



รู้จักกับ Nuxt.js

# Nuxt.js คืออะไร



เป็น JavaScript Framework ที่พัฒนาต่อออกมาจาก Vue.js เพื่อแก้ปัญหาสำหรับเว็บที่ต้องการทำ SEO (Search Engine Optimization) คือทำให้เว็บติดหน้าแรกของ Search Engine ทำให้เว็บค้นหาแล้วเจอได้ง่ายมากยิ่งขึ้น เนื่องจาก Nuxt.js นั้นช่วยทำ SSR (Server Side Rendering)

# ต้องมีพื้นฐานอะไรบ้าง

- HTML5
- CSS3
- JavaScript
- Vue.js 3.x



# จุดเด่นของ Nuxt 3

- Server Side Rendering (SSR)
- Static Site Generation (SSG) ทำการ build ไฟล์ vue.js ทั้งหมดออกมาเป็นไฟล์ HTML / CSS / JavaScript (Static Website)
- Hot Reload (แก้ไขการทำงานแล้วเห็นผลทันที)
- Routing สามารถจัดการ Routing ง่ายๆผ่าน File System (Pages)
- รองรับการทำงานกับ Vuejs 3
- รองรับการทำงานกับ TypeScript

# เครื่องมือ

- Visual Studio Code
- Node.js
- Google Chrome





รู้จักกับ CSR , SSR และ SSG

# รู้จักกับ CSR , SSR และ SSG

- Client Side Rendering (CSR)
- Server Side Rendering (SSR)
- Static Site Generation (SSG)

# รู้จักกับ CSR , SSR และ SSG

- Client Side Rendering (CSR)
- Server Side Rendering (SSR)
- Static Site Generation (SSG)



# CSR (Client Side Rendering)

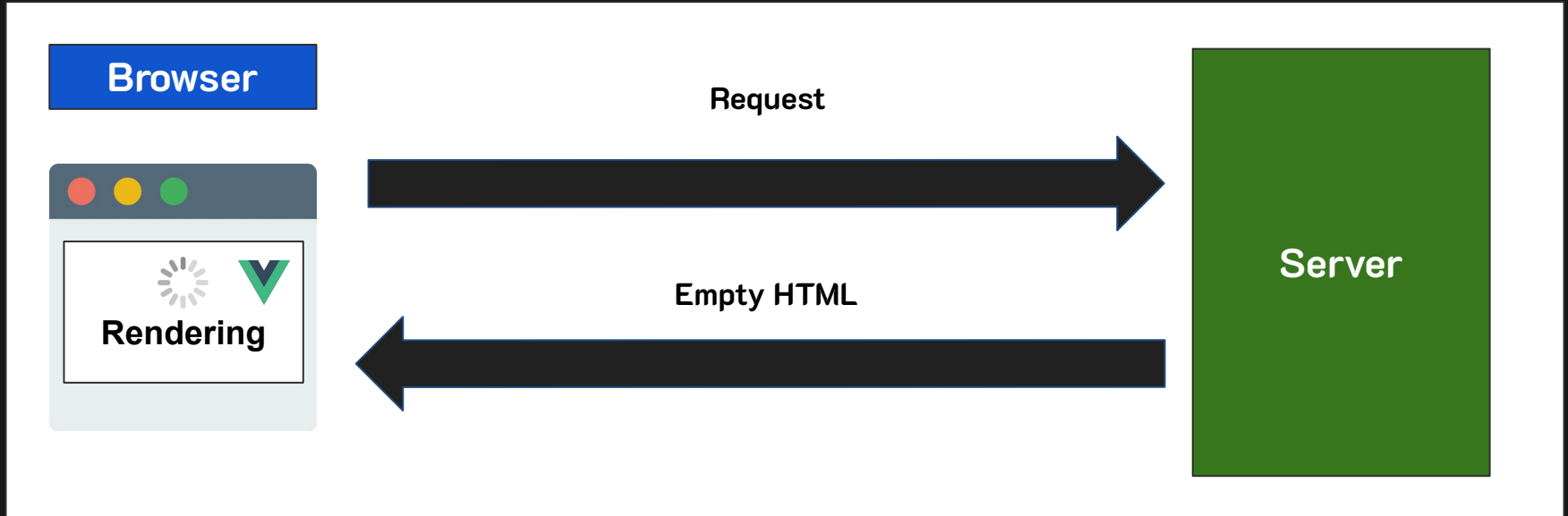
**CSR (Client Side Rendering)** กระบวนการทำงานทั้งหมดจะเกิดขึ้นที่ฝั่งผู้ใช้ (Client) ตั้งแต่การโหลดหน้าเว็บและ Execute JavaScript โดยอาศัยทรัพยากรเครื่องผู้ใช้ทั้งหมด ซึ่งจะรับไฟล์ HTML มาก่อนแล้วจึงจะมีการโหลดเนื้อหาอื่น ๆ ตามมาในภายหลัง

# CSR (Client Side Rendering)

จากนั้นจะเป็นหน้าที่ของ Framework/Library  
ที่จะ Render HTML และควบคุมการทำงานของเว็บ  
ไซต์จาก Browser ของผู้ใช้งาน



# CSR (Client Side Rendering)



# ข้อดีของ CSR

- ลดต้นทุนด้าน Server เนื่องจากขั้นตอนการประมวลผลเว็บไซต์มาอยู่ที่เว็บเบราว์เซอร์ของฝั่งผู้ใช้งาน
- รองรับ Single Page Application (SPA)
- รองรับ Static Website สามารถทำงานได้เลยโดยไม่ต้องพึ่งพากันรันแอปพลิเคชันผ่าน Server

# ข้อเสียของ CSR

- หากเว็บไซต์มีขนาดใหญ่ก็จะใช้เวลาโหลดข้อมูลนาน ส่งผลต่อประสิทธิภาพของผู้ใช้ในการใช้งานเว็บ
- ส่งผลต่ออันดับเว็บไซต์ (SEO) ที่ขึ้นมาจาก Search Engine เนื่องจากมองเห็นเนื้อหาบนเว็บไซต์ได้ยากกว่าแบบ SSR (Server Side Rendering)

# รู้จักกับ CSR , SSR และ SSG

- Client Side Rendering (CSR)
- Server Side Rendering (SSR)
- Static Site Generation (SSG)

# รู้จักกับ CSR , SSR และ SSG

- Client Side Rendering (CSR)
- Server Side Rendering (SSR)
- Static Site Generation (SSG)

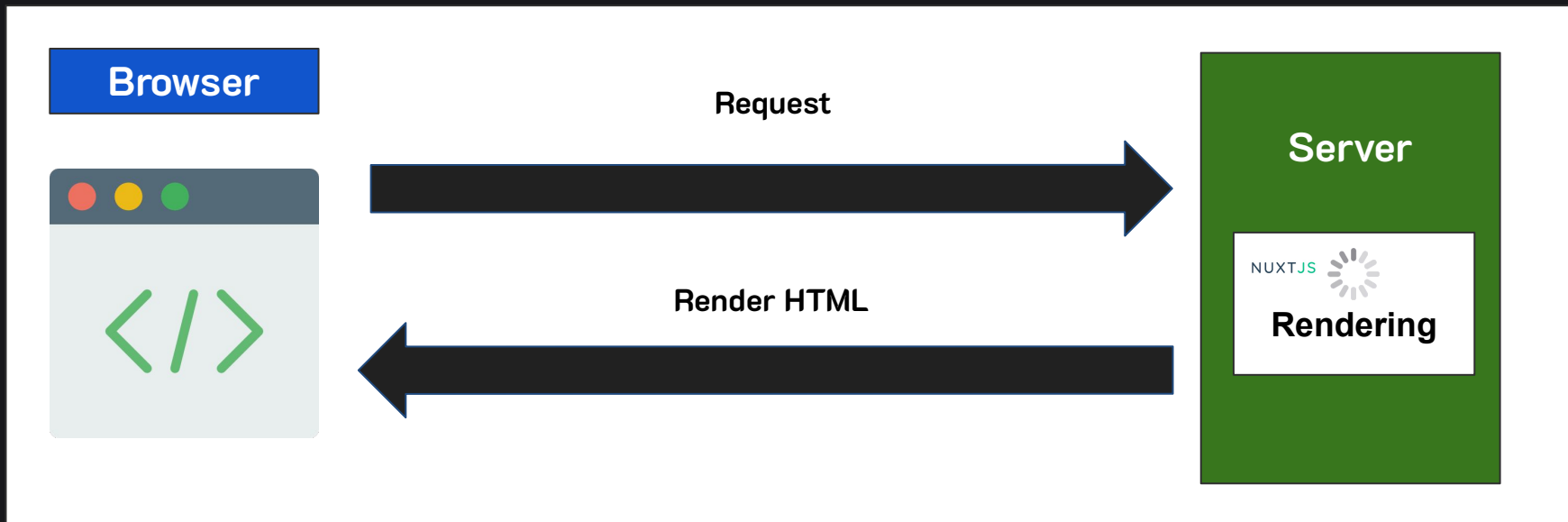
# SSR (Server Side Rendering)

**SSR (Server Side Rendering)** กระบวนการทำงานทั้งหมดจะย้ายไปที่ฝั่ง Server โดยจะทำการ Render หน้าเว็บบน Server ก่อนที่จะส่งมาให้ผู้ใช้งาน (Client)

หมายความว่าหน้าเว็บที่ส่งมาให้กับผู้ใช้ นั้นพร้อมใช้งานแล้ว  
เว็บเบราว์เซอร์สามารถนำไปแสดงผลได้ทันที ทำให้การใช้งานเว็บไซต์นั้นมีความรวดเร็วมากขึ้น



# SSR (Server Side Rendering)



# ข้อดีของ SSR

- หน้าเว็บสามารถพร้อมใช้งานได้ทันที เนื้อหาครบถ้วน ตั้งแต่ Request แรก ทำให้ผู้ใช้งานสามารถเห็นเนื้อหา ของเว็บไซต์ทั้งหมด โดยไม่จำเป็นต้องรอโหลดทีละส่วน
- ช่วยเรื่อง SEO เนื่องจากมีการแสดงผลเนื้อหาในหน้าเว็บ ทั้งหมดไว้สำหรับ Search Engine Bot

# ข้อเสียของ SSR

- เพิ่มต้นทุนด้าน Server เนื่องจากขั้นตอนการประมวลผลเว็บไซต์ทั้งหมดย้ายมาอยู่ที่ฝั่ง Server
- ไม่สามารถทำงานกับ JavaScript Library หรือ Framework บางตัวได้

# ข้อเสียของ SSR

- เนื่องจาก Server นั้นต้องมีการตอบสนองกับ Request ของผู้ใช้งาน ซึ่งการส่ง Request แต่ละครั้งจะมีการประมวลผลหรือ Render เว็บไซต์ทั้งหน้า ดังนั้นหากมีการเรียกใช้มากเกินไปจะส่งผลให้ Server ล่มได้

# รู้จักกับ CSR , SSR และ SSG

- Client Side Rendering (CSR)
- Server Side Rendering (SSR)
- Static Site Generation (SSG)

# รู้จักกับ CSR , SSR และ SSG

- Client Side Rendering (CSR)
- Server Side Rendering (SSR)
- Static Site Generation (SSG)

# SSG (Static Site Generation)

**SSG (Static Site Generation)** มีลักษณะคล้ายกับ SSR (Server Side Rendering) แต่กระบวนการทำงานของ SSG นั้นจะ Render HTML ล่วงหน้าไว้เรียบร้อยแล้วตั้งแต่ตอน Build และพร้อมนำไปใช้งานได้ทันที แต่จะไม่ Render ทุกๆ Request เหมือนกัน SSR

# SSG (Static Site Generation)

กล่าวคือ SSG นั้นจะสร้างไฟล์ HTML เสรียบพร้อมไว้  
เมื่อ Build เสร็จเรียบร้อยแล้ว ก็จะได้ชุดของไฟล์ HTML ที่  
สามารถนำไปทำงานแบบเดียวกับ Client Side  
Rendering (CSR) โดยที่ไม่จำเป็นต้องรันแอปพลิเคชัน  
ไว้รับ Request ตลอดเวลานั่นเอง (ไม่มี Server)



# ข้อดีของ SSG

- เนื่องจากหน้าเว็บมีข้อมูลต่าง ๆ ครบถ้วนอยู่แล้ว ไม่จำเป็นต้องมี Server ไว้เก็บข้อมูล จึงมีความปลอดภัยสูง
- ประหยัดค่าใช้จ่ายด้าน Server เพราะไม่มีการ Render ที่ฝั่ง Server เลย การทำงานมีแค่ส่งไฟล์หน้าเว็บไปที่ฝั่งผู้ใช้งาน และแสดงผลหน้าเว็บในไฟล์ดังกล่าว

# ข้อเสียของ SSG

- เมื่อมีการอัปเดตกระบวนการทำงานในเว็บไซต์ต้อง Build หน้าเว็บใหม่ทั้งหมด เนื่องจากไม่มี Server มารองรับการอัปเดตกระบวนการทำงานดังกล่าว
- หากเว็บไซต์มีเนื้อหาจำนวนมากหรือมีขนาดใหญ่ก็จะส่งผลต่อระยะเวลาในการ Build ด้วย

ติดตั้ง Node.js

# ติดตั้ง Visual Studio Code

สร้างโปรเจกต์ Nuxt.js

# สร้างโปรเจกต์ Nuxt.js

```
npx nuxi init <ชื่อโปรเจกต์>
```

- cd ชื่อโปรเจกต์
- npm install
- npm run dev

โครงสร้างโปรเจกต์

# โครงสร้างโปรเจกต์

```
✓ Nuxt-BASIC
  > .nuxt
  > node_modules
  ✓ public
    ★ favicon.ico
    ✖ .gitignore
    npm .npmrc
    ▼ app.vue
    TS nuxt.config.ts
    {} package-lock.json
    {} package.json
    ⓘ README.md
    TS tsconfig.json
```



# โครงสร้างโปรเจกต์

- **public** คือ โฟลเดอร์ที่ใช้จัดเก็บไฟล์ที่นำไปใช้งานในโปรเจกต์ เช่น ไฟล์รูปภาพ เป็นต้น
- **node\_modules** คือ โฟลเดอร์สำหรับจัดเก็บโมดูลหรือไลบรารีที่ทำงานภายในโปรเจกต์
- **package.json** คือ ไฟล์ที่เก็บข้อมูลต่างๆรวมถึง package ที่จะใช้ทำงานภายในโปรเจกต์

# โครงสร้างโปรเจกต์

- **app.vue** คือ ไฟล์เพจหลักสำหรับรันแอปพลิเคชันในตอนเริ่มต้นโดยนำส่วนประกอบต่างๆ มาประกอบกันแล้วนำไปแสดงผลในเบราว์เซอร์
- **nuxt.config.ts** คือ ใช้สำหรับตั้งค่าโปรเจกต์หรือกระบวนการทำงานในโปรเจกต์ เช่น การเพิ่ม plugins กำหนดส่วนของ meta , header เป็นต้น

# Pages & Routing

# Pages & Routing

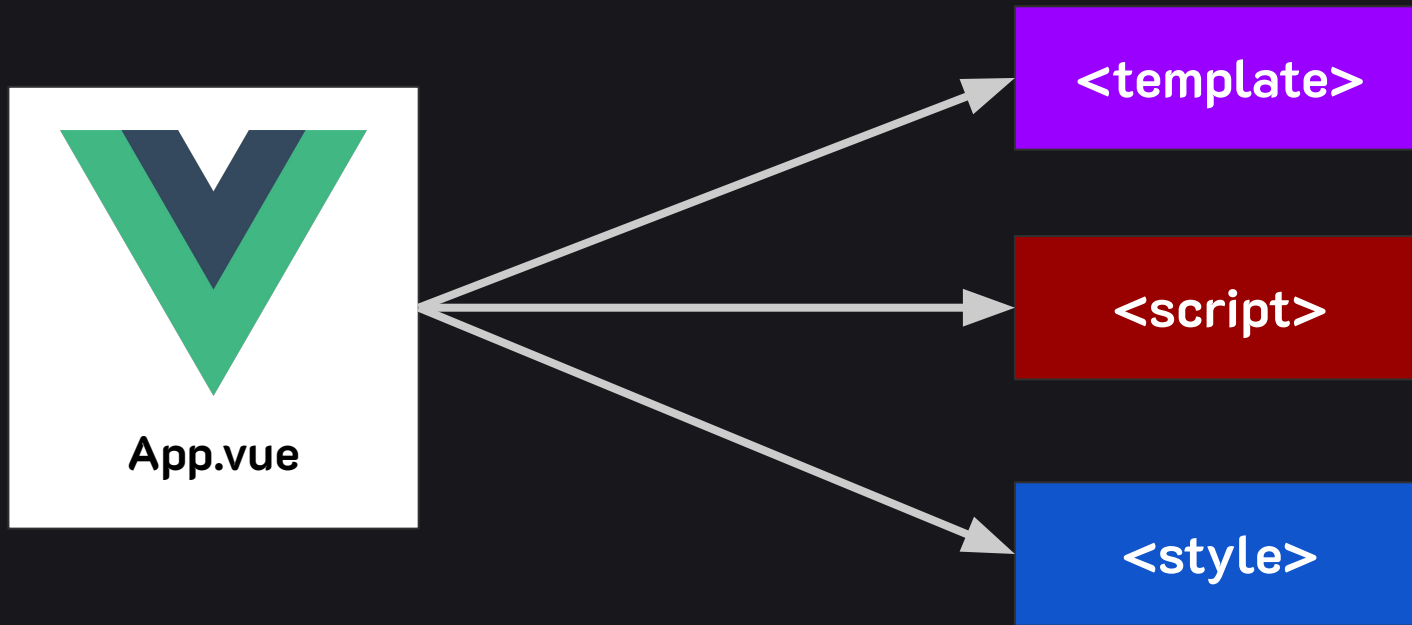
- **pages** คือ โฟลเดอร์ที่ใช้เก็บไฟล์หน้าเว็บเพจหรือส่วนที่แสดงผลต่างๆ ภายในเว็บไซต์ของเรา การสร้างโฟลเดอร์และไฟล์จะมีผลต่อการกำหนดเส้นทาง (Routing) หรือ URL ของเว็บ โดย Nuxt.js นั้นจะจัดการ Route ผ่านโฟลเดอร์ pages และสร้างไฟล์ .vue ก็สามารถเข้า Route ตามชื่อไฟล์ที่สร้างได้เลย (ยกเว้น index.vue ไม่ต้องอ้างอิง)

# Pages & Routing

ตัวอย่าง URL	Pages
<code>https://localhost:3000/</code>	<code>pages/index.vue</code>
<code>https://localhost:3000/about</code>	<code>pages/about.vue</code>
<code>https://localhost:3000/products/</code>	<code>pages/products/index.vue</code>



# พื้นฐานเกี่ยวกับคอมโพเนนต์



# พื้นฐานเกี่ยวกับคอมโพเนนต์

Vue จะแยกแอปพลิเคชันออกเป็นส่วนย่อยๆ  
เรียกว่า **คอมโพเนนต์ (Component)**

โดยคอมโพเนนต์ใน Vue ก็คือ ไฟล์ที่มีนามสกุล  
.vue ซึ่งโครงสร้างภายในไฟล์ดังกล่าวจะ  
ประกอบด้วย 3 องค์ประกอบหลักๆ ดังนี้

# พื้นฐานเกี่ยวกับคอมโพเนนต์

- แท็ก `<template></template>` สำหรับแสดงผล
- แท็ก `<script></script>` สำหรับเขียนคำสั่งเพื่อควบคุมการทำงานของคอมโพเนนต์ เช่น กำหนด data , method
- แท็ก `<style></style>` เขียน css เพื่อตกแต่งคอมโพเนนต์



# พื้นฐานเกี่ยวกับคอมโพเนนต์

```
1 <template>  
2   <div>  
3     <h1>หน้าแรกของเว็บไซต์</h1>  
4   </div>  
5 </template>
```

<template>

```
7 <script setup>  
8  
9 </script>
```

<script>

```
11 <style scoped>  
12  
13 </style>
```

<style>

# Nuxt Link

# Nuxt Link

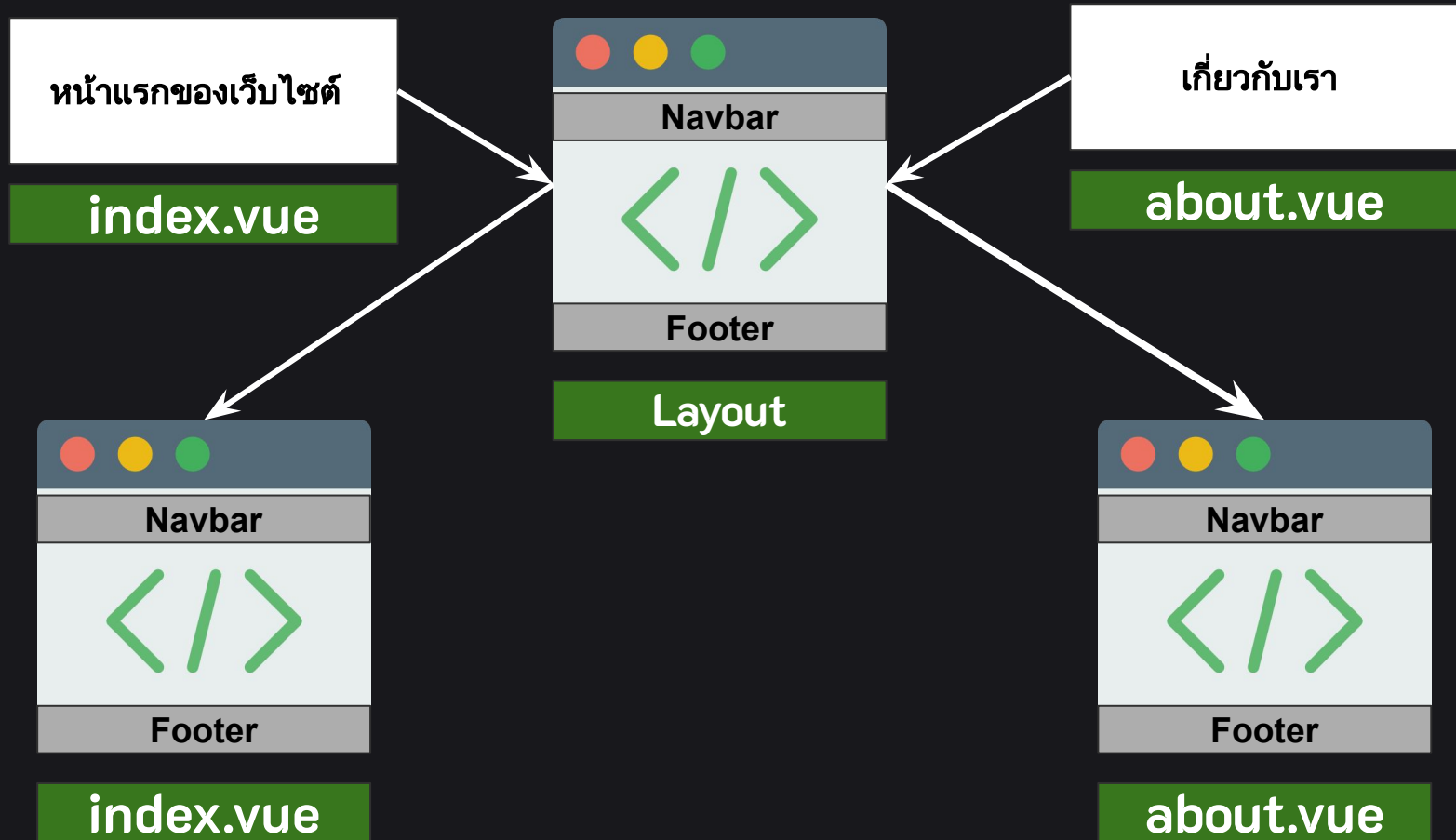
ถ้าต้องการอยากให้หน้าเว็บเพจแต่ละหน้ามีการเชื่อมโยงกัน  
ใน Nuxt.js จะใช้คอมโพเนนต์ NuxtLink

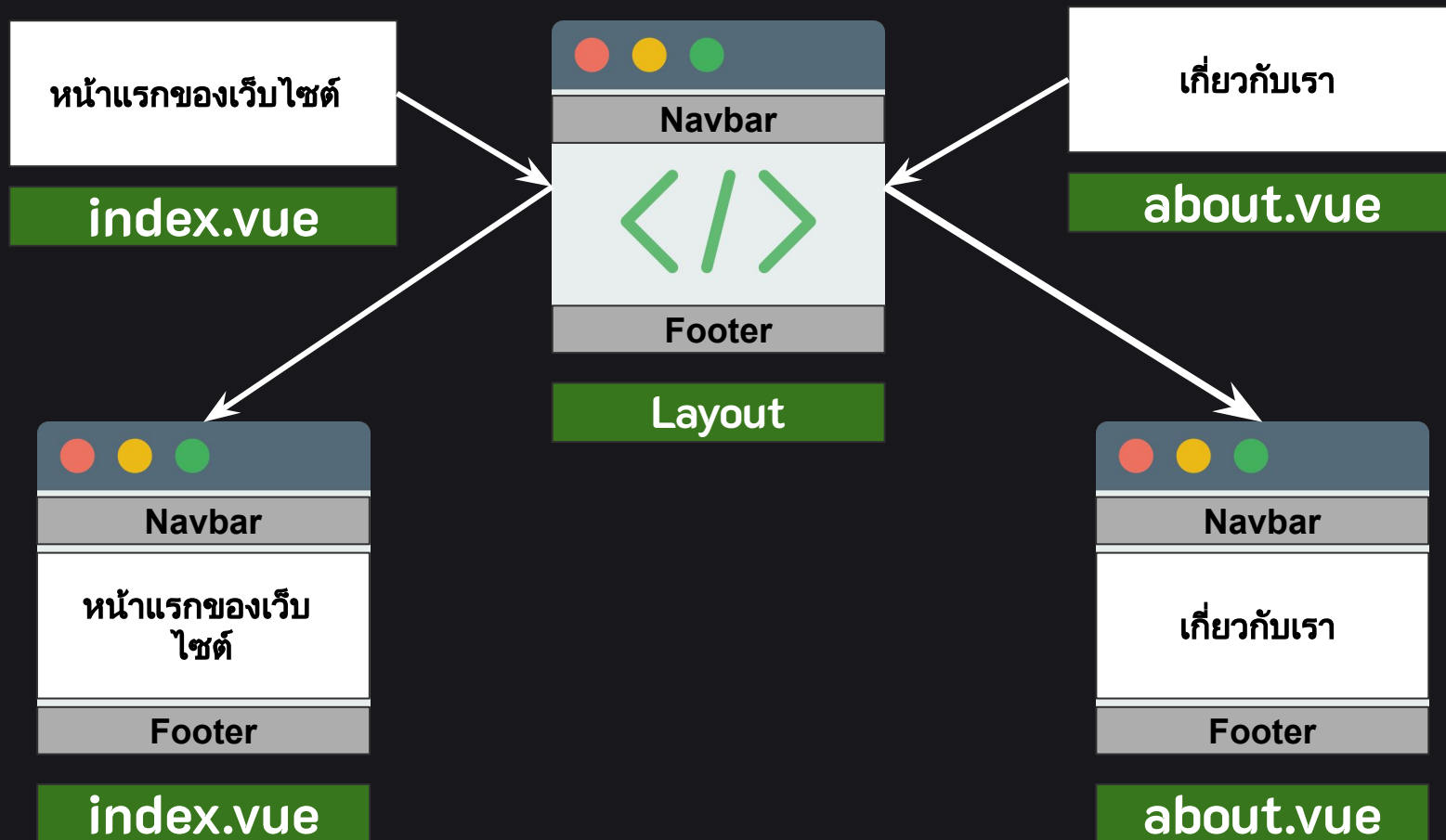
# Layouts

# Layouts

คือ การกำหนดโครงสร้างหลักในหน้าเว็บเพจที่  
ทุกๆหน้า ใช้งานร่วมกันเพื่อลดความซ้ำซ้อน  
ของโค้ด เช่น Navbar และ Footer เป็นต้น







# Layouts

**Slot** ถูกนำมาทำงานร่วมกับการแสดงผลในโครงสร้างหลัก (Layout) ที่ทำงานในแอปพลิเคชัน แต่จะบรรจุเนื้อหา  
ด้านในแตกต่างกัน ส่งผลให้ Component มีความแตกต่างหลากหลายมากขึ้น



ดึงข้อมูลจาก API

# ภาพรวมโปรเจกต์

- สร้างโปรเจกต์ในรูปแบบ SSG : static HTML + JSON
- API ที่ใช้งาน คือ <https://dummyjson.com/>
- ดึงข้อมูลสินค้าทั้งหมดมาแสดงผล
- ดูรายละเอียดสินค้าแต่ละรายการได้