
Lucifer @phannh_58 Follow
433 5 21

Published Apr 25th, 2018 7:46 PM
757 0 7

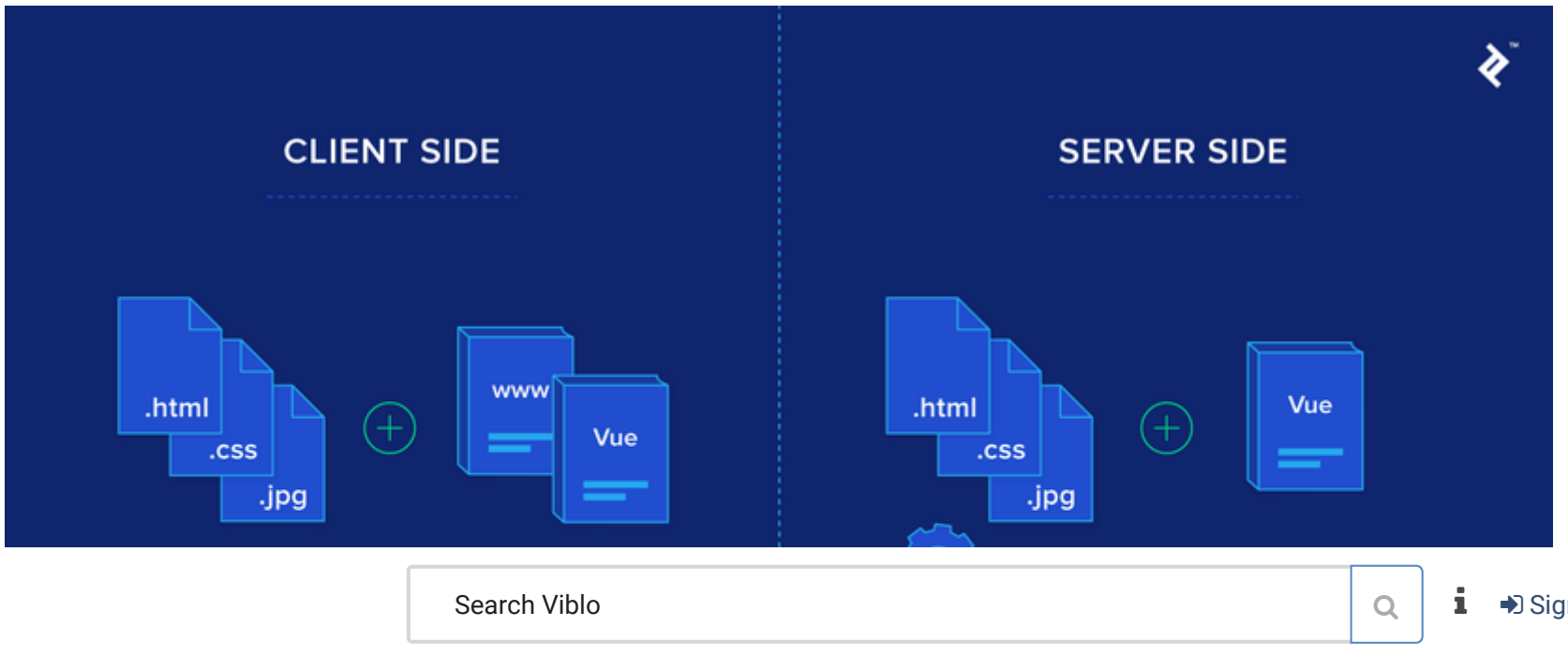
Server-side Nuxt.js for Vue.js Apps

Server nuxt.js

Thư viện/Framework Javascript giống như Vue có thể mang lại trải nghiệm tuyệt vời cho người dùng khi duyệt trang web của bạn. Tất cả các thay đổi nội dung trang đều được thực hiện một cách tự động mà không phải gửi yêu cầu tới server mỗi lần.

Tuy nhiên, có một vấn đề với cách tiếp cận này. Khi lần đầu tiên trang web của bạn được tải, trình duyệt của bạn không nhận được trang đầy đủ để hiển thị. Thay vào đó, nó tải một loạt các phần để xây dựng trang (bao gồm HTML, CSS và các files) và hướng dẫn cách đặt tất cả chúng lại với nhau (thư viện/Framework Javascript), sẽ phải mất một khoảng thời gian hoàn thiện việc tải nạp đầy đủ tài nguyên trước khi trình duyệt của bạn thực sự có gì đó để hiển thị.

Giải pháp: Yêu cầu có một framework/thư viện phía server để có thể xây dựng trang sẵn sàng cho việc hiển thị. Sau đó, gửi trạng thái hoàn chỉnh này đến trình duyệt cùng với khả năng thực hiện các thay đổi khác mà vẫn có nội dung trang động (framework/thư viện).



Giới thiệu về Nuxt.js

Nuxt.js dựa trên việc thực thi SSR cho thư viện React được gọi là Next. Sau khi nhìn thấy những ưu điểm của thiết kế này, một triển khai tương tự được thiết kế cho Vue gọi là Nuxt. Những ai đã quen với sự kết hợp React + Next sẽ phát hiện ra một loạt các điểm tương đồng trong thiết kế và cách bố trí ứng dụng. Tuy nhiên, Nuxt cung cấp các tính năng Vue cụ thể để tạo ra một giải pháp SSR mạnh mẽ và linh hoạt hơn.

Hiện tại thì Nuxt đã được cập nhật lên phiên bản 1.0 vào tháng 1/2018 và là một phần được cộng đồng sử dụng nhiều và được hỗ trợ tốt. Một trong những điều tuyệt vời mà Nuxt mang lại là xây dựng hàng loạt dự án không khác so với xây dựng bất kỳ dự án Vue nào khác. Trong thực tế, nó cung cấp một loạt các tính năng cho phép bạn tạo các codebase trong một khoảng thời gian ngắn.

Một điều quan trọng cần lưu ý là Nuxt không phải được sử dụng cho SSR. Nó được xem như một framework cho việc tạo các ứng dụng Vue.js phổ biến và chỉ cần một lệnh (*nuxt generate*) để tạo các ứng dụng Vue sử dụng cùng một codebase. Vì vậy, bạn sẽ không cần lo lắng về việc phải đi sâu tìm hiểu SSR. Bạn vẫn có thể tạo trang web tĩnh

Xây dựng ứng dụng với Nuxt

Để bắt đầu, hãy sử dụng trình tạo dự án Vue được gọi là *vue-cli* để nhanh chóng tạo project:

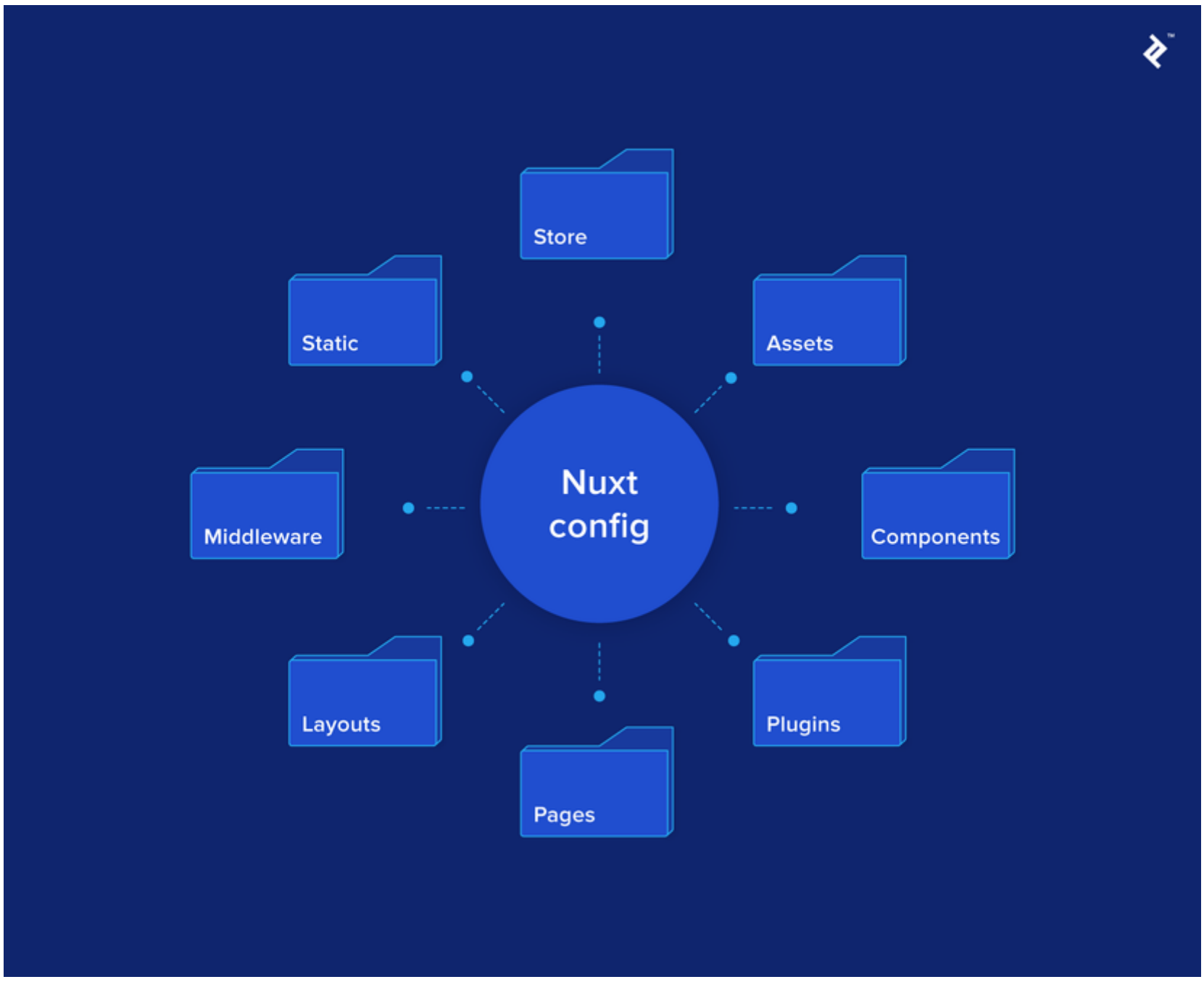
```
# install vue-cli globally
npm install -g vue-cli

# create a project using a nuxt template
vue init nuxt-community/starter-template my-nuxt-project
```

Tiếp theo, chúng ta sẽ phải cài đặt các package liên quan và bắt đầu chạy server nhé:

```
cd my-nuxt-project
npm install
npm run dev
```

Hãy mở trình duyệt của bạn tới http://localhost:3000/ và xem thành quả ban đầu nhé. Không khác nhiều so với việc tạo dự án Vue Webpack. Hãy cùng xem cấu trúc thư mục của ứng dụng xây dựng bởi Nuxt:



Nhìn vào *package.json* có thể thấy rằng chỉ có một sự phụ thuộc, chính là Nuxt. Điều này là do mỗi phiên bản của Nuxt được thiết kế để làm việc với các phiên bản vụ thể Vue, Vue-router, và Vue đã gói chúng lại với nhau cho bạn rồi đó.

Ngoài ra, còn có tập tin *nuxt.config.js*. Nó cho phép bạn tùy chỉnh một loạt các tính năng mà Nuxt cung cấp. Theo mặc định, nó đặt các thẻ tiêu đề, tải các thanh màu, và các quy tắc ESLint cho bạn. Nếu bạn muốn xem danh sách các cấu hình mà bạn có thể thêm cho ứng dụng của bạn, hãy xem tại đây (document) nhé.

Nếu bạn duyệt qua các thư mục được tạo ra, tất cả chúng đều có một file Readme kèm theo. Nó là một bản tóm tắt ngắn gọn về những gì nằm trong thư mục đó và thường là một liên kết đến các tài liệu.

Đó là một tiện ích của Nuxt: một cấu trúc mặc định cho ứng dụng của bạn. Bất kỳ nhà phát triển front-end nào cũng sẽ cấu trúc một ứng dụng tương tự như vậy, nhưng sẽ có nhiều ý tưởng khác nhau về cấu trúc và khi làm việc nhóm trong một thời gian chắc chắn sẽ phải thảo luận hoặc lựa chọn cấu trúc này.

Pages

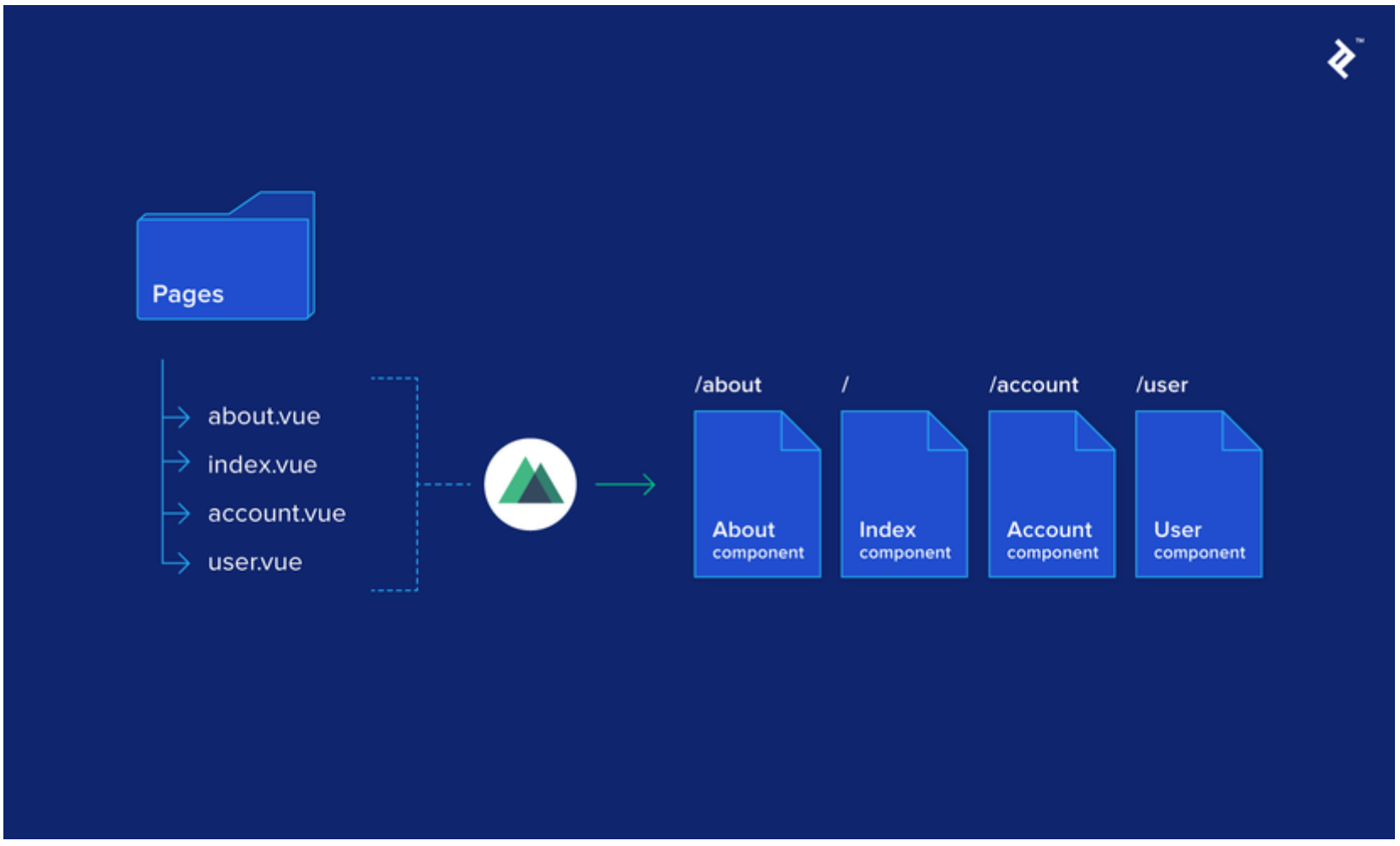
Đây là một thư mục được yêu cầu phải có. Bất kỳ thành phần Vue nào trong thư mục này sẽ tự động được thêm vào vue-router dựa trên tên tệp và cấu trúc thư mục. Điều này cực kỳ tiện lợi. Thông thường, sẽ có một thư mục *Pages* riêng biệt và phải đăng ký thủ công từng thành phần trong một tệp bộ định tuyến khác. Tệp bộ định tuyến này có thể trở nên phức tạp khi dự án lớn hơn và có thể cần phân tách để duy trì khả năng maintain sau này. Đừng lo lắng, Nuxt sẽ xử lý tất cả các phần này cho bạn.

Để thấy rõ điều này, chúng ta hãy tạo một thành phần Vue gọi là *about.vue* bên trong thư mục *pages* với nội dung đơn giản như sau:

```
<template>
 <h1>About Page</h1>
</template>
```

Khi bạn lưu lại, Nuxt sẽ tự tạo ra routes cho bạn. Nếu bạn điều hướng đến */about*, bạn sẽ thấy được thành phần mà bạn vừa thiết kế *about.vue*.

Trong thư mục *pages* tệp tin đặc biệt tên là *index.vue*, sẽ tạo một routes gốc cho thư mục. Khi dự án được tạo, đã có thành phần *index.vue* trong thư mục *pages* tương ứng với trang chủ ứng dụng.



Thư mục con trong thư mục *Pages* sẽ giúp cấu trúc các routes của bạn. Vì vậy, nếu chúng tôi muốn một trang xem thông tin sản phẩm, chúng ta có thể cấu trúc thư mục *Pages* theo mô hình sub-folder như sau:

```
/pages
--| /products
----| index.vue
----| view.vue
```

Với điều hướng */products/view*, thành phần *view.vue* sẽ được sử dụng. */products*, thành phần *index.vue* sẽ được sử dụng.

Bạn có thể hỏi tại thay vì làm như trên, tại sao không tạo thành phần products.vue trong thư mục *pages* giống với trang */about* ở trên. Bạn có thể nghĩ kết quả nó sẽ giống nhau, nhưng không, có sự khác biệt giữa hai cấu trúc. Hãy xem sự khác biệt này nhé.

Giả sử bây giờ chúng ta muốn có một trang giới thiệu cho từng nhân viên. Ví dụ, tôi sẽ tạo 1 trang giới thiệu về thông tin của tôi. Và nó được đặt tại */about/phannh*. Ban đầu, chúng ta có thể thử cấu trúc *pages* như sau:

```
/pages
--| about.vue
--| /about
----| phannh.vue
```

Khi bạn cố gắng truy cập */about/phannh*, chúng ta mong muốn nó sẽ lấy thành phần *about.vue*, giống như */about*. Điều gì sẽ diễn ra tiếp theo?

Thật thú vị, những gì Nuxt đang làm ở đây là tạo ra một routes lồng nhau. Cấu trúc này gợi ý rằng nếu bạn muốn có một định tuyến lâu dài như */about* và nó sẽ ứng với phần view riêng của nó. Trong vue-router, điều này sẽ được biểu thị bằng cách chỉ định thành phần *<router-view />* bên trong thành phần *about.vue*. Đây là khái niệm tương tự trong Vue, ngoại trừ *<router-view />*, chúng ta chỉ cần sử dụng *<nuxt />*. Vì vậy, chúng ta hãy cập nhật thành phần *about.vue* để cho phép các routes lồng nhau:

```
<template>
 <div>
   <h1>About Page</h1>
   <nuxt />
 </div>
</template>
```

Bây giờ, khi chúng ta điều hướng */about* , chúng ta có được thành phần *about.vue* mà chúng ta đã có trước đây cùng với một tiêu đề rất đơn giản. Tuy nhiên, khi chúng ta điều hướng đến */about/phannh*, thay vào đó chỉ có tiêu đề thì sẽ có thêm thành phần *phannh.vue* trả về tại vị trí của thẻ *<nuxt />*.

Đây không phải là những gì mà ban đầu chúng ta mong muốn, nhưng ý tưởng để có một trang giới thiệu với danh sách nhân viên mà khi click vào, một phần hiển thị của trang là thông tin của họ là 1 ý tưởng hợp lý, vì vậy hãy để nó như bây giờ. Nếu bạn muốn thêm các tùy chọn khác, thì tất cả những gì chúng ta cần làm là tái cơ cấu lại thư mục. Chúng ta chỉ cần di chuyển thành phần *about.vue* vào bên trong thư mục */about* và đổi tên nó thành *index.vue* :

```
/pages
--| /about
----| index.vue
----| phannh.vue
```

Cuối cùng, nếu muốn sử dụng params trên route để truy xuất vào một sản phẩm cụ thể. Ví dụ: tôi muốn chỉnh sửa sản phẩm bằng cách điều hướng đến */products/edit/1*, trong đó 1 là id của sản phẩm, thì tôi sẽ phải làm như sau:

```
/pages
--| /products
----| /edit
------| _product_id.vue
```

Lưu ý, dấu gạch dưới ở đầu thành phần *_product_id.vue* - điều này biểu thị cho một tham số được truyền trên route mà sau đó có thể truy cập vào trên đối tượng $route.params hoặc trên đối tượng *params*. Khóa cho params sẽ là tên thành phần không có gạch dưới ban đầu - trong trường hợp này, là product_id:

```
<template>
 <h1>Editing Product {{ $route.params.product_id }}</h1>
</template>
```

Store

Nuxt có thể xây dựng kho lưu trữ Vue dựa vào thư mục store, tương tự như thư mục pages. Nếu không cần store, bạn có thể xóa. Có 2 modes cho Store, đó là Classic và Modules.

Classic yêu cầu bạn cần có file *index.js* trong thư mục store. Ở đó, mình cần phải export một hàm trả về một Vue instance:

```
import Vuex from 'vuex'

const createStore = () => {
 return new Vuex.Store({
   state: ...,
   mutations: ...,
   actions: ...
 })
}

export default createStore
```

Điều này cho phép bạn tạo ra các store theo từng mục đích sử dụng mà bạn muốn, nó cũng khá giống cách sử dụng Vuex trong các dự án Vue thông thường.

Chế độ Mudules cũng yêu cầu bạn phải tạo tệp index.js trong thư mục store. Tuy nhiên, tệp này chỉ cần export root state/mutations/actions cho Vuex store của mình.Ví dụ:

```
export const state = () => ({})
```

Sau đó, mỗi tệp trong thư mục Store sẽ phải thêm vào namespace hoặc module của riêng nó. Ví dụ: tạo ra một nơi để lưu trữ sản phẩm hiện tại. Nếu chúng ta tạo tệp *product.js* trong thư mục Store, thì phần namespaced của Store đó sẽ là *$store.product* :

```
export const state = () => ({
 _id: 0,
 title: 'Unknown',
 price: 0
})

export const actions = {
 load ({ commit }) {
   setTimeout(
     commit,
     1000,
     'update',
     { _id: 1, title: 'Product', price: 99.99 }
   )
 }
}

export const mutations = {
 update (state, product) {
   Object.assign(state, product)
 }
}
```

setTimeout được sử dụng trong các lần gọi API, quá trình mà store cập nhật các thông tin trong response sẽ cần 1 khoảng thời gian nhất định, trong TH này là 1s. Bây giờ hãy sử dụng nó trong trang */products/view*:

```
<template>
<div>
  <h1>View Product {{ product._id }}</h1>
  <p>{{ product.title }}</p>
  <p>Price: {{ product.price }}</p>
</div>
</template>

<script>
import { mapState } from 'vuex'
export default {
created () {
  this.$store.dispatch('product/load')
},
computed: {
  ...mapState(['product'])
}
}
</script>
```

Một vài điều cần lưu ý: Ở đây, API fake được gọi khi component được tạo. Bạn có thể thấy rằng hành động */product/load* được gửi đi đặt theo namspaces của Product. Điều này làm cho nó rõ ràng chính xác với những gì mà Store đang thực hiện. Sau đó, bằng cách ánh xạ các trạng thái đến một thuộc tính, chúng ta có thể sử dụng nó trong template của mình.

Components

Thư mục này chứa các thành phần có thể tái sử dụng như các thanh điều hướng, thư viện ảnh, phân trang, bảng dữ liệu, ... Xem các thành phần trong thư mục *Pages* được chuyển vào trong routes, bạn cần một nơi khác để lưu trữ các thành phần này. Các thành phần này có thể truy cập được trong các trang hoặc các thành phần khác bằng cách:

```
import ComponentName from ~/components/ComponentName.vue
```

Assets

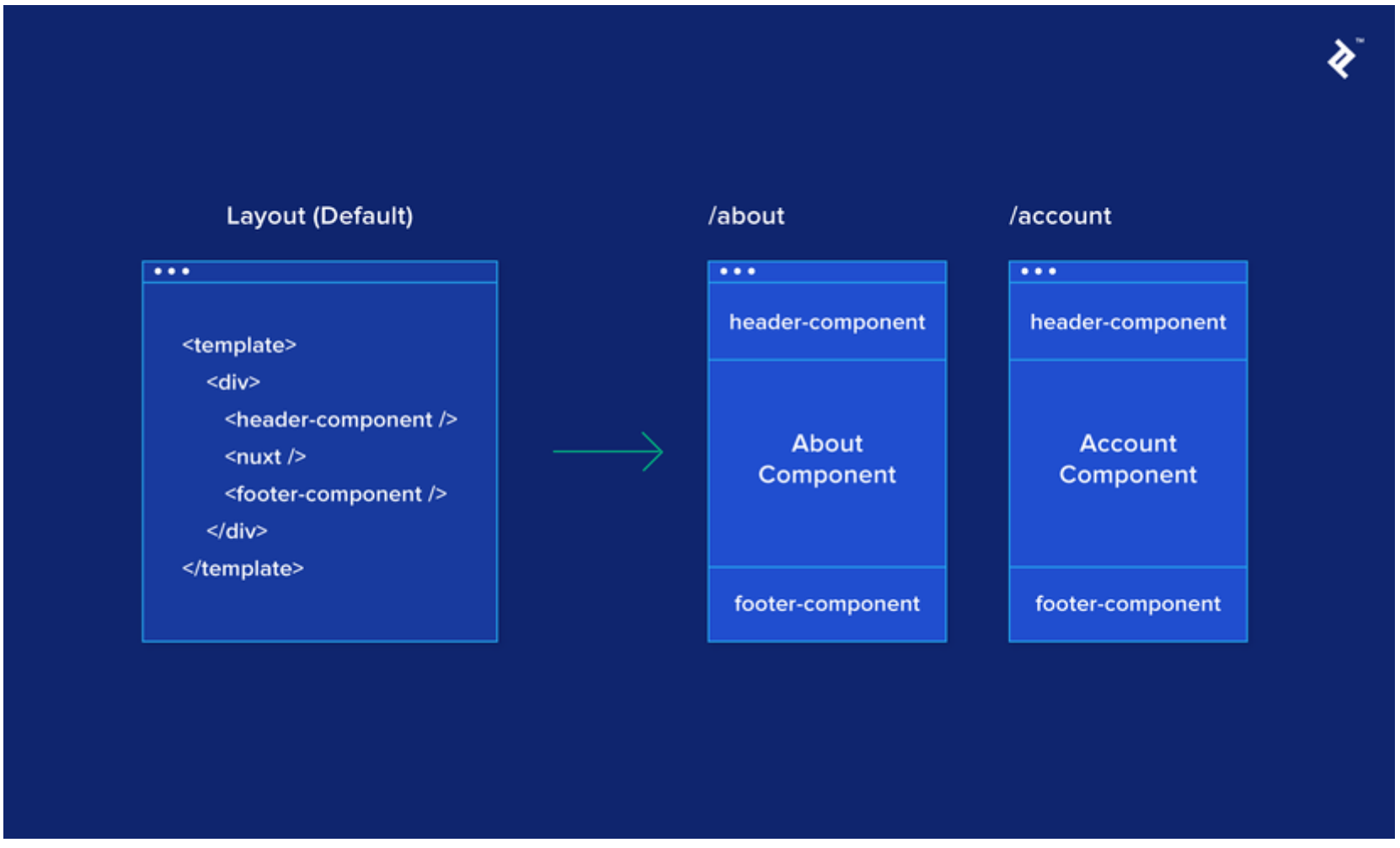
Chứa các assets chưa được biên dịch và có nhiều việc phải làm hơn với cách Webpack tải và xử lý các tệp, thay vì cách Nuxt hoạt động. Bạn có thể tham khảo tại đây.

Static

Chứa các tệp tĩnh được ánh xạ tới thư mục gốc trang web của bạn. Ví dụ, tệp hình ảnh: */logo.png*, hay tệp văn bản: *robots.txt*, *requirement.txt*.....

Layouts

Thông thường, trong một dự án Vue, bạn có một số loại thành phần, thường gọi là *App.vue* . Đây là nơi bạn có thể thiết lập bố cục của ứng dụng (thường là tĩnh), có thể bao gồm các thành điều hường, footer, và nội dung cho bộ đinh tuyến vue-router của bạn.Bố cục mặc định được thực hiện chính xác và được cung cấp cho bạn trong thư mục layouts.Ban đầu, tất là cả một thẻ div với một thành phần <*nuxt* /> (tương đương với <*router-view* />) nhưng nó có thể được tạo ra theo cách mà bạn muốn. Ví dụ tôi có một thanh điều hướng đơn giản (navbar):



Bạn cũng có thể thiết kế một bố cục riêng cho ứng dụng của mình. Có một số loại CMS hoặc panel Admin khác nhau cho bạn lựa chọn. Bạn có thể tạo bố cục mới cho ứng dụng của bạn trong thư mục Layouts. Ví dụ: tạo bố cục *admin-layout.vue* chỉ có thẻ tiêu đề và không có thanh điều hướng:

```
<template>
<div>
  <h1>Admin Layout</h1>
  <nuxt />
</div>
</template>
```

Sau đó, chúng ta có thể tạo trang *admin.vue* trong thư mục *Pages* và sử dụng thuộc tính được cung cấp bởi Nuxt gọi là layout để chỉ định tên (thường là String) của layout mà chúng ta muốn sử dụng cho component đó:

```
<template>
<h1>Admin Page</h1>
</template>

<script>
export default {
layout: 'admin-layout'
}
</script>
```

Đó là tất cả những gì cần phải làm để có page Admin mà bạn muốn. Các thành phần của trang sẽ sử dụng bố cục mặc định trừ khi nó được chỉ định 1 layout cụ thể nào khác, khi bạn điều hướng đến */admin*, bây giờ nó sẽ sử dụng layout *admin-layout.vue* mà bạn định nghĩa ở trên. Tất nhiên, bố cục này có thể được dùng lại trên một số màn hình quản trị nếu bạn muốn. Điều quan trọng cần nhớ là những chỗ muốn sử dụng phải chứa phần tử *<nuxt />* nhé.

Trong các trường hợp bạn điều hướng url mà không tìm thấy trang, một trang lỗi sẽ được hiển thị. Trang lỗi này, trên thực tế, Nuxt đã có định nghĩa tại đây, nhưng nếu bạn muốn làm một trang tương tự cho riêng ứng dụng cho mình với bố cục mà bạn muốn, chỉ cần tạo thành phần *error.vue* và thiết kế theo ý mình muốn.

Middleware

Middleware là các hàm có thể được thực hiện trước khi hiển thị page hoặc layout. Có nhiều lý do mà bạn nên làm như vậy. Bảo vệ định tuyến routes là một cách sử dụng phổ biến nơi bạn có thể kiểm tra Vuex store cho việc đăng nhập hợp lệ hoặc xác thực một số params (thay vì sử dụng phương thức xác thực trên chính thành phần đó).

Các hàm này có thể không đồng bộ nên cần cẩn thận, vì sẽ không có gì được hiển thị cho người dùng cho đến khi thành phần trung gian middleware được thực thi xong.

Plugins

Thư mục này cho phép bạn đăng ký các plugin Vue trước khi ứng dụng được tạo. Điều này cho phép các plugin được sử dụng trong suốt quá trình tương tác với ứng dụng của bạn trên instance Vue và bạn có thể truy cập bất cứ vào thành phần nào.

Hầu hết các plugin chính đều có phiên bản Nuxt có thể dễ dàng được đăng ký cho các thể hiện Vue bằng cách theo dõi tài lại của họ. Tuy nhiên, sẽ có trường hợp khi bạn phát triển plugin hoặc cần phải điều chỉnh plugin hiện tại cho mục đích này. Ví dụ tôi có thể lấy từ tài liệu để thấy cách thức thực hiện của Vue *vue-notifications*. Điều đầu tiên, sẽ cần phải cài đặt package: `npm install vue-notifications --save`

Sau đó, tạo tệp trong thư mục plugin có tên là *vue-notifications.js* và include như sau:

```
import Vue from 'vue'
import VueNotifications from 'vue-notifications'

Vue.use(VueNotifications)
```

Rất giống với cách bạn đăng ký một plugin trong môi trường Vue bình thường. Sau đó chỉnh sửa tệp *nuxt.config.js* tại thư mục dự án và thêm mục sau vào đối tượng module.exports:

```
plugins: ['~/plugins/vue-notifications']
```

Bây giờ bạn có thể sử dụng *vue-notifications* trong app của bạn

Server-side Rendered App (SSR App)

Đây có lẽ là những gì đang nhắm đến khi sử dụng Nuxt. Khái niệm cơ bản để triển khai ở đây là chạy quá trình xây dựng trên bất kỳ nền tảng nào bạn chọn và thiết lập một vài cấu hình. Ở đây, tôi lựa chọn Heroku (bạn có thể tìm hiểu tại đây)

Trước tiên, hãy thiết lập các tập lệnh cho Heroku trong *package.json*:

```
"scripts": {
 "dev": "nuxt",
 "build": "nuxt build",
 "start": "nuxt start",
 "heroku-postbuild": "npm run build"
}
```

Sau đó, thiết lập môi trường Heroku bằng cách sử dụng *heroku-cli* (docs)

```
# set Heroku variables
heroku config:set NPM_CONFIG_PRODUCTION=false
heroku config:set HOST=0.0.0.0
heroku config:set NODE_ENV=production

# deploy
git push heroku master
```

Cân nhắc khi sử dụng SSR

Khi bạn truy cập trang lần đầu tiên, một yêu cầu được gửi tới Nuxt, server sẽ xây dựng trang và hiển thị cho bạn. Sau đó, client sẽ tiếp tục điều hướng và truy xuất các tài nguyên khi được yêu cầu.

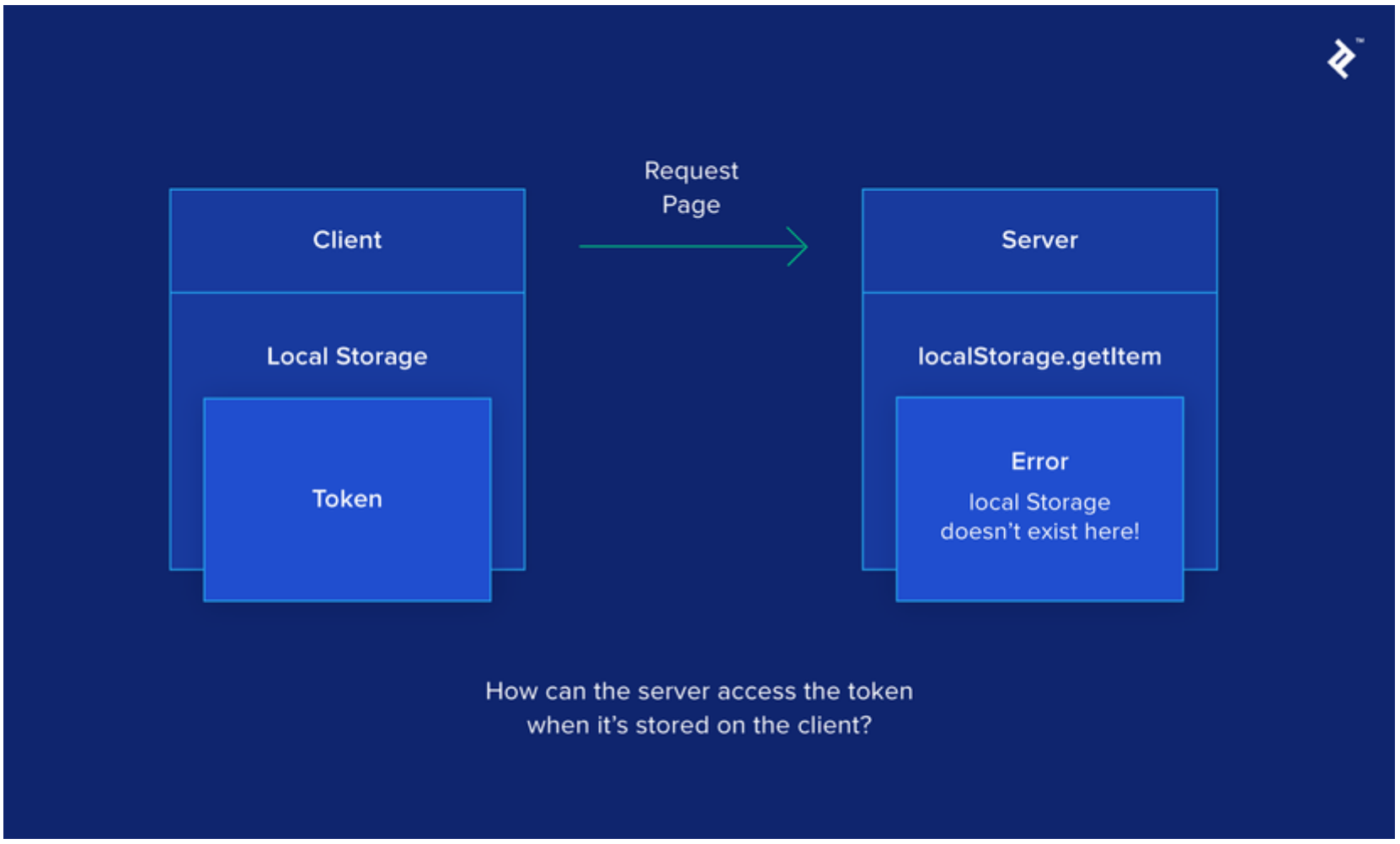
Lần đầu truy cập, chúng ta sẽ muốn server thực hiện được càng nhiều càng tốt, nhưng đôi khi nó không có quyền truy cập vào một số thông tin cần thiết, điều này dẫn đến công việc này được thay thế thực hiện ở phía client. Hoặc tệ hơn, khi nội dung cuối cùng được trình bày phía client khác với những gì phía server mong đợi, phía client được yêu cầu xây dựng lại nó từ đầu. Đây là dấu hiệu lớn cho thấy có điều gì đó sai khác với logic ứng dụng. Thật may mắn, lỗi này được hiển thị ở phía console của browser của bạn (F12 nhé) nếu điều này bắt đầu xảy ra.

Cùng xem ví dụ về cách giải quyết vấn đề quản lý phiên làm việc. Hãy tưởng tượng bạn có một ứng dụng Vue, bạn có thể đăng nhập vào một tài khoản và phiên làm việc của bạn lưu trữ bằng cách sử dụng một đoạn mã token (JWT) và bạn quyết định lưu giữ thông tin này trong *localStorage*. Khi lần đầu truy cập trang web, bạn muốn xác thực mã token này thông qua API, nó sẽ trả về một số thông tin người dùng cơ bản nếu đoạn mã hợp lệ và sẽ lưu lại thông tin này trong Store.

Sau khi tìm hiểu qua về Nuxt, có một phương thức khá tiện dụng là *NuxtServerInit* cho phép bạn lưu trữ không đồng bộ Store trong lần khởi tạo tải dữ liệu khởi tạo đầu tiên. Khi bạn tạo module người dùng trong Store và thêm hành động thích hợp vào tệp index.js trong thư mục Store:

```
export const actions = {
 nuxtServerInit ({ dispatch }) {
   // localStorage should work, right?
   const token = localStorage.getItem('token')
   if (token) return dispatch('user/load', token)
 }
}
```

Khi bạn refresh lại trang, bạn thấy lỗi, *localStorage* không được định nghĩa. Điều gì đang xảy ra ở đây vậy? Phương thức này được chạy ở phía server, nó không có ý tưởng gì về việc được lưu trữ trong *localStorage* phía client; thực tế mà nói thì thậm chí còn không biết "localStorage" là gì, vì vậy, đây không phải là một lựa chọn tốt.



Vậy giải pháp cho vấn đế này là gì? Thực tế thì sẽ có một vài hướng giải quyết. Bạn có thể yêu cầu phía client khởi tạo Store thay cho việc khởi tạo tại server, nhưng cuối cùng lợi ích của SSR lại không được sử dụng vì phía client sẽ kết thúc tất cả các công việc đang được thực hiện. Hoặc bạn cũng có thể thiết lập phiên làm việc trên server và sau đó sử dụng nó để xác thực người dùng, nhưng việc đó sẽ được thiết lập ở một tầng khác. Hoặc thay thế phương thức *localStorage* bằng việc sử dụng cookies, cũng có thể nói nó khá giống với cách thức hoạt động của *localStorage*.

Nuxt có quyền truy cập vào cookies bởi vì nó được gửi kèm trong mỗi request từ client đến server. Chúng ta có thể truy cập thông tin về header và tất cả các thông tin khác thông qua đối tượng *req* trong mỗi request được gửi lên từ phía client.

Vì vậy, sau khi lưu trữ được mã token trong cookies, hãy truy cập nó thông qua trên server:

```
import Cookie from 'cookie'

export const actions = {
 nuxtServerInit ({ dispatch }, { req }) {
   const cookies = Cookie.parse(req.headers.cookie || '')
   const token = cookies['token'] || ''
   if (token) return dispatch('user/load', token)
 }
}
```

Như vậy là ứng dụng SSR của bạn đã sẵn sàng cho mọi người sử dụng. Các nền tảng khác sẽ có các thiết lập khác nhau, nhưng quá trình xây dựng là tương tự nhau: Now, Dokku, Nginx.

Trên đây, mình đã giới thiệu với các bạn về thư viện Nuxt.js xây dựng server-side cho Vue.js app, vì mới tìm hiểu nên có thể có thiếu xót.

Mình có xây dựng 1 app đơn giản: https://zzzkenzzz23.herokuapp.com/ .

Link Github:

Nguồn tài liệu: https://www.toptal.com/vue-js/server-side-rendered-vue-js-using-nuxt-js, https://www.sitepoint.com/nuxt-js-universal-vue-js/

Related

E-mail hoạt động như thế nào?

Nguyen Van Sang
930 12 3 22

Giới thiệu về Thư viện ReactPHP

Hoàng Đức Quân
341 1 5 8

Nuxt.js: Authentication sử dụng Laravel làm server

undefined
214 0 2 4

Xây dựng ứng dụng đơn giản với Laravel và Nuxt.js sử ...

Vũ Nguyễn
199 5 13 15

More from Lucifer

Tìm hiểu về chuẩn hóa cơ sở dữ liệu

Lucifer
224 5 2 8

Một số mẹo và thủ thuật TypeScript

Lucifer
137 2 0 3

5 bước đơn giản để hiểu về JWT (JSON Web Tokens)

Lucifer
1052 8 0 9

OOP trong Javascript

Lucifer
286 1 0 4

Comments

Login to comment