# GIỚI THIỆU

## NUXT

Là một frameword của Vue, Nuxt giúp bạn xây dựng ứng dụng web hiện đại, mạnh mẽ, đặc biệt là **SSR (Server Side Rendering)**, **Static Site Generation (SSG)**, và cả **Hybrid Rendering**.

Có 2 bản là Nuxt 2 và Nuxt 3 hỗ trợ Vue 2 và Vue 3 tương ứng.

### Tự động hóa và quy ước

* Định tuyến: định tuyến tự động dựa trên cấu trúc nằm trong thư mục pages/
* Tự động tách mã nhỏ hơn
* Đi kèm khả năng tích hợp SSR (tự thiếp lập máy chủ)
* Tự động import component

### Công cụ máy chủ

Cung cấp khả năng lập trình fullstack, sử dụng **Rollup** và **Nodejs**. Tạo server API bởi đọc file **server/api/** và server middleware từ **server/middleware**

Khi build project Nitro sẽ build app và server thành 1 file phổ quát là **.output**

## Installation

### Yêu cầu

* Node.js phiên bản 18. trở lên
* IDE lập trình như VSCode hoặc Webstorm

### Cài đặt

npx nuxi@latest init <project-name>

## Configuration

Các tệp cấu hình dự án thường nằm tại thư mục gốc của dự án

VD: myproject/các tệp cấu hình

### nuxt.config.ts

Ghi đè hoặc mở rộng các hành vi của ứng dụng

export default defineNuxtConfig({

// My Nuxt config

})

Hoặc có thể khai báo các biến môi trường tại đây thay vì các file .env như trước kia và được đặt trong biến runtimeConfig (private), runtimeConfig/public.

Các dữ liệu công khai cho phép cả server và client đọc được thì lưu ở **runtimeConfig/public** còn chỉ server đọc được thì lưu ở **runtimeConfig**

export default defineNuxtConfig({

runtimeConfig: {

// The private keys which are only available server-side

apiSecret: '123',

// Keys within public are also exposed client-side

public: {

apiBase: '/api'

}

}

})

Sử dụng bằng cách gọi useRuntimeConfig()

<script setup lang="ts">

const runtimeConfig = useRuntimeConfig()

</script>

Tuy nhiên nếu bạn có file .env nó sẽ ghi đè file config bởi .env (ưu tiên .env hơn)

### app.config.ts

Khá giống với nuxt.config.ts là đều khai báo các biến có thể dùng nhiều nơi trong ứng dụng, tuy nhiên với app.config.ts sẽ có vài điểm khác biệt.

Thường dùng cho cấu hình giao diện của ứng dụng: tên app, màu sắc, cấu hình mặc định cho UI components, và các cấu hình public khác. Khác với runtimeConfig, những thứ này sẽ không thể bị ghi đè bởi các cấu hình môi trường

export default defineAppConfig({

title: 'My Awesome Nuxt App',

theme: {

primaryColor: '#3490dc'

}

})

Cách sử dụng

<script setup lang="ts">

const appConfig = useAppConfig()

console.log(appConfig.title) // "My Awesome Nuxt App"

console.log(appConfig.theme.primaryColor) // "#3490dc"

</script>

### Các bản config khác

Thông thường có thể bạn sẽ có những file config khác như: postcss.config.js, vite.config.ts, tsconfig.json, tailwind.config.js, … Tuy nhiên trong nuxt sẽ gộp tất cả các file cấu hình đó vào 1 nut.config.ts, tuy nhiên chỉ những file liên quan trực tiếp với build system

VD: postcss.config.js thì được viết trong nuxt.config.ts như sau

export default defineNuxtConfig({

postcss: {

plugins: {

tailwindcss: {},

autoprefixer: {}

}

}

})

Tuy nhiên với những file không liên quan trực tiếp đến build system thì không cần cho vào mà vẫn giữ nguyên file: tsconfig.json, eslint.config.js, prettier.config.js, stylelint.config.js, tailwind.config.js, vitest.config.ts, …

## Views

## Styling

### Css toàn dự án

Để import 1 file css global cho toàn bộ dự án, truy cập file nuxt.config.ts

export default defineNuxtConfig({

...

css: ['~/assets/css/main.css']

})

Trong đó ~/assets/css/main.css là đường dẫn đến file css global

### Css một file cụ thể

Có 3 cách import 1 file style thông thường

* Cách 1 (phổ biến nhất)

<script lang="ts" setup>

import "~/assets/css/first.css";

</script>

* Cách 2

<script lang="ts" setup>

import("~/assets/css/first.css");

</script>

* Cách 3

<style>

@import url("~/assets/css/first.css");

</style>

### Font

Nếu bạn có mẫu font riêng, bạn có thể lưu chúng trong mục /public/fonts

VD: /fonts/FarAwayGalaxy.woff

Lúc này trong file main.css (css toàn cục và phải import vào nuxt.config.css):

@font-face {

font-family: 'FarAwayGalaxy';

src: url('/fonts/FarAwayGalaxy.woff') format('woff');

font-weight: normal;

font-style: normal;

font-display: swap;

}

Bây giờ bạn có thể sử dụng font ở bất kì file css nào trong dự án

<style>

h1 {

font-family: 'FarAwayGalaxy', sans-serif;

}

</style>

### Stylesheet

Có thể dùng thư viện css khác để sử dụng

VD: thư viện animate

npm install animate.css

<script lang="ts" setup>

import "animate.css";

</script>

<template>

<section>

<p class="animate\_\_animated animate\_\_shakeX">This page will be displayed at the /about route.</p>

</section>

</template>

Hoặc có thể import trong nuxt.config.ts để có thể sử dụng toàn bộ dự án:

export default defineNuxtConfig({

...

css: ['~/assets/css/main.css','animate.css']

})

### External stylesheet

Bạn cũng có thể sử dụng liên kết ngoài thay vì tải về, sử dụng trong app / head vì đây sẽ là phần khai báo trong thẻ head

export default defineNuxtConfig({

app: {

head: {

link: [{ rel: 'stylesheet', href: 'https://cdnjs.cloudflare.com/ajax/libs/animate.css/4.1.1/animate.min.css' }]

}

}

})

Ngoài ra nuxt cung cấp cho bạn composable useHead() để có thể import cho file cụ thể

useHead({

link: [

{

rel: 'stylesheet',

href: 'https://cdnjs.cloudflare.com/ajax/libs/animate.css/4.1.1/animate.min.css'

}

]

})

### Nitro plugin

Nếu muốn can thiệp sâu hơn vào quá trình render html có thể dùng Nitro plugin để chèn trực tiếp vào head. Tuy nhiên không khuyến khích vì phải tải xong CDN bên ngoài mới render có thể gây chậm lag, nên nếu cài npm sẽ tối ưu hơn.

Tạo 1 file plugin: server/plugins/my-plugin.ts

export default defineNitroPlugin((nitro) => {

nitro.hooks.hook('render:html', (html) => {

html.head.push('<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/animate.css/4.1.1/animate.min.css">')

})

})

### Preprocessor

Cài preprocessor bạn muốn dùng, ví dụ sass:

npm install -D sass

Sau khi cài có thể sử dụng các file .scss .sass

VD:

<style lang="scss">

@use "~/assets/scss/main.scss";

</style>

@use cú pháp scss tốt hơn @import

Hoặc có thể import trong config

export default defineNuxtConfig({

css: ['~/assets/scss/main.scss']

})

**Inject biến/mixins vào mọi file SCSS**

Giả sử bạn có file biến như sau:

assets/\_colors.scss

$primary: #49240F;

$secondary: #E4A79D;

Bạn muốn dùng ở mọi file scss mà không muốn import nó lại mỗi lần

export default defineNuxtConfig({

vite: {

css: {

preprocessorOptions: {

scss: {

additionalData: '@use "~/assets/\_colors.scss" as \*;'

}

}

}

}

})

### Styling Single File Component

Mọi cú pháp tương tự như trong vue

VD:

<script setup lang="ts">

const color = ref("red")

</script>

<template>

<div class="text">hello</div>

</template>

<style>

.text {

color: v-bind(color);

}

</style>

## Routing

Khác với vue thông thường là phải khai báo, import thì nuxt sẽ tự động tải các mục trong folder /pages

VD:

-| pages/

---| about.vue

---| index.vue

---| posts/

-----| [id].vue

### NuxtLink

Giống như <router-link> trong vue. Khi render sẽ thành thẻ a, nhưng khi chuyển trang sẽ mượt mà và không tải lại trang (client-side)

VD:

<NuxtLink to="/about">About Us</NuxtLink>

Vì Component này sẽ được tải trước nên việc nhấn vào sẽ xuất hiện ngay lập tức, có thể tắt tính năng này bằng thuộc tính prefetch="false" trong NuxtLink

### Router có tham số

Trong Nuxt, bạn có thể tạo một file trong thư mục pages/ với tên có **dạng dấu ngoặc vuông** để định nghĩa một **route động**.

VD:

pages/posts/[id].vue

Kết quả:

/posts/:id

Khi bạn truy cập /post/123 thì id sẽ là 123

Ngoài ra dùng useRoute() để lấy giá trị tham số

### Route middleware

Là cho phép xử lý logic trước khi chuyển trang, chẳng hạn như kiểm tra đăng nhập, quyền, chuyển hướng, …

Có 3 loại route middleware:

* Anonymous (inline): Viết trực tiếp trong file pages/\*.vue
* Named: Viết trong middleware/, dùng tên để gọi
* Global: File .global.ts trong middleware/

### Xác thực route

Dùng để kiểm tra tính hợp lệ của một route trong file .vue

VD: Nếu nhập id không phải số thì trả về màn 404

<script setup lang="ts">

definePageMeta({

validate: async (route) => {

return typeof route.params.id === "string" && /^\d+$/.test(route.params.id);

},

});

</script>

<template>

<h1>Post Detail #{{ $route.params.id }}</h1>

</template>

Giải thích: Nếu như không phải là số thì return false (sẽ thành màn 404 – chỉ có giao diện 404 chứ không phải chuyển hướng sang trang 404)

## SEO và META

Cho phép sửa phần tĩnh của html trang

## Transition

Cho phép bạn tạo các hiệu ứng cho trang web

### Page transition

Bạn có thể tạo hiệu ứng chuyển hướng cho trang web

* Bước 1: Cấu hình cho phép hiệu ứng chuyển hướng, tại nuxt.config.ts

export default defineNuxtConfig({

. . .

app: {

pageTransition: { name: "page", mode: "out-in" },

},

});

* Bước 2: Tại trang app.vue nơi chứa NuxtPage thêm các css cho hiệu ứng trang

VD: Hiệu ứng bóng mờ, khi sử dụng class chung (page-enter-active, … ) thì toàn bộ page sẽ có hiệu ứng này

<style>

.page-enter-active,

.page-leave-active {

transition: all 0.4s;

}

.page-enter-from,

.page-leave-to {

opacity: 0;

filter: blur(1rem);

}

</style>

VD: Hiệu ứng quay 1 góc 15 độ mỗi khi chuyển hướng. Đối với hiệu ứng cho 1 trang cụ thể, đầu tiên khai báo code sau ở app.vue

.rotate-enter-active,

.rotate-leave-active {

transition: all 0.4s;

}

.rotate-enter-from,

.rotate-leave-to {

opacity: 0;

transform: rotate3d(1, 1, 1, 15deg);

}

Sau đó qua trang cần hiệu ứng thêm cấu hình trong script

definePageMeta({

pageTransition: {

name: "rotate",

},

});

Hoặc bạn có thể để file css trong file này cũng được, nhưng nếu sử dụng nhiều nơi thì nên đặt trong app.vue để tiện sử dụng.

Nếu bạn đã cấu hình ở file app.vue rồi mà bạn cấu hình ở file cụ thể thì cấu hình file cụ thể sẽ ghi đè lên file cấu hình chung.

Ngoài ra, bạn có thể tắt chức năng hiệu ứng đó bằng cách khai báo:

<script setup lang="ts">

definePageMeta({

pageTransition: false,

layoutTransition: false

})

</script>

### Javascript Hook

Trong các trường hợp nâng cao, bạn muốn sử dụng JS để tạo hiệu ứng linh hoạt hơn

VD:

<script setup lang="ts">

definePageMeta({

pageTransition: {

name: 'custom-flip',

mode: 'out-in',

onBeforeEnter: (el) => {

console.log('Before enter...')

},

onEnter: (el, done) => {},

onAfterEnter: (el) => {}

}

})

</script>

### Hiệu ứng linh hoạt

Để áp dụng các hiệu ứng linh hoạt có điều kiện, bạn có thể dụng middleware để gán các trasition name khác nhau

VD: Khi nhấn next thì lướt phải, nhấn prev thì lướt trái:

<script setup lang="ts">

definePageMeta({

pageTransition: {

name: 'slide-right',

mode: 'out-in'

},

middleware (to, from) {

if (to.meta.pageTransition && typeof to.meta.pageTransition !== 'boolean')

to.meta.pageTransition.name = +to.params.id! > +from.params.id! ? 'slide-left' : 'slide-right'

}

})

</script>

<template>

<h1>#{{ $route.params.id }}</h1>

</template>

<style>

.slide-left-enter-active,

.slide-left-leave-active,

.slide-right-enter-active,

.slide-right-leave-active {

transition: all 0.2s;

}

.slide-left-enter-from {

opacity: 0;

transform: translate(50px, 0);

}

.slide-left-leave-to {

opacity: 0;

transform: translate(-50px, 0);

}

.slide-right-enter-from {

opacity: 0;

transform: translate(-50px, 0);

}

.slide-right-leave-to {

opacity: 0;

transform: translate(50px, 0);

}

</style>

### Hiệu ứng với NuxtPage

Bạn có thể khai báo transition tại nuxtPage mà không cần definePageMeta ở từng page. Tuy nhiên chức năng này mức độ ưu tiên dưới definePageMeta nên có thể sẽ bị ghi đè nên lưu ý.

VD:

<template>

<div>

<NuxtLayout>

<NuxtPage :transition="{

name: 'bounce',

mode: 'out-in'

}" />

</NuxtLayout>

</div>

</template>

### View transition API (thử nghiệm)

Thông thường khi chuyển trang sẽ có một chút ngắt quãng, trang này biến mất trang kia hiện ra, tuy nhiên với chức năng này sẽ giúp chuyển cảnh mượt hơn, không tác dụng với async/await

Tại nuxt.config.ts thêm:

experimental: {

viewTransition: true,

},

Tại component chung, có thể là default layout nếu muốn toàn bộ trang web

<div class="wrapper">

<!-- Hộp nằm ngoài content -->

<div

:class="['box', route.path === '/' ? 'red' : 'blue']"

view-transition-name="color-box"

></div>

<!-- Phần nội dung sẽ thay đổi -->

<NuxtPage />

</div>

.wrapper {

display: flex;

flex-direction: column;

align-items: center;

padding: 50px;

}

.box {

width: 200px;

height: 200px;

margin-bottom: 20px;

border-radius: 16px;

transition: background-color 0.5s ease;

}

.red {

background-color: red;

}

.blue {

background-color: blue;

}

Lúc này khi chuyển giữa các trang sẽ thấy hộp vuông chuyển màu

## Data fetching

### $fetch, useFetch, useAsyncData

Nuxt cung cấp một composabled (component dùng chung tái sử dụng) để xử lý dữ liệu xuyên suốt quá trình sử dụng app.

Nuxt sử dụng: useFetch, useAsyncData, $fetch

* $fetch: cách đơn giản nhất để tạo gửi yêu cầu
* useFetch(): wrapper của $fetch, hỗ trợ render 1 lần trong SSR
* useAsyncData(): giống useFetch nhưng mạnh hơn và kiểm soát tốt hơn

VD1: Đơn giản không tự động tích hợp với Nuxt

const data = await $fetch('/api/posts')

VD2: Có thể chạy ở server (SSR) khi lần đầu vào trang, sau đó tự động cache, không gọi lại ở client nữa.

<script setup>

const { data, pending, error } = await useFetch('/api/posts')

</script>

VD3: Không bắt buộc dùng $fetch có thể dùng bất kỳ logic async nào bên trong

<script setup>

const { data, pending } = await useAsyncData('posts', () => $fetch('/api/posts'))

</script>

Thông thường đối với việc sử dụng $fetch sẽ gọi 1 lần ở server và 1 lần ở client gây lãng phí tài nguyên, thậm chí lỗi dữ liệu. Vậy nên $fetch chỉ nên sử dụng khi muốn gửi request từ browser lên server mà không cần phản hồi

### Suspense

Là tính năng chờ đồng bộ hoá dữ liệu, sau khi hoàn tất sẽ hiển thị nội dung, tuy nhiên thì nuxt đã tích hợp tính năng này nên không cần khai báo Suspense.

VD:

<script setup lang="ts">

const { data } = await useFetch('/api/user')

</script>

<template>

<div>{{ data }}</div>

</template>

### Client only fetching

Thông thường useFetch sẽ gọi API cả ở server và client, để phục vụ SSR. Nhưng nếu dữ liệu bạn cần không quan trọng với SEO hoặc không cần thiết khi mới vào trang, bạn có thể chỉ gọi nó ở phía client, giúp tăng tốc độ tải trang đầu tiên.

const { status, data: comments } = useFetch('/api/comments', {

lazy: true,

server: false

})

* server: false: Không fetch ở server, chỉ fetch ở client sau khi trang được "hydrate".
* lazy: true: Chỉ gọi API khi component thực sự được dùng đến.

### Minimize payload size

Khi bạn dùng useFetch() trên server, dữ liệu bạn lấy sẽ được đưa sẵn vào HTML trả về. Điều này giúp client không cần gọi lại API nữa.

Nhưng nếu bạn lấy quá nhiều dữ liệu, Nuxt sẽ nhúng toàn bộ vào HTML ➜ làm tăng size của trang ➜ ảnh hưởng đến tốc độ tải và SEO.

Giải pháp dùng pick hoặc transform

* Pick: Lấy vài trường trong object
* Transform: Lọc hoặc xử lý lại danh sách

**Pick**

Chọn chính xác các trường bạn cần, giúp giảm dung lượng payload.

VD: HTML sinh ra sẽ chỉ chứa title và description, còn các trường khác (như height, location, image, ...) sẽ không bị đưa vào trang HTML.

const { data: mountain } = await useFetch('/api/mountains/everest', {

pick: ['title', 'description'] // chỉ lấy 2 trường

})

**Transform**

Khi bạn fetch mảng nhiều object, bạn có thể lọc hoặc sửa nội dung.

const { data: mountains } = await useFetch('/api/mountains', {

transform: (mountains) => {

return mountains.map(m => ({

title: m.title,

description: m.description

}))

}

})

### Catching và refetching

**Keys**

Nuxt tự động lưu lại dữ liệu (cache) sau khi fetch, để không phải gọi lại API nếu đã từng gọi rồi

* useFetch dùng URL làm key mặc định.
* useAsyncData dùng chuỗi đầu tiên làm key. Nếu bạn truyền vào hàm trực tiếp, nó sẽ tự tạo key từ tên file và dòng code.

**Dùng chung dữ liệu khi dùng cùng key**

Nếu 2 component hoặc 2 lệnh useFetch/useAsyncData dùng cùng key, chúng sẽ:

* Dùng chung data / error / status.
* Không gọi lại API lần 2 (nếu đã có dữ liệu từ cache).

Tuy nhiên, nếu bạn dùng cùng key thì phải đảm bảo các option quan trọng giống nhau:

|  |  |
| --- | --- |
| Bắt buộc giống nhau | Không cần giống nhau |
| Handler (hàm fetch) | sever |
| deep, pick, transform | lazy |
| getCachedData, default | watch |

Nếu bạn muốn gọi nhiều lần cùng API nhưng không dùng chung cache thì đặt key khác nhau

const { data: u1 } = useAsyncData('u1', () => $fetch('/api/users'))

const { data: u2 } = useAsyncData('u2', () => $fetch('/api/users'))

**Reactive key**

Bạn có thể dùng ref, computed làm key để tự động re-fetch khi key thay đổi:

const userId = ref('123')

const { data: user } = useAsyncData(

computed(() => `user-${userId.value}`),

() => fetchUser(userId.value)

)

userId.value = '456' // sẽ fetch lại dữ liệu

* refresh() hoặc execute() dùng để gọi lại fetch
* clear() để xóa dữ liệu đang có
* clearNuxtData(key) để xóa toàn bộ cache theo key

### Chờ

Thông thường useFetch sẽ gọi ngay khi được khai báo, tuy nhiên có thể tắt tính năng này bằng immediate: false, và khi cần thực thi thì gọi hàm execute

VD:

<script setup lang="ts">

const { data, error, execute, status } = await useLazyFetch('/api/comments', {

immediate: false

})

const onClick = () => {

execute();

}

</script>

### Truyền headers và cookies

Đối với các SSR có bảo mật thì việc sử dụng headers và cookies là điều tất yếu, $fetch sẽ không có những chức năng này, tuy nhiên có thể sử dụng useFetch

## State Management

Nuxt cung cấp useState để tạo mọt phản ứng và SSR thân thiện chia sẻ trạng thái thông qua các component.

useState là một SSR thân thiện thay thế ref. Vì dữ liệu trong useState được chuyển thành JSON, không chứa classes, hàm và ký hiệu.

VD:

<script setup lang="ts">

const counter = useState("counter", () => 0);

</script>

<template>

<div>

Counter: {{ counter }}

<button @click="counter++">+</button>

<button @click="counter--">-</button>

</div>

</template>

**Trạng thái khởi tạo**

Bạn có thể tạo 1 state lưu thông tin chung của app bạn, sau đó các component khác chỉ việc lấy ra khi sử dụng, chẳng hạn như tại app.vue

const config = useState('config', () => ({ title: 'My Site' }))

Nơi sử dụng:

const websiteConfig = useState('config')

### Sử dụng với pinia

Cài đặt thư viện: npx nuxi@latest module add pinia

Tạo 1 store: stores/website.ts

import { defineStore } from "pinia";

export const useGlobalStore = defineStore("globalStore", {

state: () => ({

data: "",

}),

actions: {

async fetch() {

this.data = await $fetch("https://dummyjson.com/products");

},

},

});

Nơi sử dụng:

<script setup lang="ts">

import { useGlobalStore } from "~/stores";

const website = useGlobalStore();

await callOnce(website.fetch);

</script>

<template>

<div>

<h1>{{ website.data }}</h1>

</div>

</template>

## Error handling

Bắt lỗi toàn cục, tạo 1 plugins: plugins/error-handler.ts

export default defineNuxtPlugin((nuxtApp) => {

// Bắt lỗi toàn cục của Vue

nuxtApp.vueApp.config.errorHandler = (error, instance, info) => {

console.error('Global Vue Error:', error)

// Có thể gọi service để gửi log (Sentry, LogRocket, v.v.)

}

// Nuxt hook cũng bắt được tương tự

nuxtApp.hook('vue:error', (error, instance, info) => {

console.error('Nuxt Hook Error:', error)

})

})

## Server

Nuxt sử dụng Nitro làm server engine, nitro trước được làm riêng cho nuxt, nhưng giờ được tách cho UnJS có thể dùng độc lập

### Server side

Là một server side có thể tạo các endpoint, thêm middleware xử lý request/response, có thể trả về text, JSON, html, stream(file, video, …)

### Universal Deployment

Có thể build và deploy nuxt ở bất kỳ đâu:

* Cloudflare workers
* Netlify
* Vercel
* Deno, Bun, Node, …

### Hybrid Rendering với routeRules

Bạn có thể kiểm soát cách render mỗi route của app:

export default defineNuxtConfig({

routeRules: {

'/': { prerender: true }, // Static HTML

'/api/\*': { cache: { maxAge: 3600 } }, // Cache API

'/old-page': { redirect: { to: '/new-page', statusCode: 302 } } // Redirect

}

})

* prerender: Biến trang thành static HTML (SEO tốt)
* cache: Cache route (thường dùng với API)
* redirect: Tự động chuyển trang
* ssr, noScripts, appMiddleware: Điều khiển cách render HTML phía server

## Layer

Layer là một cách tổ chức code để chia sẻ lại cấu trúc, component, composable, config... như một “template” cho dự án khác.

Nó giống như việc bạn có một “dự án gốc” rồi dự án mới kế thừa lại phần cấu hình và thư mục của nó.

Khi nào thì dùng layer

* Dùng chung config: Dùng nuxt.config.ts và app.config.ts để định nghĩa preset
* Thư viện component: Viết component một lần, tái sử dụng trong nhiều app
* Bộ thư viện logic: Dùng composables/ và utils/ chia sẻ hàm tiện ích
* Tạo theme Nuxt: Một bộ theme có thể dùng cho nhiều website
* Chia module theo Domain: Dễ chia module theo tính năng trong dự án lớn

Cách sử dụng:

Tại mục root tạo 1 folder: layers và có thể đăng ký các layer cần sử dụng , có thể được mở rộng bằng extend, và ghi đè

## Prerender

Là quá trình build sẵn HTML cho một số trang, để khi người dùng truy cập thì Nuxt chỉ cần trả về file tĩnh (HTML + JSON), không cần xử lý lại như SSR.

Ưu điểm:

* Nhanh như site tĩnh
* Dễ triển khai lên hosting tĩnh (Vercel, Netlify, Cloudflare…)
* Cải thiện SEO (Google thấy nội dung ngay)

Cách sử dụng: cài thư viện: nuxi generate

npx nuxi generate

Kết quả được lưu vào .output/public, xem thử bằng

npx serve .output/public

Cách thức hoạt động:

* Load html bắt đầu từ trang "/" và từ thư mục "pages/"
* Tìm tất cả thẻ a href và tiếp tục đi sâu vào các liên kết con bên trong

**Chọn một Pre rendering cụ thể**

Khai báo rõ ràng các route cần hoặc không cần prerender:

export default defineNuxtConfig({

nitro: {

prerender: {

routes: ["/user/1", "/user/2"],

ignore: ["/dynamic"],

},

},

});

Hữu ích cho các route không nằm trong HTML (ví dụ: /sitemap.xml, /robots.txt, các API route…)

**routeRules**

Cũng là 1 cách thủ công, cần bật tính năng experimental.inlineRouteRules trong nuxt.config.ts

VD:

export default defineNuxtConfig({

routeRules: {

"/": { prerender: true }, // Trang chủ được prerender

"/blog/\*\*": { prerender: true }, // Các blog liên kết được prerender

"/private": { prerender: false }, // Bỏ qua

},

});

Hoặc đặt trực tiếp trong file page

<script setup>

defineRouteRules({ prerender: true })

</script>

## Deployment

Một ứng dụng Nuxt có thể được triển khai trên node.js server, pre-rendered cho hosting tĩnh, hoặc triển khai trên phi server hoặc CDN

### Nodejs Server

* Entry point

Để build dự án sử dụng lệnh:

node .output/server/index.mjs

* PM2 (Process Manager 2)

Để sử dụng PM2 cần tạo file ecosystem.config.cjs

module.exports = {

apps: [

{

name: 'NuxtAppName',

port: '3000',

exec\_mode: 'cluster',

instances: 'max',

script: './.output/server/index.mjs'

}

]

}

### Static hosting

* SSG với ssr: true (khuyến nghị): Render nội dung thật khi build, Mỗi route sẽ có sẵn file .html và .json, /200.html: dùng cho route fallback (SPA), /404.html: hiển thị khi không tìm thấy route, Tốt cho SEO, tốc độ tải nhanh, tương thích các nền tảng như Netlify, Vercel, Cloudflare...

npx nuxi generate

* SSG với ssr: false (Static SPA)

Không render nội dung trước. Chỉ có thẻ trống <div id="\_\_nuxt"></div> trong HTML. Không có nội dung sẵn trong HTML → client phải tải và render toàn bộ sau khi load, Không tốt cho SEO, HTML lúc đầu gần như rỗng (blank shell)

Chỉ sử dụng khi: App thuần client-side, Không quan tâm SEO, Hoặc nội dung không thể SSR (ví dụ: chỉ chạy được trên trình duyệt)

export default defineNuxtConfig({

ssr: false

})

Nếu chỉ một phần không SSR được hãy dùng ClientOnly

<ClientOnly>

<your-client-component />

</ClientOnly>

### Client-side Only Rendering

Nếu bạn không muốn sử dụng pre-render, một cách khác là sử dụng trang tĩnh hosting để cài thuộc tính ssr thành false trong nuxt.config. nuxi generate sẽ trả về .output/public/index.html và gói javascript như Vuejs ứng dụng thông thường.

export default defineNuxtConfig({

ssr: false

})

### Host Provider

NUXT có thể được triển khai cho một số nhà cung cấp đám mây với số lượng cấu hình khá ít

## Upgrate

Để upgrate nuxt chạy lệnh:

npx nuxi upgrade