểm diễn ra tất cả các bước này.

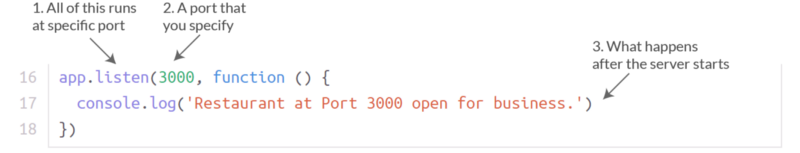
Server của bạn có các **ports,**tương tự như địa chỉ của chính nhà hàng. Vì server có thể xử lý nhiều loại nhà hàng (hoặc các script server-side) cùng 1 lúc, nên bạn cần phải cho server biết nơi mà mỗi script nên chạy.



Trong ví dụ ở trên, port là 3000 và được định vị trong máy tính của bạn. Vì vậy, nếu bạn gõ:

<https://localhost:3000/>

vào browser của mình, và bạn đang chạy app Node, server sẽ biết cách chạy script cụ thể. Trong trường hợp này, ngay khi nhập URL, bạn sẽ nhập message vào console và có thể sử dụng **các routes**bất kì. Nếu bản thân nhà hàng là toàn bộ ứng dụng của bạn thì hiện tại nó đã bắt đầu kinh doanh ở địa chỉ 3000.



* 1. ReactJS là gì?

React.JS là một thư viện Javascript dùng để xây dựng giao diện người dùng. Với cá nhân tôi cũng như nhận xét chung của cộng đồng về ReactJS thì nó nhanh, dễ học và vui.

Tiếp theo chúng ta sẽ bắt đầu đến với những khái niệm.

* + 1. Component

React được xây dựng xung quanh các component, chứ không dùng template như các framework khác. Bạn có thể tạo ra một component bằng các gọi phương thức createClass của đối tượng React, điểm bắt đầu khi tiếp cận với thư viện này.

Ví dụ.

var Button = React.createClass({

render: function(){

return (

<input type="submit" />

);

}

});

Phương thức createClass nhận vào một tham số, là đối tượng mô tả đặc tính của component. Đối tượng này bao gồm tất cả các phương thức để hình thành nên component. Phương thức quan trọng nhất là render, phương thức này được trigger khi component đã sẵn sàng để được render lên trên page.

Trong hàm đó, bạn ẽ trả về một mô tả cho việc bạn muốn React render cái gì lên trên page. Như trong ví dụ ở trên, đơn giản tôi muốn render một button.

Chú ý: Hàm render chính là mô tả cụ thể của UI tại bất cứ thời điểm nào. Vì thế nếu dữ liệu thay đổi, React sẽ take care việc update UI với dữ liệu tương ứng. Các bạn có thể hiểu đơn giản là, khi dữ liệu thay đổi, React sẽ tự động gọi hàm render để update lại UI.

### **2.4.2 JSX — Javascript Syntax Extension**

Đây đơn giản là một syntax extension của Javascript. Với nó bạn có thể viết Javascript với những tag giống như XML (XML-like). Về bản chất, các tag thực sự là những lời gọi hàm, sẽ được chuyển đổi trong React code và end up dưới dạng HTML và Javascript trong cây DOM.

Nhưng với những gì bạn biết ở hiện tại, chỉ cần hiểu đơn giản nó giống như là HTML/XML với một số khả năng khác.

* + 1. Multiple components

Nếu bạn muốn lồng nhiều component vào nhau, bạn sẽ làm điều này trong lệnh return của phương thức render.

Ví dụ.

var Form = React.createClass({

render: function(){

return (

<div>

<input type="submit" onClick={this.props.onUserClick} />

<h3>You have pressed the button {this.props.counter} times!</h3>

</div>

);

}

});

var App = React.createClass({

getInitialState: function(){

return {

counter: 0

}

},

onUserClick: function(){

var newCount = this.state.counter += 1;

this.setState({

counter: newCount

});

},

render: function(){

return (

<div>

<h1> Welcome to the counter app!</h1>

<Form counter={this.state.counter} onUserClick={this.onUserClick} />

</div>

);

}

});

React.render(<App />, document.getElementById("app"));

Phía trên, tôi đang lồng Form component vào trong App component. Đây là một dạng quan hệ cha con (parent-child) mà bạn có thể dễ dàng nhận thấy trong HTML

Phương thức React.render() như các bạn thấy ở trên nhằm mục đích kickstart việc render, và render thừ root component, trong trường hợp trên là App vào trong DOM với container cụ thể là element có id là app

* + 1. Props & State là gì?

Có hai kiểu của data trong React đó là props và state. Sự khác biệt giữa hai kiểu thì hơi khó khăn để hiểu ngay từ ban đầu, ít nhất là về mặt khái niêm. Nhưng một khi bạn bắt đầu code, bạn sẽ nhanh chóng tách biệt được hai loại.

Điểm mấu chốt của sự khác nhau là state thì private và chỉ có thể được thay đổi bên trong bản thân component. Props thì mang tính external, và không bị kiểm soát bởi bản thân component. Nó được truyền từ component cao hơn theo phân cấp, hay có thể hiểu đơn giản là truyền từ component cha xuống component con, cái mà điều khiển dữ liệu trước khi truyền xuống.

Vì thế trong khi một component không thể thay đổi props của nó một cách trức tiếp (điều này có thể làm một cách gián tiếp nhưng hãy để nó vào những phần sau), thì nó có thể tự thay đổi state của bản thân.

**Props**

Nào cùng cài đặt app của chúng ta đã làm trong bài trước với một ít dữ liệu, sử dụng props nhé. Đầu tiên chúng ta cần lấy dữ liệu từ một nơi nào đó. Đó có thể là Ajax call để lấy một số dữ liệu từ API, tuy nhiên chúng ta sẽ hard code nó như một variable.

var text = "Click the button";

Cách để đưa props vào một component nhìn rất giống cách mà chúng ta khai báo attribute cho một HTML element.

<App text={text} />

Lý do chúng ta sử dụng cặp ngoặc nhọn là vì chúng ta cần nói cho JSX biết rằng đó là một Javascript expression.

Một khi App component được cài đặt như thế này, nó có thể truy xuất vào biến text mà ta đã khai báo ở trên thông qua lời gọi this.props.text. Tuy nhiên, nó không thể trực tiếp thay đổi dữ liệu. Từ góc nhìn của component, props của nó là bất biến (immutable). Nó chỉ là thông tin được cài đặt cho component.

Đây là ví dụ.

var text = "Click the button";

var Form = React.createClass({

render: function(){

return (

<div>

<h3>{this.props.text}</h3>

<input type="submit" />

</div>

);

}

});

var App = React.createClass({

render: function(){

return (

<div>

<h1> Welcome to my app!</h1>

<Form text={this.props.text}/>

</div>

);

}

});

React.render(<App text={text}/>, document.getElementById("app"));

Như các bạn thấy, props được truyền vào trong App component trong phương thức React.render(). Sau đó App component có thể truy xuất biến text thông qua lời gọi this.props.text. Nó cũng có thể truyền dữ liệu xuống component con của nó như chúng ta thấy cách mà Form component được App component cài đặt props trong ví dụ.

Khi dữ liệu đến được Form component, chúng ta thấy đây là điểm kết thúc, dữ liệu sẽ được render ra thẻ h3 như trên.

Đây là cách mà dữ liệu được luân chuyển trong React thông qua props.

**State**

Một cách khác để storing dữ liệu trong React là state. Không giống như props, bất biến dưỡi góc nhìn của component thì state có thể thay đổi (mutable).

Vì thế nếu bạn muốn dữ liệu trong ứng dụng thay đổi, ví dụ như dựa trên tương tác người dùng, thì dữ liệu phải được lưu trữ trong component state.

State là private và được quản lý bởi chỉ duy nhất một component, nó không thể truyền xuống cho component con. Nếu bạn muốn truyền xuống cho component con thì bạn phải truyền nó như là một props.

Để cài đặt state, đơn giản chúng ta cài đặt hàm getInitialState() vào component, và trả về bất cứ gì bạn muốn cài đặt trong state của component đó.

Để thay đổi state, đơn giản ta gọi hàm this.setState(), và truyền vào state mới như là một tham số.

Ví dụ.

var App = React.createClass({

getInitialState: function(){

return {

active: true

}

},

handleClick: function(){

this.setState({

active: !this.state.active

});

},

render: function(){

var buttonSwitch = this.state.active ? "On" : "Off";

return (

<div>

<p>Click the button!</p>

<input type="submit" onClick={this.handleClick} />

<p>{buttonSwitch}</p>

</div>

);

}

});

React.render(<App />, document.getElementById("app"));

Đoạn code trên cũng cho bạn làm quen với hệ thống event trong React, rất đơn giản. Chúng ta hook một event listener vào trong button, ở trên là onClick. Khi nó được trigger, chúng ta gọi hàm handleClick, cái mà đã được cài đặt trước đó, và luôn sẵn sàng được gọi thông qua từ khóa this.

Trong hàm handleClick, chúng ta gọi this.setState(), cái mà sẽ thay đổi trạng thái của component.

Chú ý: React event được wrap để chạy trên tất cả các browser, có nghĩa là React giúp bạn đảm bảo event của bạn chạy được trên tất cả các trình duyện.

Chúng ta nên giữ state ở đâu?

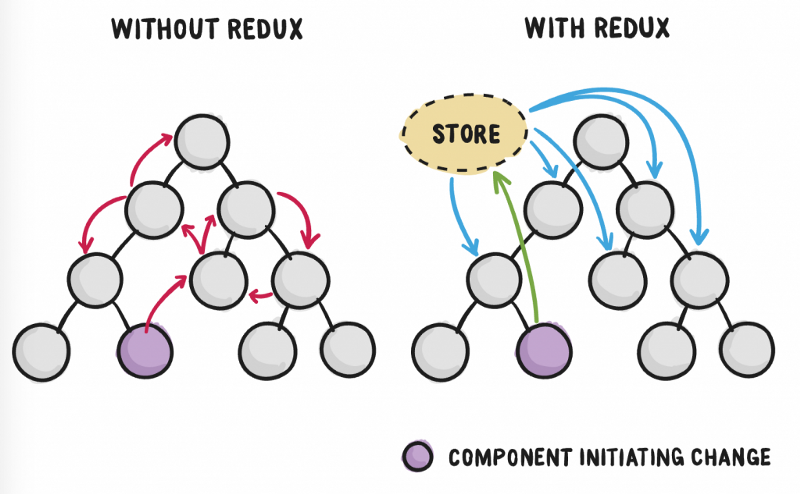
Bạn nên cố gắng giữ số lượng các stateful component ít nhất có thể, và thậm chí giữ tối thiểu lượng dữ liệu trong state. Nếu component cấp dưới cần truy xuất dữ liệu từ state, thì hãy truyền nó thông qua props.

Lưu ý: Stateful component thì luôn luôn là higher level, trong khi Stateless component thường là lower level trong hệ thống phân cấp.

Để hình dung việc state được giữ ở đâu, bạn hãy hỏi bản thân một vài câu hỏi, những câu hỏi này được lấy từ React docs:

* Xác định mỗi component mà render thông tin gì đó dựa trên state.
* Tìm một component mà nó chủ sở hữu chung của các component khác (một component nằm bên trên tất cả các component khác trong hệ thống phân cấp thì cần có state)
* Hoặc là những component là chủ sở hữu chung hoặc là những component nằm trên hệ thống phân cấp sẽ nên giữ state.
* Nếu bạn không thể tìm ra component nào phù hợp, hãy tạo một component mới đơn giản giữ nhiệm vụ lưu trữ state và đặt nó đâu đó nằm bên trên các component là chủ sở hữu chung trong hệ thống phân cấp.
  1. Redux là gì?

**Redux** là một predictable state management tool cho các ứng dụng Javascript. Nó giúp bạn viết các ứng dụng hoạt động một cách nhất quán, chạy trong các môi trường khác nhau (client, server, and native) và dễ dàng để test. [**Redux**](https://topdev.vn/blog/?s=redux) ra đời lấy cảm hứng từ tư tưởng của ngôn ngữ **Elm** và kiến trúc **Flux**của Facebook. Do vậy Redux thường dùng kết hợp với React.



* + 1. Tại sao phải có redux

Do yêu cầu cho các ứng dụng single-page sử dụng Javascript ngày càng trở lên phức tạp thì code của chúng ta phải quản lý nhiều state hơn.

State có thể bao gồm là data trả về từ phía Server và được cached lại hay như dữ liệu được tạo ra và thao tác ở phía client mà chưa được đẩy lên phía server. UI state cũng trở lên phức tạp vì chúng ta cần quản lý việc active Routes, selected tabs, spinners, điều khiển phân trang …vv.

Với Redux, state của ứng dụng được giữ trong một nơi gọi là store và mỗi componentđều có thể access bất kỳ state nào mà chúng muốn từ chúng store này.

Hầu hết các lib như React, Angular, etc được built theo một cách sao cho các components đến việc quản lý nội bộ các state của chúng mà không cần bất kỳ một thư viện or tool nào từ bên ngoài.

Nó sẽ hoạt động tốt với các ứng dụng có ít components nhưng khi ứng dụng trở lên lớn hơn thì việc quản lý states được chia sẻ qua các components sẽ biến thành các công việc lặt nhặt.

Trong một app nơi data được chia sẻ thông qua các components, rất dễ nhầm lẫn để chúng ta có thể thực sự biết nơi mà một state đang live. Một sự lý tưởng là data trong một component nên live trong chỉ một component. Vì vậy việc share data thông qua các components anh em sẽ trở nên khó khăn hơn.

Ví dụ, trong react để share data thông qua các components anh em, một state phải live trong component cha. Một method để update chính state này sẽ được cung cấp bởi chính component cha này và pass như props đến các components con.

Đây là một ví dụ:

class App extends React.Component {

constructor(props) {

super(props);

this.state = { userStatus: "NOT LOGGED IN"}

this.setStatus = this.setStatus.bind(this);

}

setStatus(username, password) {

const newUsers = users;

newUsers.map(user => {

if (user.username == username && user.password === password) {

this.setState({

userStatus: "LOGGED IN"

})

}

});

}

render() {

return (

<div>

<Status status={this.state.userStatus} />

<Login handleSubmit={this.setStatus} />

</div>

);

}

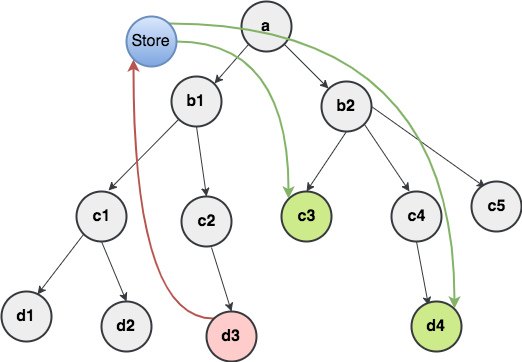
});

Giờ chúng ta hãy tưởng tượng rằng nếu một state phải được chia sẻ giữa các component cách khá xa nhau trong một tree components và state này phải được passed từ một component đến một component khác cho đến khi nó đến được nơi mà nó được gọi.

Cơ bản là, state mà chúng ta đang nói đến phải được nhấc lên một component cha gần nhất và tiếp nữa cho đến khi nó đến được cái component tổ tiên chứa tất cả các components nó cần cái state này sau đó pass cái state này xuống @@. Điều này sẽ khiến state trở nên khó hơn trong việc maintain và less predictable.

Điều này khiến cho bộ phận quản lý state trong app trở lên bừa bộn cũng như app trở lên vô cùng phức tạp. Đó là lý do tại sao chúng ta cần một state management tool như Redux.

* + 1. Hiểu cách redux làm việc



Cái cách mà Redux hoạt động là khá đơn giản. Nó có 1 store lưu trữ toàn bộ state của app. Mỗi component có thể access trực tiếp đến state được lưu trữ thay vì phải send drop down props từ component này đến component khác.

Có 3 thành phần của Redux: Actions, Store, Reducers.

**Actions**

Actions đơn giản là các events. Chúng là cách mà chúng ta send data từ app đến Redux store. Những data này có thể là từ sự tương tác của user vs app, API calls hoặc cũng có thể là từ form submission.

Actions được gửi bằng cách sử dụng store.dispatch() method, chúng phải có một type property để biểu lộ loại action để thực hiện. Chúng cũng phải có một payload chứa thông tin. Actions được tạo thông qua một action creator. Ví dụ:

const setLoginStatus = (name, password) => {

return {

type: "LOGIN",

payload: {

username: "foo",

password: "bar"

}

}

}

**Reducers**

Reducers là các function nguyên thủy chúng lấy state hiện tại của app, thực hiện một action và trả về một state mới. Những states này được lưu như những objects và chúng định rõ cách state của một ứng dụng thay đổi trong việc phản hồi một action được gửi đến store.

Đây là một ví dụ về cách mà Reducers hoạt động trong Redux:

const LoginComponent = (state = initialState, action) => {

switch (action.type) {

case "LOGIN":

return state.map(user => {

if (user.username !== action.username) {

return user;

}

if (user.password == action.password) {

return {

...user,

login\_status: "LOGGED IN"

}

}

});

default:

return state;

}

};

**Store**

Store lưu trạng thái ứng dụng và nó là duy nhất trong bất kỳ một ứng dụng Redux nào. Bạn có thể access các state được lưu, update state, và đăng ký or hủy đăng ký các listeners thông qua helper methods.

Tạo một store cho một login app:

const store = createStore(LoginComponent);

Các actions thực hiện trên một state luôn luôn trả về một state mới. Vì vậy, state này là đơn giản và dễ đoán. Bây giờ, chúng ta đã biết hơn một chúng về Redux, hãy trở lại với ví dụ Login component và xem cách cách mà Redux có thể giúp chúng ta được gì.

class App extends React.Component {

render() {

return (

<div>

<Status user={this.props.user.name}/>

<Login login={this.props.setLoginStatus}/>

</div>

)

}

}

* + 1. Nguyên lý vận hành

