# Tìm hiểu React

1. Giới thiệu React

React là thư viện javascript dùng để xây dựng giao diện người dùng.

Một số ưu điểm đáng chú ý của React:

+ Cộng đồng lớn

+ Hiệu suất cao

+ Được đánh giá là thân thiện với SEO

+ Khả năng mở rộng tốt, tính sử dụng cao

+ Phát triển nhanh chóng

Ngoài ra React hỗ trợ rất mạnh trong việc phát triển SPA, tạo sự mượt mà, nhanh chóng trong quá trình sử dụng.

1. Tìm hiểu JSX

const element = <h1>Hello, world!</h1>

JSX là sự kết hợp Javascript + XML, nó sẽ chuyển đổi cú pháp dạng giống XML về thành Javascript. Vì thế, nó cần một trình biên dịch gọi là Babel để có thể chuyển đổi code JSX sang code tương ứng của nó là React.createElement. Ngoài ra, chúng ta hoàn toàn có thể không sử dụng JSX, đoạn mã sau hoàn toàn giống với đoạn mã ở trên.

React.createElement(‘h1’, null, ‘Hello, world!’)

Khi không sử dụng JSX chúng ta buộc phải dùng tới API trong React là createElement để tạo element. React.createElement() chấp nhận 3 tham số:

* Component hoặc Tag để sử dụng tạo element
* Props của component
* Children

Qua đó, ta có thể thấy rõ được sự tiện lợi, ngắn gọn trong việc sử dụng JSX và việc nó khá giống với HTML giúp người dùng thích nghi nhanh chóng.

1. Rendering element

Không giống DOM elements trong browser, React element là một plain object vì thế chi phí tạo element trong React sẽ nhỏ hơn.

Theo mặc định, sẽ luôn có <div id="root"></div> sẵn trong file HTML, và mọi thứ bên trong root sẽ được React quản lý.

Việc rerender element sẽ được thực hiện khi react cảm thấy cần thiết, react sử dụng heuristic O(n) algorithm để xem xét việc rerender.

1. State và Props

Props là một object javascripts, props thường dùng để truyền dữ liệu từ component cha xuống component con, các component con không thể thay đổi props.

Component App sẽ truyển props tên là name có giá trị Sara.

function App() {

return (

<div>

<Welcome name="Sara" />

</div>

);

}

Component Welcome sẽ nhận props từ component App.

function Welcome(props) {

return <h1>Hello, {props.name}</h1>;

}

State là một object có thể được sử dụng để chứa dữ liệu hoặc thông tin về components. State có thể được thay đổi bất cứ khi nào mong muốn. Khác với props bạn có thể truyền props sang các components khác nhau thì state chỉ tồn tại trong phạm vi của components chứa nó, mỗi khi state thay đổi thì components đó sẽ được render lại.

constructor(props) {

super(props);

//Chỉ định một state

this.state = { website: "Freetuts.net" };

}

Cập nhật state:

this.setState({

name : 'newValue'

})

1. Lifecycle

Lifecycle là các giai đoạn mà component trải qua, quá trình này là tuần hoàn. Có 4 giai đoạn chính của component:

* initialization
* mounting
* updating
* unmounting

Initialization là giai đoạn khởi tạo các state, props, method.

class App extends React.Component {

constructor(props) {

super(props);

this.state = {

website: 'Học ReactJS cùng Freetuts.net'

};

}

}

Mounting được thực hiện sau Initialization, chuyển DOM ảo thành DOM và hiển thị trên trình duyệt, lưu ý muounting chỉ xảy ra khi component render lần đầu.

componentWillMount()

Được khởi chạy khi một component chuẩn bị được mount (tức là trước khi thực hiện render), sau khi thực hiện xong componentWillMount() thì component mới có thể được mount.

componentDidMount()

Được gọi khi component đã được mount (render thành công ), quá trình này xảy ra sau khi componentWillMount() thực hiện xong. Đặc biệt, side effect chỉ nên được tực hiện trong componentDidMount.

Updating là giai đoạn component phải thực hiện sau initialization, mount, … Trong giai đoạn này, dữ liệu của các phần (props & state) sẽ được cập nhật để đáp ứng với các sự kiện của người dùng.

ShouldComponentUpdate() dùng để quyết định nên update hay không

ComponentWillUpdate() xảy ra trước khi tiến hành rerender.

ComponentDidUpdate() xảy ra sau khi rerender.

Unmount là giai đoạn component bị xóa khỏi DOM.

componentWillUnmount() xảy ra trước khi unmount.

1. Conditional rendering

Conditional redering giúp react trở nên linh hoạt, tính sử dụng lại tăng cao.

if (isLoggedIn) {

return <UserGreeting />;

}

return <GuestGreeting />;

Theo đoạn mã ở trên, react sẽ render component tùy thuộc và biến isLoggedIn. Nếu biến isLoggedIn là mang giá trị true thì sẽ render component UserGreeting, ngược lại sẽ render component GuestGreeting.

Ngoài ra, ta cũng có thể sử dụng toán tử ba ngôi:

return (

<div>

{isLoggedIn

? < UserGreeting />

: < GuestGreeting />

}

</div>

);

1. Thinking in React

Để có thể sử dụng React hiệu quả, việc phân chia component là hết sức quan trọng. Thay vì code tất cả trong 1 file chúng ta nên chia nhỏ trang web ra thành nhiều phần nhỏ và cố gắng tái sử dụng các thành phần.

# Tìm hiểu ASP

1. HTTP Request and HTTP Response

HTTP viết tắt của Hyper text transform protocol, là một giao thức giao tiếp để sever và client trao đổi.

HTTP request là một message được gửi đi nhầm yêu cầu truy cập file, dữ liệu, …

Mỗi HTTP request gồm 3 thành phần: Request Line, Request Header và Request Body.

Request line:

* Chỉ định phương thức (GET, POST, PUT, …)
* Chứa URL của Request được sử dụng để tìm tài nguyên trên máy chủ
* Chỉ định phiên bản giao thức HTTP

Request header:

* Content-type: định dạng dữ liệu
* Accep-language: cho máy chủ biết ngôn ngữ ưu tiên của máy khách

Request Body:

* Gồm các dữ liệu đính kèm

HTTP response là phản hồi request từ phía máy chủ. Có cấu trúc tương tự với HTTP request: Status Line, Response Header và Response Body.

Status line:

* Status code: cung cấp mã trạng thái (200, 500, …)
* HTTP Version: Phiên bản giao thức HTTP
* Reason Phrase (còn gọi là Status Text): Mô tả trạng thái

Response Header và Response Body tương tự như Request Header và Request Body.

1. Cơ chế Routing

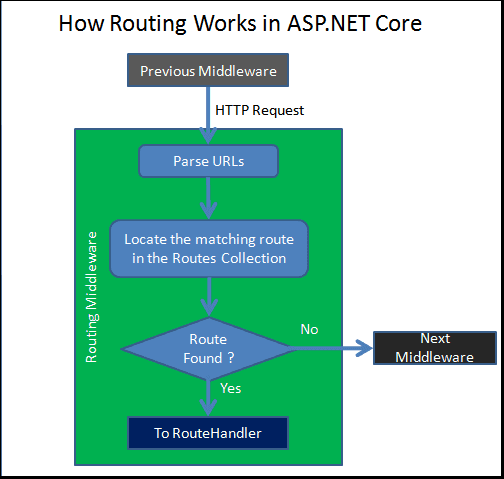
Routing là một quá trình khi ASP.NET Core xem xét các URL request gửi đến và "chỉ đường" cho nó đến Controller Actions. Nó cũng được sử dụng để tạo ra URL đầu ra. Quá trình này được đảm nhiệm bởi Routing Middleware. Routing Middleware có sẵn trong thư viện Microsoft.AspNetCore.Routing.

Routing có 2 trách nhiệm chính:

* Nó map request đến vào Controller Action.
* Tạo ra URL đầu ra tương ứng với Controller action.

Khi request đến thì Routing Middleware sẽ làm những việc sau đây:

* Phân tích URL
* Tìm kiếm xem có cái Route nào match trong RouteCollection
* Nếu Route tìm thấy thì đẩy nó sang RouteHandler
* Nếu không tìm thấy Route nào thì bỏ qua và gọi middleware tiếp theo



1. Swagger

Một cách đơn giản để thêm Swagger vào ứng dụng ASP.NET Core: cần cài đặt Swashbuckle.AspNetCore gói nuget bao gồm - trình tạo Swagger, phần mềm trung gian để hiển thị Swagger được tạo dưới dạng điểm cuối JSON và phần mềm trung gian để hiển thị swagger-ui được cung cấp bởi các điểm cuối đó. Khi gói được cài đặt, mã tối thiểu cần thiết để định cấu hình swagger Startup.cs sẽ là:

public void ConfigureServices(IServiceCollection services)

{

services.AddControllers();

services.AddSwaggerGen(c =>

{

c.SwaggerDoc("v1", new OpenApiInfo { Title = "My API", Version = "v1" });

});

}

Tạo các phương thức mở rộng cho từng dịch vụ được thêm vào IserviceCollection và IApplicationBuildertrong các tệp riêng biệt. Điều này sẽ giúp mã có khả năng tái sử dụng và cũng giúp mã có thể quản lý được.

1. Data Transfer Object (DTO)

Đối tượng truyền dữ liệu (thường được gọi là DTO) thường là một thể hiện của lớp POCO (đối tượng CLR cũ thuần túy) được sử dụng như một vùng chứa để đóng gói dữ liệu và chuyển nó từ lớp này sang lớp khác của ứng dụng. DTO được sử dụng trong lớp dịch vụ để trả dữ liệu trở lại lớp trình bày. Ưu điểm lớn nhất của việc sử dụng DTO là tách khách hàng khỏi cấu trúc dữ liệu nội bộ.