



Nodejs เบื้องต้น

[ฉบับปรับปรุง 2021]

แนะนำเนื้อหา





เล่นทั้งหมด

สอน NodeJS สำหรับมือใหม่หัดขับ

วิดีโอ 25 รายการ • การดู 138,397 ครั้ง • อัปเดตล่าสุดเมื่อ 28 ม.ค. 2017

≡+ ↺ ↻ ...

 Kong Ruksiam [ติดตาม](#)

- สอน NodeJS ตอนที่ 1 - Download & Install
Kong Ruksiam 8:49
- สอน NodeJS ตอนที่ 2 - HelloWorld
Kong Ruksiam 4:29
- สอน NodeJS ตอนที่ 3 - Module Export Require
Kong Ruksiam 16:25
- สอน NodeJS ตอนที่ 4 - File System Read File
Kong Ruksiam 7:23
- สอน NodeJS ตอนที่ 5 - File System Write File
Kong Ruksiam 5:34



<https://www.youtube.com/c/KongRuksiamOfficial/>



<https://www.facebook.com/KongRuksiamTutorial/>

ผู้เรียนต้องมีพื้นฐานอะไรบ้าง



- HTML5 เบื้องต้น
- CSS3 เบื้องต้น
- JavaScript เบื้องต้น
- JavaScript ES6
- MongoDB เบื้องต้น 2021





เรียนรู้พื้นฐานได้ที่ไหน



<https://www.youtube.com/c/KongRuksiamOfficial/>



<https://www.facebook.com/KongRuksiamTutorial/>



เนื้อหา HTML , CSS , JavaScript

ปูพื้นฐาน HTML | CSS3 | JavaScript | Bootstrap5 ▾



ปูพื้นฐานการสร้างเว็บด้วย HTML5 [2020]

KongRuksiam Official
ดูทั้งเพลย์ลิสต์



ออกแบบเว็บไซต์ด้วย CSS3 [2020]

KongRuksiam Official
ดูทั้งเพลย์ลิสต์



ปูพื้นฐานการสร้างเว็บด้วย JavaScript เบื้องต้น [2020]

KongRuksiam Official
อัปเดตแล้ววันนี้
ดูทั้งเพลย์ลิสต์



สอน Bootstrap 5 สำหรับผู้เริ่มต้น [2021]

KongRuksiam Official
ดูทั้งเพลย์ลิสต์



เนื้อหา JavaScript Workshop [Promote Version]

KongRuksiam Official
ดูทั้งเพลย์ลิสต์



ปูพื้นฐาน JavaScript ES6 สำหรับมือใหม่

KongRuksiam Official



คอร์ส JavaScript 30 Workshop | ฉบับ 2020 [Family]

KongRuksiam Official



คอร์ส JavaScript 25 Workshop (For Supporter)

KongRuksiam Official



สแกนเพื่อเข้าเรียน

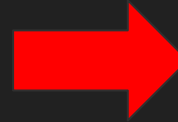


<https://www.youtube.com/c/KongRuksiamOfficial/>



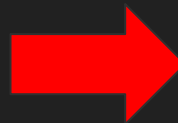
<https://www.facebook.com/KongRuksiamTutorial/>

JavaScript ES6



สแกนเพื่อเข้าเรียน

จัดการฐานข้อมูลด้วย MongoDB



สแกนเพื่อเข้าเรียน

รู้จักกับ JavaScript และ Nodejs



A yellow square containing the letters "JS" in a large, bold, black sans-serif font, representing the JavaScript logo.

JavaScript คือ ภาษาคอมพิวเตอร์ที่ใช้
ในการพัฒนาเว็บร่วมกับ HTML เพื่อให้เว็บมี
ลักษณะแบบไดนามิก หมายถึง เว็บสามารถ
ตอบสนองกับผู้ใช้งานหรือแสดงเนื้อหาที่แตก
ต่างกันไปได้โดยจะอ้างอิงตามเว็บเบราว์เซอร์ที่ผู้
เข้าชมเว็บใช้งานอยู่



<https://www.youtube.com/c/KongRuksiamOfficial/>



<https://www.facebook.com/KongRuksiamTutorial/>

รู้จักกับ JavaScript และ Nodejs

A yellow square containing the letters "JS" in a large, bold, black sans-serif font.

JS

เป็นภาษาที่ทำงานฝั่งผู้ใช้ (Client Side Script)
โดยเว็บเบราว์เซอร์จะทำหน้าที่ประมวลผลคำสั่งที่
ถูกเขียนขึ้นมาและตอบสนองต่อผู้ใช้ได้ทันที เช่น
การแสดงความแจ้งเตือน (Alert) การตรวจสอบ
ข้อมูลที่ใช้ป้อน (Validation) เป็นต้น

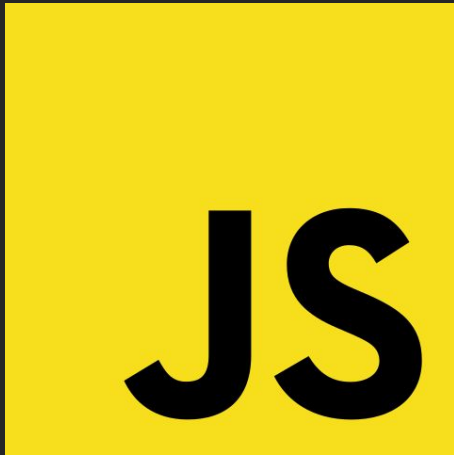


<https://www.youtube.com/c/KongRuksiamOfficial/>



<https://www.facebook.com/KongRuksiamTutorial/>

รู้จักกับ JavaScript และ Nodejs



การที่ JavaScript สามารถทำงานได้ตามโค้ดหรือคำสั่งที่เขียนขึ้นมาได้นั้น ต้องอาศัยตัวแปลคำสั่ง ซึ่งปกติหน้าเว็บเพจจะรันภายใน Web Browser แล้วใช้ตัวแปลคำสั่งใน Web Browser ประมวลผลเพื่อให้ Script หรือคำสั่งที่เขียนสามารถทำงานได้



รู้จักกับ JavaScript และ Nodejs



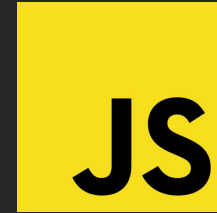
ในปัจจุบันได้มีการพัฒนาชุดแปล
คำสั่งของ JavaScript ขึ้นมาใหม่โดยที่ไม่
ต้องขึ้นอยู่กับ Web Browser นั่นก็คือ



รู้จักกับ JavaScript และ Nodejs



Node.js เป็นชุดเครื่องมือในการแปลคำสั่ง
ของ JavaScript และ เป็น JavaScript
Runtime Environment กล่าวคือ สามารถนำ
JavaScript ไปรันใน Windows , Mac , Linux
ได้ โดยไม่ขึ้นกับ Web Browser



<https://www.youtube.com/c/KongRuksiamOfficial/>

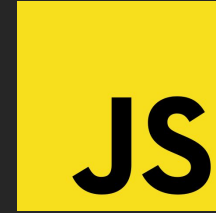
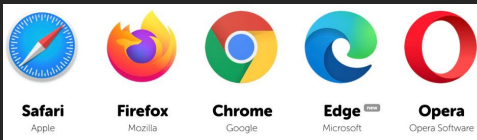


<https://www.facebook.com/KongRuksiamTutorial/>

รู้จักกับ JavaScript และ Nodejs



ส่งผลให้สามารถรันโค้ด JavaScript ด้วย
Nodejs ได้เลย โดยไม่จำเป็นต้องสร้างเป็นเว็บเพ
จแล้วนำเว็บเพจไปรันใน Web Browser นั้นเอง
(ไม่จ้อ Web Browser)



<https://www.youtube.com/c/KongRuksiamOfficial/>



<https://www.facebook.com/KongRuksiamTutorial/>

รู้จักกับ JavaScript และ Nodejs



Browser



<https://www.youtube.com/c/KongRuksiamOfficial/>



<https://www.facebook.com/KongRuksiamTutorial/>

รู้จักกับ JavaScript และ Nodejs



Node.js ถูกสร้างขึ้นมาเพื่อทำงานฝั่ง Server เป็นหลัก คล้ายๆ กับ PHP , Django Framework (Python) , Laravel Framework (PHP) แต่การใช้งาน Nodejs จะมีข้อดีคือ ผู้พัฒนาเว็บสามารถควบคุมการทำงานของเว็บทั้งฝั่ง Frontend และ Backend ได้โดยใช้ JavaScript เพียงภาษาเดียว โดยที่ไม่ต้องเสียเวลาเรียนรู้หลายภาษา

รู้จักกับ JavaScript และ Nodejs



Frontend คือ การพัฒนาโปรแกรมระบบหน้าบ้าน (UI : User Interface หรือ หน้าตาของแอปพลิเคชัน) โดยผู้ใช้งานสามารถมองเห็นและมีส่วนร่วมหรือโต้ตอบภายใน Web Browser ได้



Backend คือ การพัฒนาโปรแกรมหลังบ้านหรือการทำงานเบื้องหลังในแอป เช่น การทำงานกับฐานข้อมูล เป็นต้น โดยผู้ใช้งานไม่สามารถมีส่วนร่วมหรือโต้ตอบได้

ข้อดีของ Nodejs



- Nodejs ใช้ JavaScript ในการพัฒนาเว็บทั้งฝั่ง Frontend และ Backend
- ทำงานแบบ Non-Blocking I/O โดยใช้วิธีการแบบ Asynchronous โดยจะไม่รอการตอบสนองแต่ละ Request ให้แล้วเสร็จ แต่จะทำการย้ายการทำงานไปอยู่เบื้องหลัง แล้วรอรับ Request ต่อๆ ไปทันที
- แบ่งการทำงานออกเป็นแต่ละโมดูล แล้วนำมาใช้เฉพาะส่วนที่จำเป็น ทำให้โค้ดที่ต้องประมวลผลมีขนาดเล็กลง





ติดตั้ง Nodejs และ Visual Studio Code



<https://www.youtube.com/c/KongRuksiamOfficial/>



<https://www.facebook.com/KongRuksiamTutorial/>

เครื่องมือพื้นฐาน

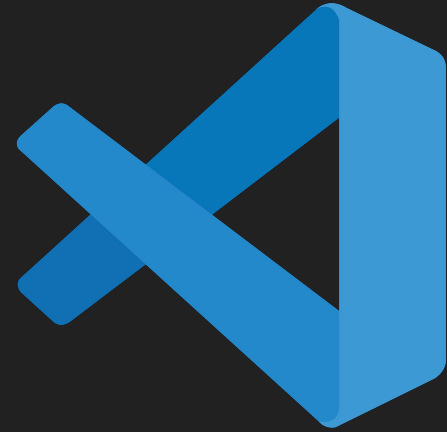


- Visual Studio Code
- Nodejs
- MongoDB Compass



Visual Studio Code : Extension

- Live Server
- Code Runner





เริ่มต้นใช้งาน Node.js



รู้จักกับ Non-Blocking I/O



<https://www.youtube.com/c/KongRuksiamOfficial/>



<https://www.facebook.com/KongRuksiamTutorial/>

ข้อดีของ Node.js



- Nodejs ใช้ JavaScript ในการพัฒนาเว็บทั้งฝั่ง Frontend และ Backend
- ทำงานแบบ Non-Blocking I/O โดยใช้วิธีการแบบ Asynchronous โดยจะไม่รอการตอบสนองแต่ละ Request ให้แล้วเสร็จ แต่จะทำการย้ายการทำงานไปอยู่เบื้องหลัง แล้วรอรับ Request ต่อๆ ไปทันที
- แบ่งการทำงานออกเป็นแต่ละโมดูล แล้วนำมาใช้เฉพาะส่วนที่จำเป็น ทำให้โค้ดที่ต้องประมวลผลมีขนาดเล็กลง



ข้อดีของ Node.js



- Nodejs ใช้ JavaScript ในการพัฒนาเว็บทั้งฝั่ง Frontend และ Backend
- ทำงานแบบ Non-Blocking I/O โดยใช้วิธีการแบบ Asynchronous โดยจะไม่รอการตอบสนองแต่ละ Request ให้แล้วเสร็จ แต่จะทำการย้ายการทำงานไปอยู่เบื้องหลัง แล้วรอรับ Request ต่อๆ ไปทันที
- แบ่งการทำงานออกเป็นแต่ละโมดูล แล้วนำมาใช้เฉพาะส่วนที่จำเป็น ทำให้โค้ดที่ต้องประมวลผลมีขนาดเล็กลง



Non-Blocking I/O

โดยทั่วไปกระบวนการทำงานภายในโปรแกรมจะเป็นไปตามลำดับ
ขั้นตอนที่กำหนดไว้ หมายถึง **ต้องทำงานส่วนนี้ให้เสร็จก่อนถึงจะไปทำงาน
ถัดไปได้ (Blocking)** แต่การทำงานบางอย่างอาจจะต้องใช้เวลาต้องรอ
จนกว่างานนั้นจะเสร็จ ถึงจะเริ่มต้นทำงานถัดไป ซึ่งทำให้เกิดความล่าช้า
ในการทำงาน ใน JavaScript จะจัดการปัญหาล่าช้าดังกล่าวโดยวิธีการทำ
งานที่เรียกว่า

Non-Blocking หรือ Asynchronous



Non-Blocking I/O

Non-Blocking หรือ Asynchronous

หมายถึง เวลาที่ระบบเริ่มต้นกระบวนการประมวลผลหรือทำงานไปแล้ว
ถ้างานนั้นยังทำไม่เสร็จหรือรอแล้วใช้เวลานาน จะข้ามไปทำงานถัดไป
ได้เลย แล้วค่อยกลับมาทำส่วนที่รอในภายหลัง วิธีนี้จะช่วยให้การทำงานมีความ
รวดเร็วมากยิ่งขึ้น เพราะไม่ต้องเสียเวลารอในบางขั้นตอน

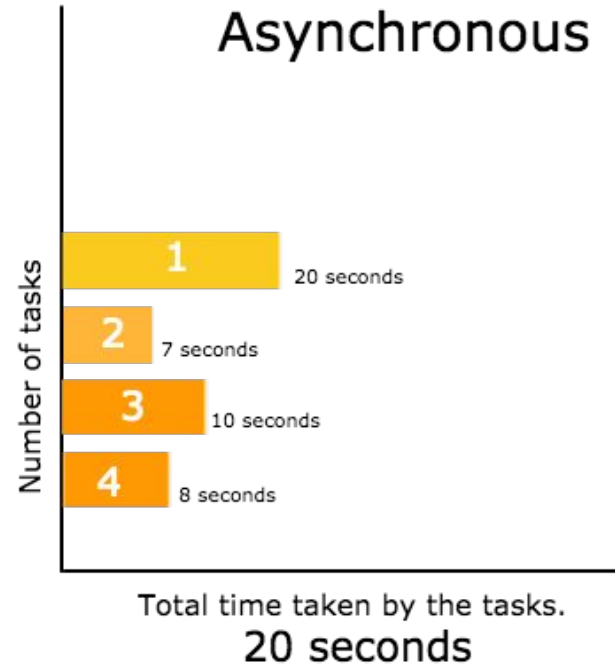
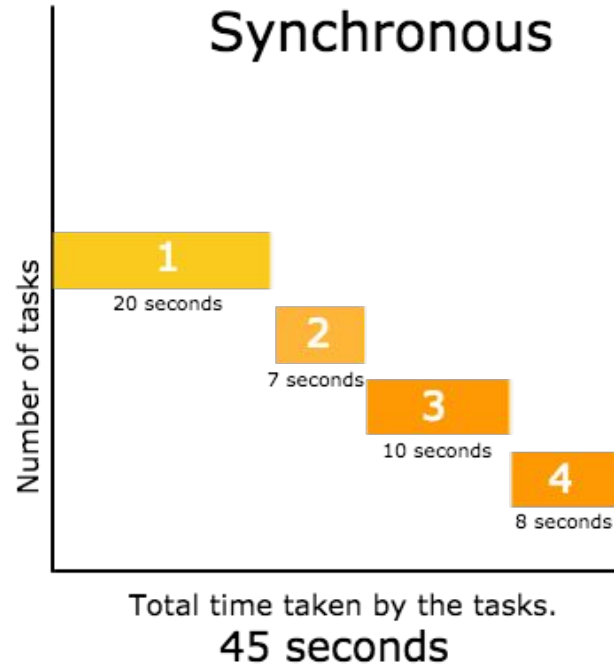


Blocking และ Non-Blocking

Blocking หรือ Synchronous คือ การทำงานแบบตามลำดับ สมมุติมีงาน A และงาน B โดยจะให้เริ่มทำงานใน A ให้เสร็จก่อนจนกระทั่งงาน A นั้นเสร็จสิ้นถึงจะทำงาน B หรืองานอื่นต่อได้ สรุปคือ ต้องรอทำงานคำสั่งก่อนหน้าให้เสร็จแล้วค่อยทำคำสั่งถัดๆไป

Non-Blocking หรือ Asynchronous การทำงานที่สามารถจะสลับไปทำงานอื่นได้ เช่น สามารถทำงาน A และ B ไปพร้อมๆกันโดยที่ไม่ต้องรอให้อีกงานเสร็จ (Node.js)





Non-Blocking หรือ Asynchronous ในชีวิตประจำวัน

- **ทานข้าว** - เราสามารถที่จะทานข้าวไปพร้อมๆกับดูคลิปในยูทูปได้ในเวลาเดียวกัน
- **สั่งอาหาร** - เราสามารถสั่งให้คนมาส่งอาหารให้เราได้พร้อมๆกับนั่งเขียนโค้ดไปด้วยในเวลาเดียวกัน ระหว่างรอคนส่งของมาถึง งานเราก็ใกล้จะเสร็จพอดีในเวลาเดียวกัน
- **โปรแกรมแชท** - เราสามารถที่จะอัปโหลดและส่งภาพในช่องแชทได้ ระหว่างที่รออัปโหลดและส่งภาพไปเราก็ยังสามารถพิมพ์แชทได้ในเวลาเดียวกัน โดยที่ไม่ต้องรอให้อัปโหลดภาพและส่งภาพเสร็จค่อยแชทได้

Blocking

เป็นการทำงานที่เกิดการรอเกิดขึ้น ไม่ว่าจะมีการทำงานหลายๆงานพร้อมกัน
ต้องรอทำงานเก่าจนเสร็จก่อนแล้วค่อยไปทำงานใหม่ลำดับถัดไป

ข้อดี

- มีการทำงานที่เป็นลำดับขั้นตอนและเข้าใจง่าย

ข้อเสีย

- สิ้นเปลืองทรัพยากร



Non - Blocking

เป็นการทำงานโดยที่ไม่ต้องรอการทำงานของงานก่อนหน้าเสร็จ ก็สามารถไปทำงานอื่นๆได้เลยทันที

ข้อดี

- ทำงานเร็วเพราะไม่ต้องรอทำงานตามลำดับ

ข้อเสีย

- การทำงานไม่เป็นลำดับอาจจะมีความเข้าใจยาก





แนวคิดและการทำ งานของ Node.js



<https://www.youtube.com/c/KongRuksiamOfficial/>



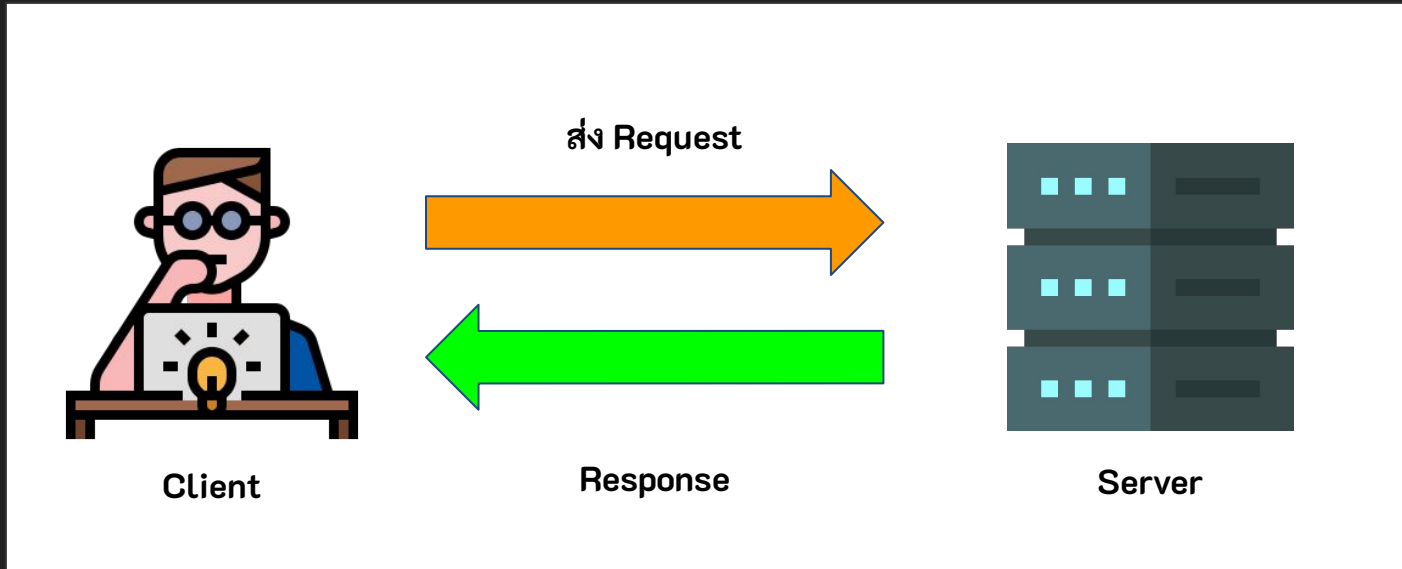
<https://www.facebook.com/KongRuksiamTutorial/>

คุณสมบัติของ Node.js

- ใช้ Google V8 Engine ในการ Compile Javascript
- ทำงานแบบ Single-Thread (1 Thread : 1 Process)
- ทำงานแบบ Non-Blocking I/O , Event Loop



แผนภาพการทำงาน

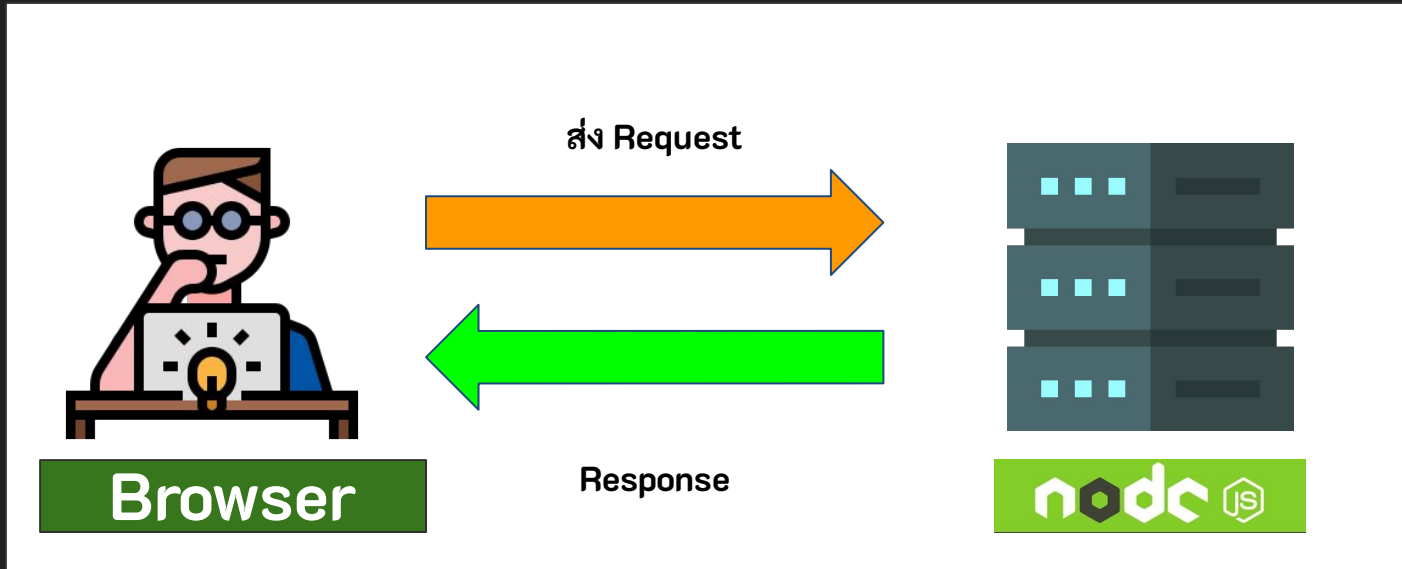


คำศัพท์พื้นฐาน

- Server - ผู้ให้บริการ
- Client - ผู้ใช้บริการ (User/Browser)
- Request - คำขอในการเข้าถึง
- Response - ตอบกลับคำขอ



แผนภาพการทำงาน



รูปแบบการทำงานของ Node.js



A diagram consisting of a light gray rounded rectangle containing a green square. Inside the green square, the text 'Single-Thread' and '(โค้ดที่ทำงานด้านใน)' is written in white.

Single-Thread
(โค้ดที่ทำงานด้านใน)

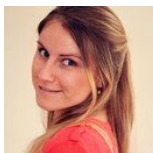


<https://www.youtube.com/c/KongRuksiamOfficial/>



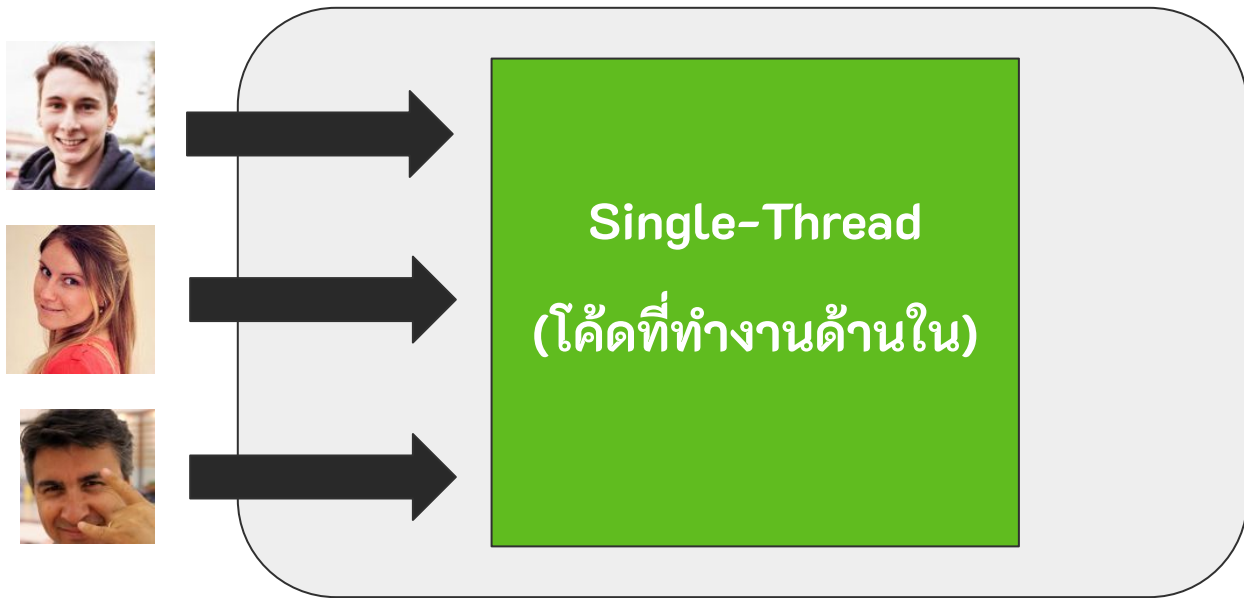
<https://www.facebook.com/KongRuksiamTutorial/>

รูปแบบการทำงานของ Node.js



Single-Thread
(โค้ดที่ทำงานด้านใน)

รูปแบบการทำงานของ Node.js

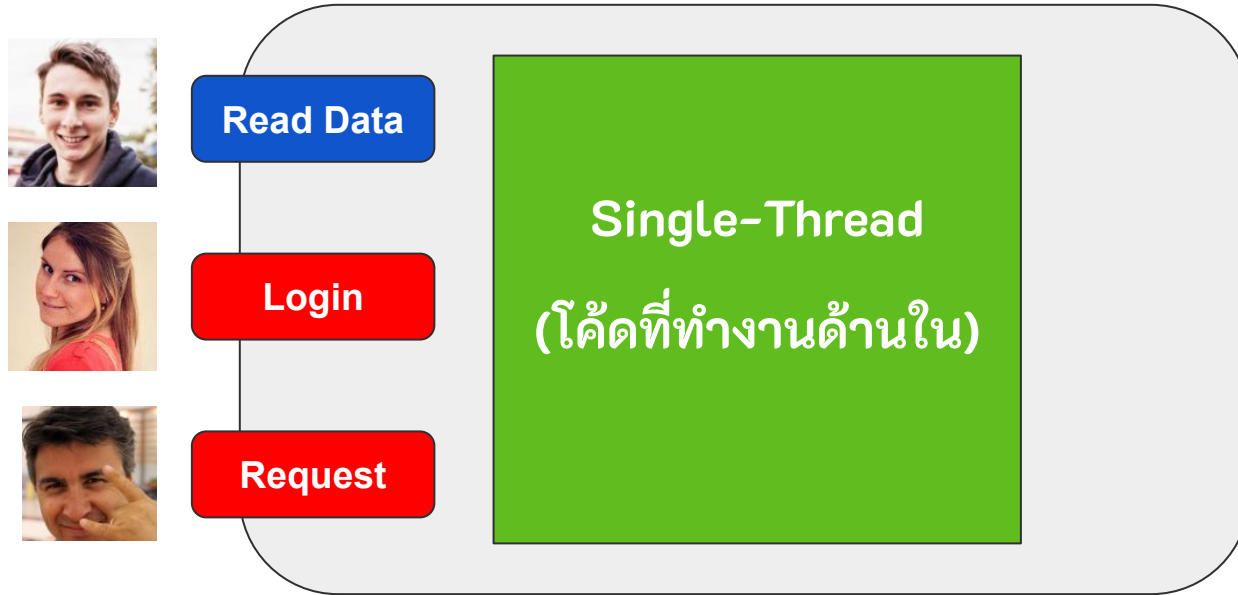


รูปแบบการทำงานของ Node.js



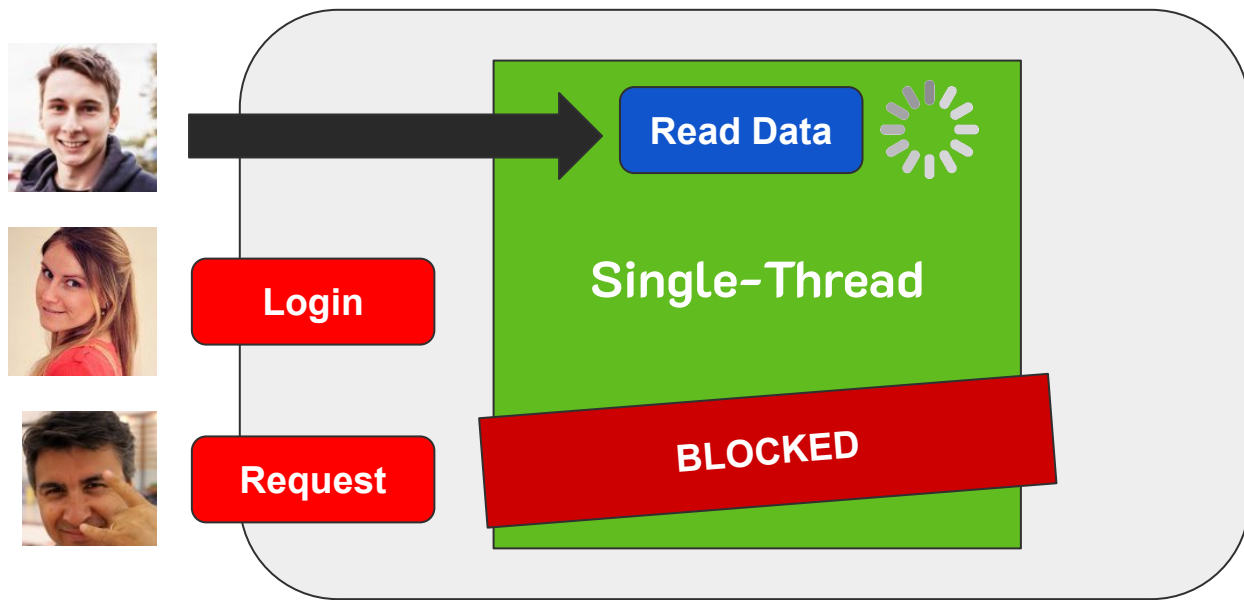
ใช้ Blocking Model

รูปแบบการทำงานของ Node.js



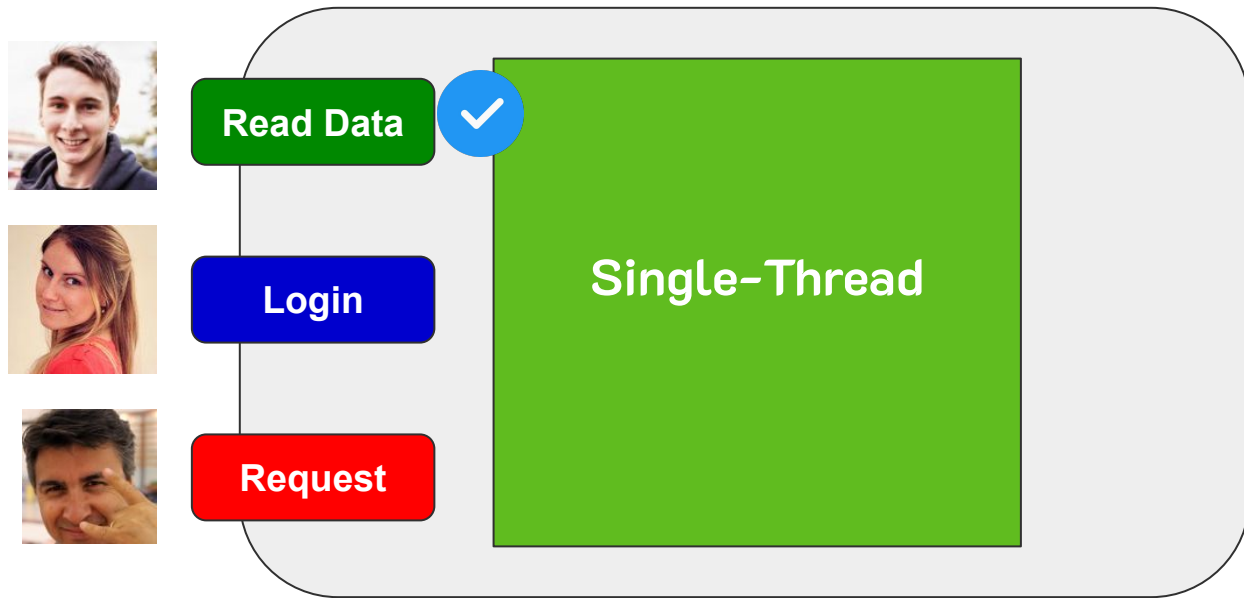
ใช้ Blocking Model

รูปแบบการทำงานของ Node.js



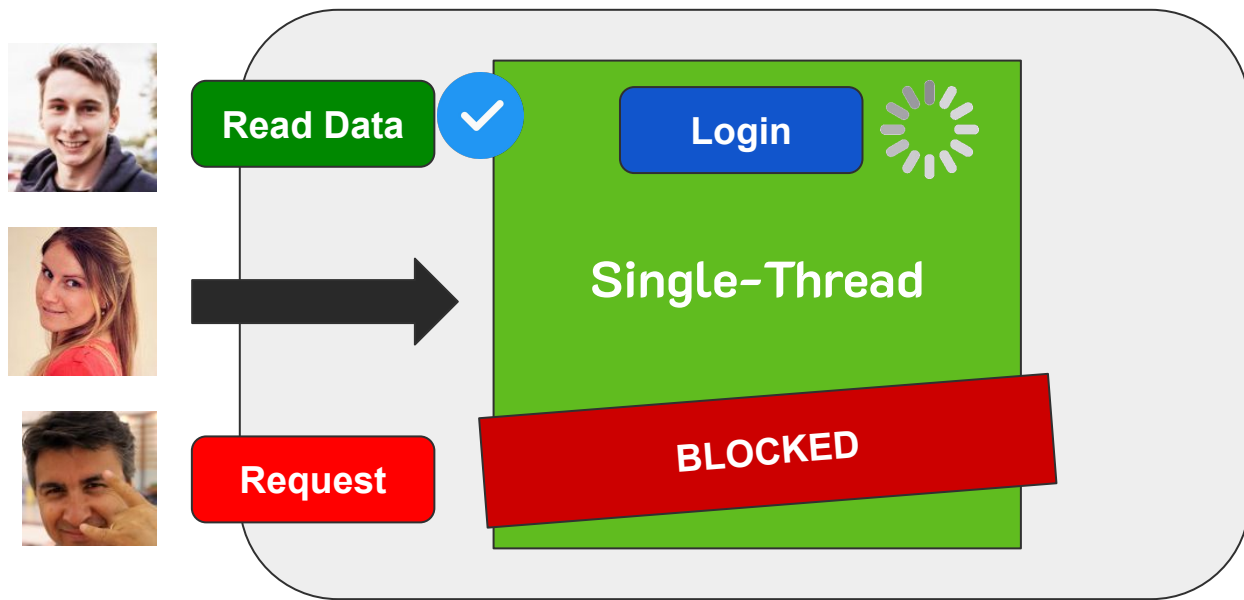
ใช้ Blocking Model

รูปแบบการทำงานของ Node.js



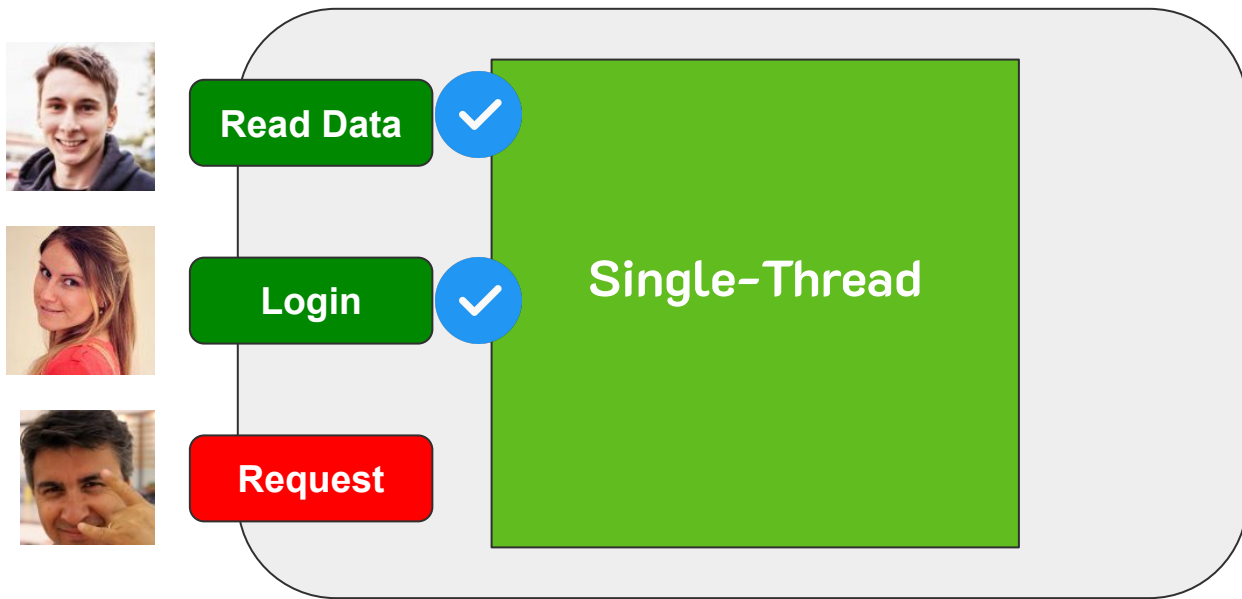
ใช้ Blocking Model

รูปแบบการทำงานของ Node.js



ใช้ Blocking Model

รูปแบบการทำงานของ Node.js



ใช้ Blocking Model

รูปแบบการทำงานของ Node.js



Single-Thread
Event - Loop

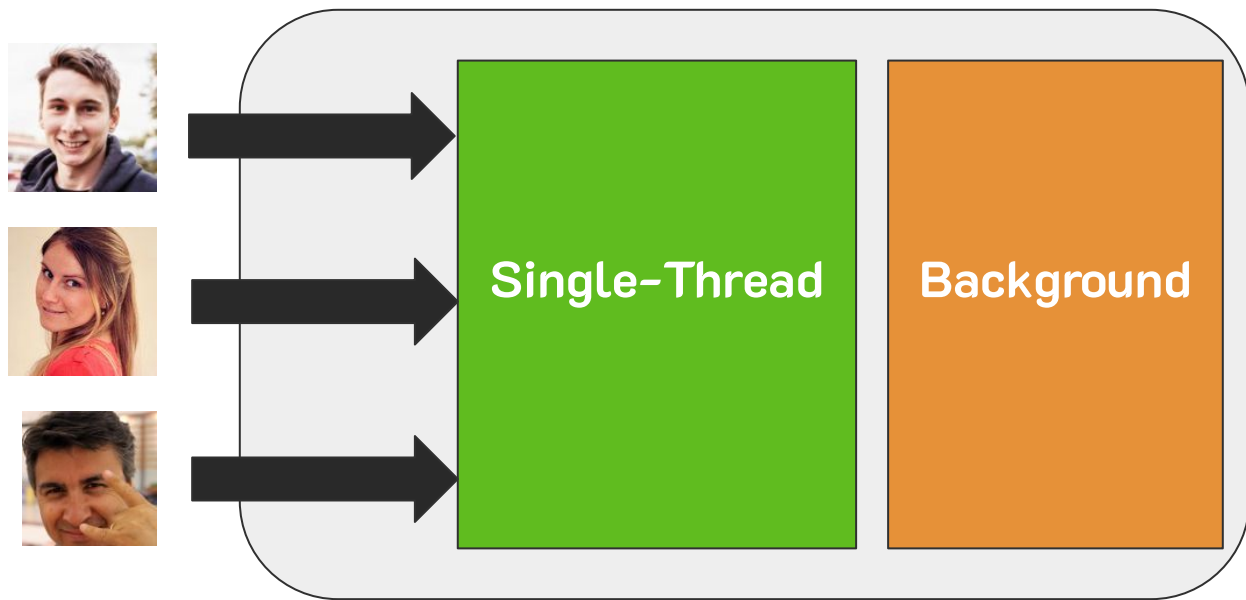
Background
(งานที่ต้องใช้เวลาทำนาน)

Non-Blocking Model

Event Loop คืออะไร

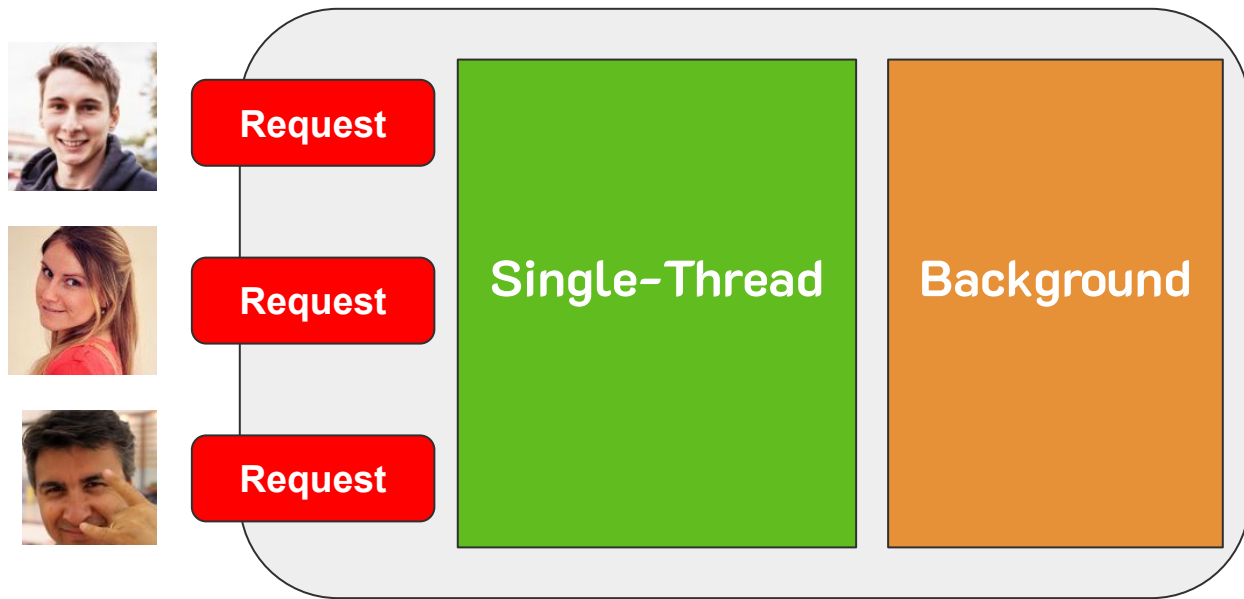
- เป็นรูปแบบทำงานแบบ Asynchronous เพื่อแก้ปัญหา Blocking I/O
- ใช้สำหรับวน Loop เพื่อตรวจสอบงานที่จัดลำดับใน Thread ว่ามีงานใน Thread ให้ทำหรือไม่ ?
- เมื่อเจองานที่เป็น Blocking I/O จะส่งงานดังกล่าวไปยัง Background (Thread Pool : Thread ย่อย) ทันทีและไม่ต้องรอผลลัพธ์ จากนั้นก็วนลูปใหม่เพื่อมองหาหรือรับงานลำดับถัดไป

รูปแบบการทำงานของ Node.js



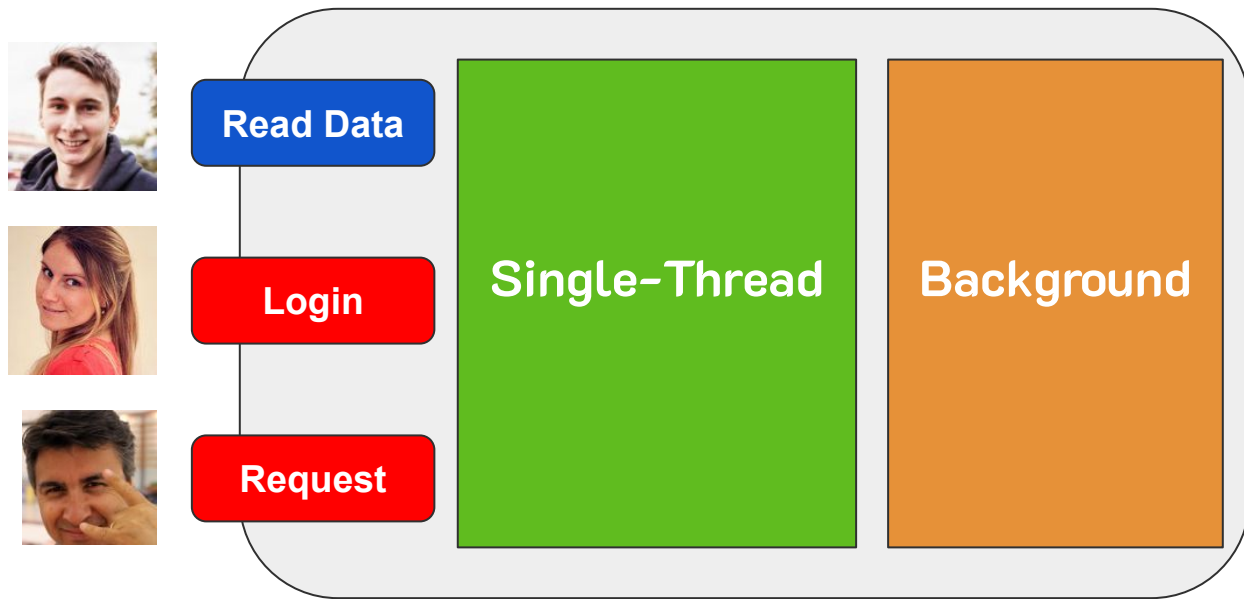
Non-Blocking Model

รูปแบบการทำงานของ Node.js



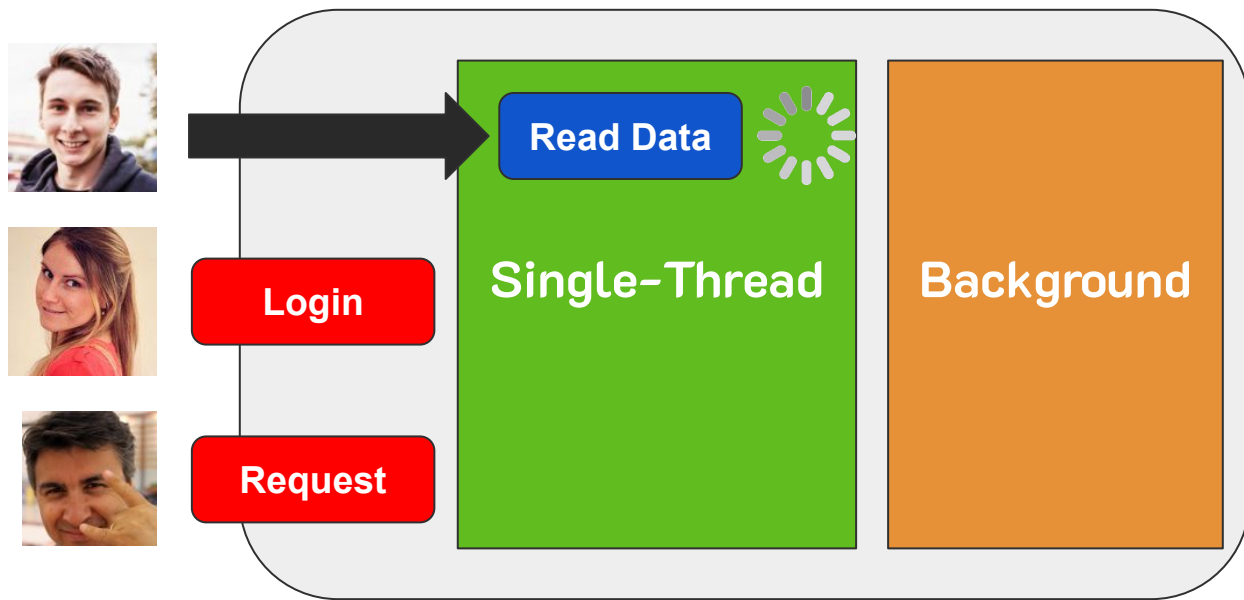
Non-Blocking Model

รูปแบบการทำงานของ Node.js



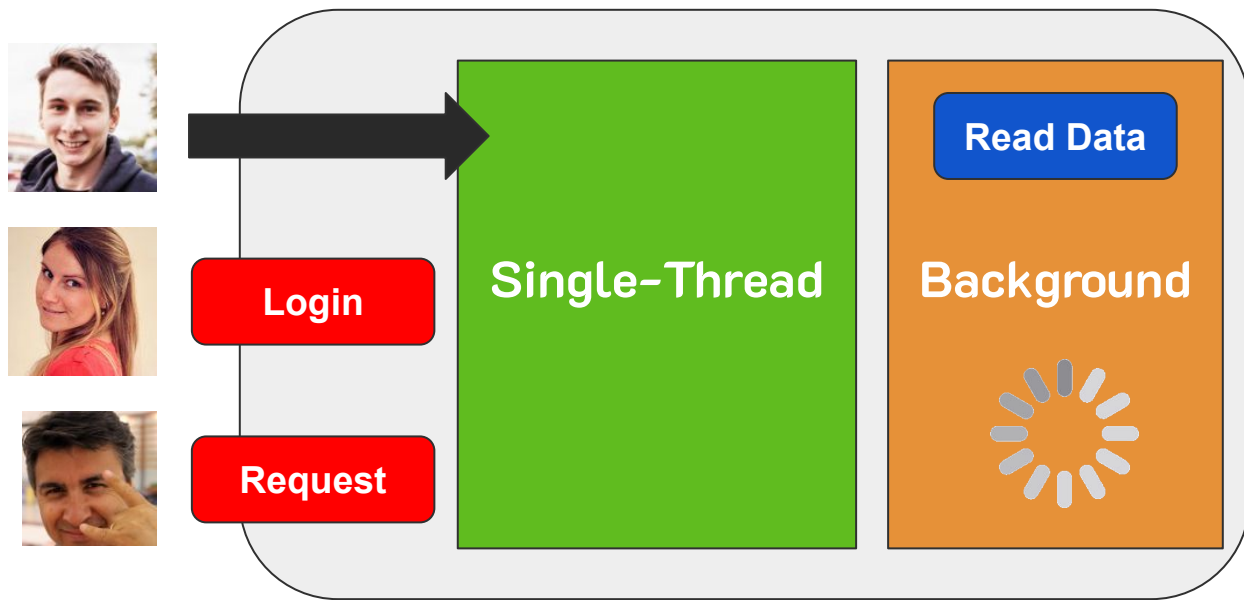
Non-Blocking Model

รูปแบบการทำงานของ Node.js



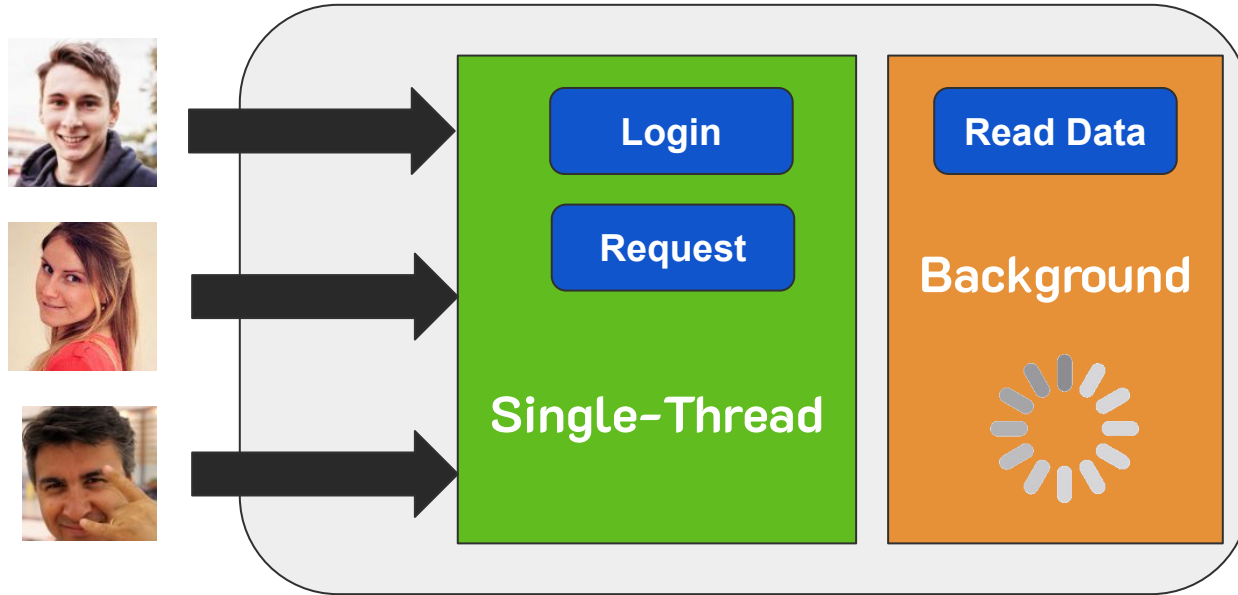
Non-Blocking Model

รูปแบบการทำงานของ Node.js



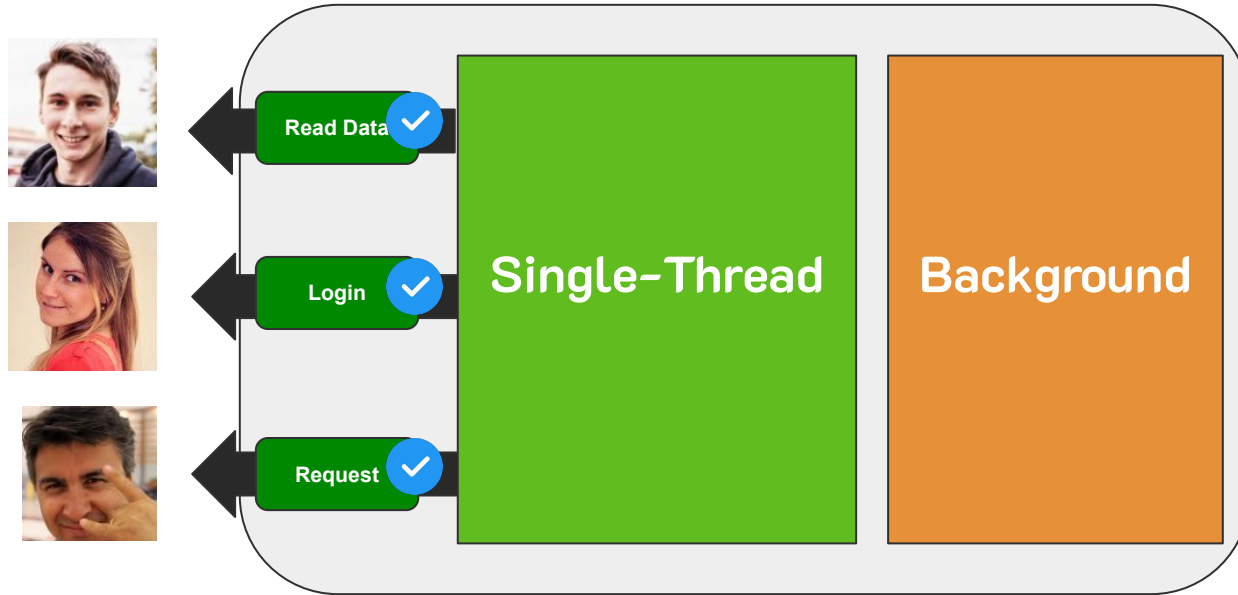
Non-Blocking Model

รูปแบบการทำงานของ Node.js



Non-Blocking Model

รูปแบบการทำงานของ Node.js



Non-Blocking Model



CallBack,Promise, Async-Await



ใน JavaScript จะมีการทำงานลักษณะแบบ **Non-Blocking** หรือ **Asynchronous** คือ การทำงานแบบไม่พร้อมกันหรือไม่ต่อเนื่องกันโดยงานบางส่วนที่ต้องใช้เวลาจะถูกทำงานในเบื้องหลังส่วนงานอื่นๆที่ไม่ต้องรอเวลาจะสามารถทำงานล่วงหน้าไปก่อนได้เลย และทางในกลับกันการหยุดรอจนกว่าการทำงานนั้นจะเสร็จสมบูรณ์ จะเรียกว่า **Blocking** หรือ **Synchronous**



คำสั่ง Asynchronous ใน JavaScript

- setTimeout
- setInterval
- Promise

คำสั่ง Synchronous ใน JavaScript

- confirm

เป็นผลมาจาก Browser รูปแบบเดิม



ตัวอย่างโค้ด

```
console.log("เริ่มต้น")  
console.log("ดาวน์โหลด")  
console.log("จบการทำงาน")
```

ตัวอย่างโค้ด

```
console.log("เริ่มต้น")  
setTimeout(()=>{  
  console.log("ดาวน์โหลด")  
},3000)  
console.log("จบการทำงาน")
```

กระบวนการทำงาน





ถ้าต้องการควบคุมการทำงาน
ที่เป็นแบบ Asynchronous
ให้ทำงานตามลำดับที่ต้องการ
จะทำอย่างไร ?



จัดการการทำงานแบบ Asynchronous จะมีหลายรูปแบบ

- Callback
- Promise
- Async & Await

Callback -> Promise -> Async & Await



จัดการการทำงานแบบ Asynchronous จะมีหลายรูปแบบ

- **CallBack**
- Promise
- Async & Await

CallBack -> Promise -> Async & Await



รู้จักกับ Callback



CallBack

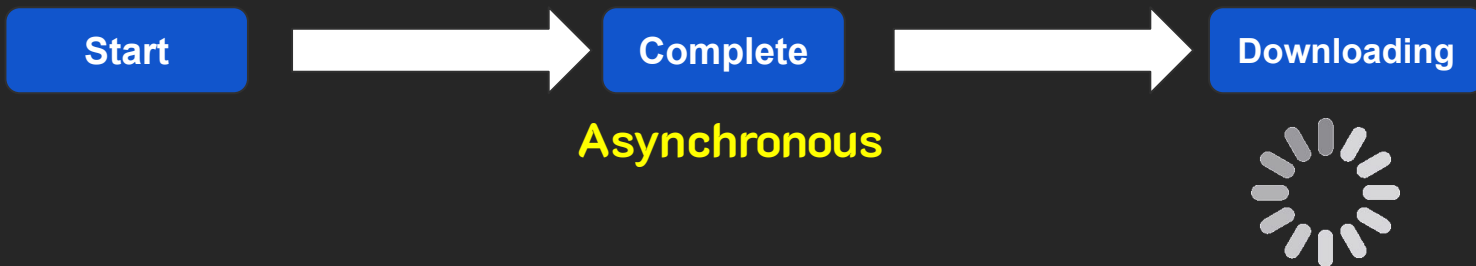
อาศัย CallBack Function คือ ฟังก์ชันที่จะถูกเรียกใช้
งานเมื่ออีกฟังก์ชันทำงานเสร็จ





CallBack

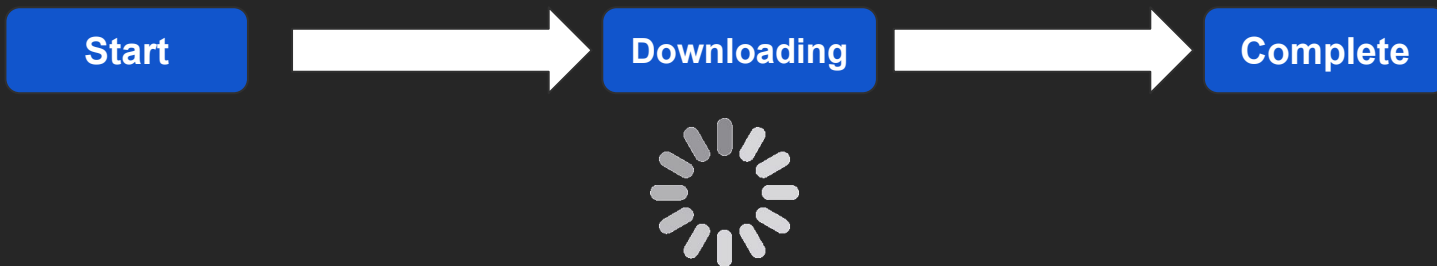
อาศัย CallBack Function คือ ฟังก์ชันที่จะถูกเรียกใช้
งานเมื่ออีกฟังก์ชันทำงานเสร็จ





CallBack

อาศัย CallBack Function คือ ฟังก์ชันที่จะถูกเรียกใช้
งานเมื่ออีกฟังก์ชันทำงานเสร็จ

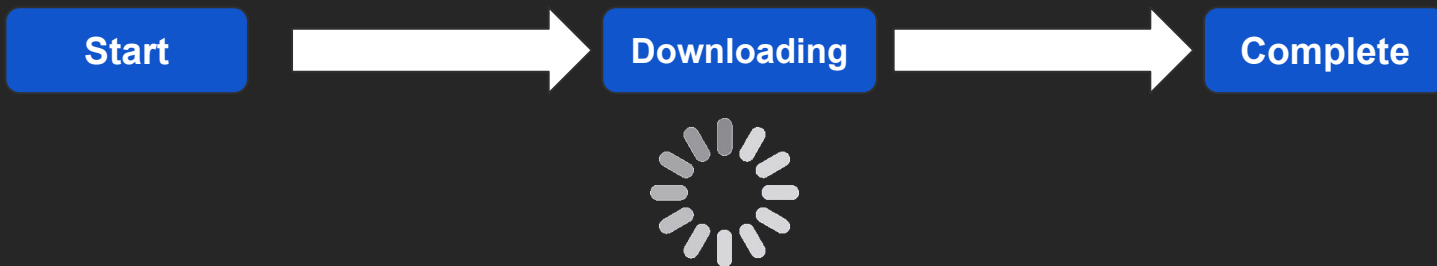




CallBack

อาศัย CallBack Function คือ ฟังก์ชันที่จะถูกเรียกใช้
งานเมื่ออีกฟังก์ชันทำงานเสร็จ

CallBack มาควบคุมลำดับการทำงาน





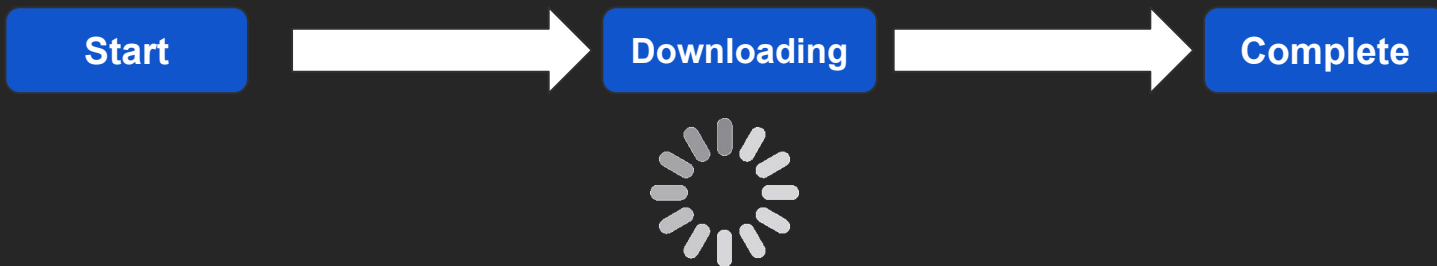
ทบทวน Callback

เขียนโค้ดโปรแกรมบวกเลขแบบปกติและลดรูป



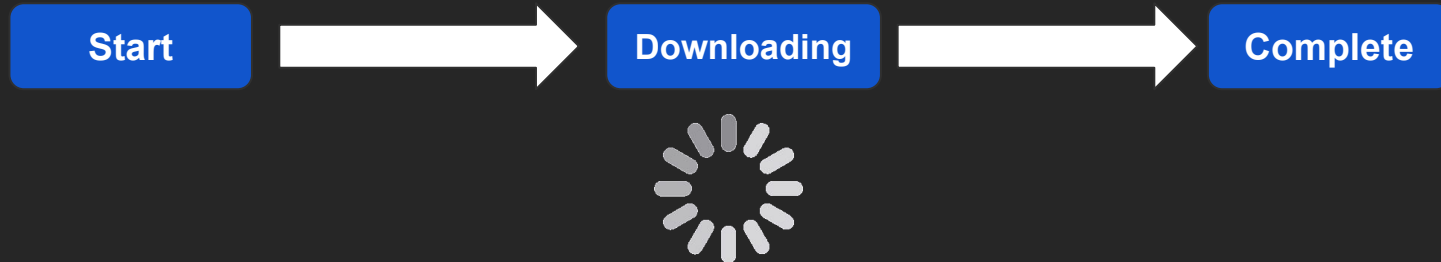
1. ใช้ Callback มาควบคุมการทำงานแบบ Asynchronous ให้คำสั่งทำงานตามลำดับที่ต้องการ

ส่ง url ของไฟล์เป็นพารามิเตอร์เข้าไปทำงาน





กำหนด url ไฟล์ตัวอย่างให้กับการ Download ไฟล์ในฟังก์ชัน
โดยการโหลดไฟล์ตั้งแต่ไฟล์ที่ 1 ไปจนถึงไฟล์ที่ 5 เรียงลำดับ





Callback Hell

การใช้ Callback อาจก่อให้เกิดปัญหาที่เรียกว่า
Callback Hell ซึ่งก็คือการนำฟังก์ชัน Callback มา
ซ้อนกันไปเรื่อยๆ แล้วส่งผลทำให้เกิดความสับสนกับการ
เช็คค่าลำดับการทำงานเลยเกิดวิธีแก้ปัญหาดังกล่าว
โดยการใช้ **“Promise”**



รู้จักกับ Promise



Promise (พรอมิส)

Promise ถูกนำมาใช้งานเกี่ยวกับการทำงานแบบ
Asynchronous คือ ให้รอในระหว่างที่ผลลัพธ์ยังไม่เกิดขึ้น
ใช้กับงานที่มีลักษณะการหน่วงเวลา (Delay) หรืองานที่ต้อง
ทำเบื้องหลัง แล้วจะมีผลเกิดขึ้นในเวลาต่อมาและถูกนำมาใช้
แก้ปัญหา CallBack Hell



```
Promise(function(resolve,reject){
```

การทำงานใน Promise จะมี 3 สถานะ คือ pending, resolve, reject



Promise (พรอมิส)

- **pending** เป็นสถานะเริ่มต้นของ Promise
 - ถ้าทำงานสำเร็จจะเป็น resolve
 - ถ้าล้มเหลวจะเป็น reject
- **resolve/fulfilled** เป็นพารามิเตอร์ของ callback ซึ่งใช้กำหนดสถานะหากทำงาน “สำเร็จ”
- **reject** เป็นพารามิเตอร์ของ callback ซึ่งใช้กำหนดสถานะหากทำงาน “ผิดพลาด”



Promise (พรอมิส)

```
let connect = true  
const downloading = new Promise(function (resolve, reject) {  
  if (connect) {  
    resolve("ดาวน์โหลดเสร็จเรียบร้อยแล้ว");  
  } else {  
    reject("เกิดข้อผิดพลาดระหว่าง Download");  
  }  
});
```



Promise (พรอมิส)

```
let connect = true  
const downloading = new Promise(function (resolve, reject) {  
  setTimeout(()=>{  
    if (connect) {  
      resolve("ดาวน์โหลดเสร็จเรียบร้อยแล้ว");  
    } else {  
      reject("เกิดข้อผิดพลาดระหว่าง Download");  
    }  
  },3000)  
});
```

เมธอด `then()` , `catch()` , `finally()`



การทำงานของ Promise ระหว่างที่ตรวจสอบสถานะของ Promise อยู่ว่าเป็น resolve หรือ reject สามารถกำหนดขั้นตอนต่อไปในการทำงานได้ โดยอาศัย `then()` , `catch()` มาใช้ตอบสนองสถานะดังกล่าว

เมธอด `then()` , `catch()` , `finally()`



- `then()` ใช้งานร่วมกับสถานะ resolve หรือเมื่อ Promise ทำงานสำเร็จ
- `catch()` ใช้งานร่วมกับสถานะ reject หรือเมื่อ Promise ทำงานผิดพลาด
- `finally()` ไม่ว่าผลลัพธ์ของสถานะจะเป็นอย่างไรให้ทำงานต่อส่วนนี้ได้เลย

เมธอด then() , catch() , finally()



```
let connect= true

const downloading = new Promise(function (resolve, reject) {
  //
});

downloading.then(result=>{

})

downloading.catch(result=>{

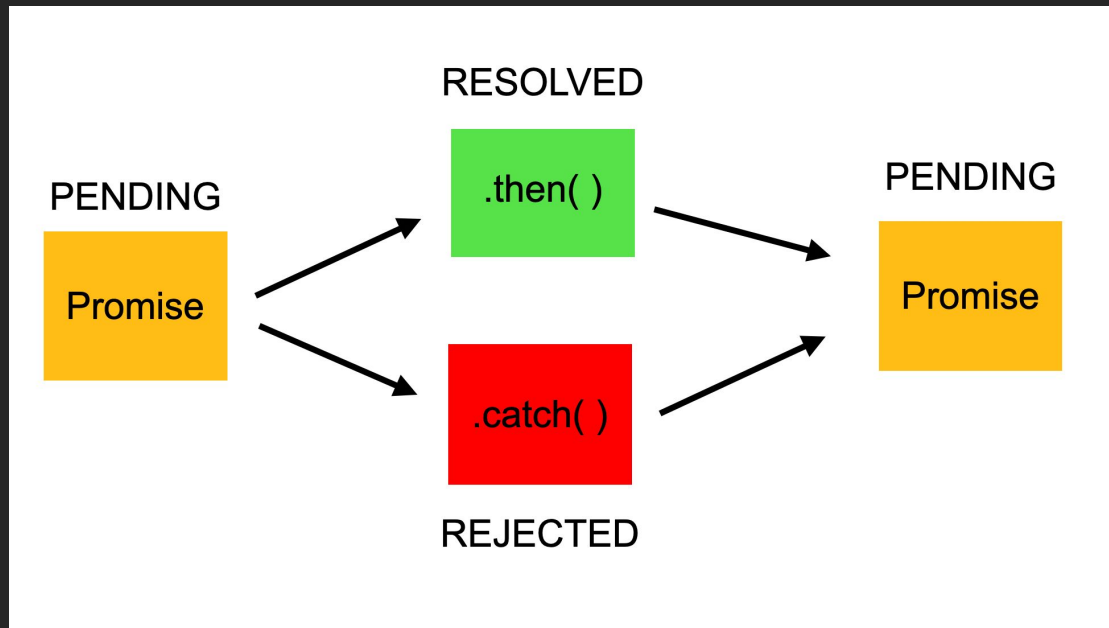
})
```


เมธอด then() , catch() , finally()



```
let connect = true  
  
const downloading = new Promise(function (resolve, reject)  
{  
  //  
})  
  
.then(result=>{})  
  
.catch(result=>{});
```

สรุป Promise (พรอมิส)



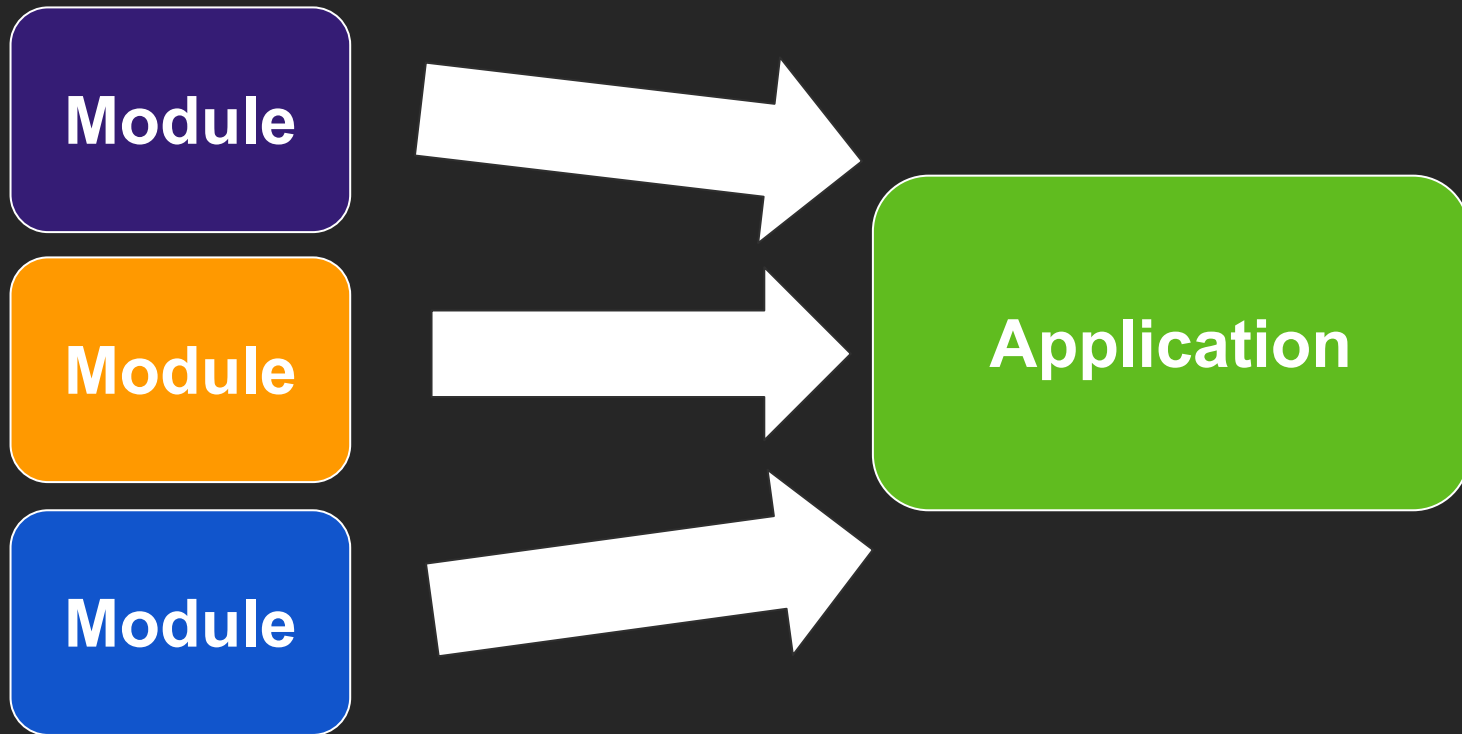
โมดูล (Module)



Module คือ ไฟล์ที่จัดเก็บโค้ดของ JavaScript ซึ่งประกอบด้วยการทำงานต่างๆ ได้แก่ ตัวแปร ฟังก์ชัน , Class , Object หรืออื่นๆ เพื่อนำไปใช้งานในส่วนต่างๆของโปรเจค

ในปัจจุบันมีผู้สร้าง Module สำหรับสนับสนุนและให้บริการอยู่หลายแบบเพื่ออำนวยความสะดวกเกี่ยวกับงานแต่ละด้าน เช่น จัดการ Request , จัดการ Database , Image , DateTime , Validate เป็นต้น

โมดูล (Module)



การสร้างและส่งออกโมดูล (Module)



ชื่อโมดูล.js

```
const PI = 3.14
```

```
function add(x,y){
```

```
    return x+y
```

```
}
```

```
module.exports.PI=PI
```

```
module.exports.add = add
```

การนำโมดูลมาใช้งาน



ตัวแปรรับค่าโมดูล = `require (<location>)`

A diagram consisting of a purple rectangle with a white outline. Inside the rectangle, the text "ตำแหน่งของโมดูล" is written in white. A white arrow points upwards from the top center of the rectangle towards the text in the box above.

ตำแหน่งของโมดูล

```
const util = require('mymodule')
```



อ่านและเขียนไฟล์ ด้วยโมดูล fs (File System)

การอ่านและเขียนไฟล์



การอ่านและเขียนไฟล์จะใช้โมดูลชื่อว่า **fs (File System)**

โดยแบ่งการทำงานออกเป็น 2 รูปแบบ

- อ่านและเขียนไฟล์แบบ Synchronous (Blocking)
- อ่านและเขียนไฟล์แบบ Asynchronous (Non-Blocking)

การอ่านและเขียนไฟล์



อ่านไฟล์แบบ Synchronous (Blocking)

โครงสร้างคำสั่ง

```
const data = fs.readFileSync('ตำแหน่งไฟล์', encoding)
```

******เก็บค่าที่ได้จากการอ่านไฟล์ไว้ในตัวแปร data

การอ่านและเขียนไฟล์



เขียนไฟล์แบบ Synchronous (Blocking)

โครงสร้างคำสั่ง

```
const data = "Hello World"
```

```
fs.writeFileSync('ตำแหน่งไฟล์', data)
```



<https://www.youtube.com/c/KongRuksiamOfficial/>



<https://www.facebook.com/KongRuksiamTutorial/>

การอ่านและเขียนไฟล์



การอ่านและเขียนไฟล์แบบ Synchronous (Blocking)

```
const data = fs.readFileSync('ตำแหน่งไฟล์', encoding)
fs.writeFileSync('ตำแหน่งไฟล์', data)
console.log("เขียนไฟล์สำเร็จ!")
```



<https://www.youtube.com/c/KongRuksiamOfficial/>



<https://www.facebook.com/KongRuksiamTutorial/>

การอ่านและเขียนไฟล์



จากโค้ดดังกล่าว ถ้ายังอ่านไฟล์ไม่เสร็จจะไปทำงานในส่วนของการเขียนไฟล์ต่อไม่ได้ เนื่องจากเป็นรูปแบบ Synchronous แล้วถ้าไฟล์มีขนาดใหญ่ก็ต้องใช้เวลาอ่านไฟล์นานและการทำงานของระบบก็จะช้าไปด้วย สามารถจัดการปัญหาดังกล่าวโดยใช้รูปแบบ **Asynchronous**



<https://www.youtube.com/c/KongRuksiamOfficial/>



<https://www.facebook.com/KongRuksiamTutorial/>

การอ่านและเขียนไฟล์



อ่านไฟล์แบบ Asynchronous (Non-Blocking)

โครงสร้างคำสั่ง

```
fs.readFile('ตำแหน่งไฟล์', encoding, callback)
```

```
fs.readFile('ตำแหน่งไฟล์', encoding, (err, data) {  
    console.log(data) // แสดงข้อมูลที่เก็บใน data  
})
```

การอ่านและเขียนไฟล์



เขียนไฟล์แบบ Asynchronous (Non-Blocking)

โครงสร้างคำสั่ง

```
const data = "Hello World"
```

```
fs.writeFile('ตำแหน่งไฟล์', data, callback)
```

การอ่านและเขียนไฟล์



อ่านและเขียนไฟล์แบบ Asynchronous (Non-Blocking)

```
fs.readFile('ตำแหน่งไฟล์', encoding, (err, data) {  
  fs.writeFile('ตำแหน่งไฟล์', data, (err) => {  
    if (err) return console.log(err)  
    console.log("เขียนไฟล์เรียบร้อยแล้ว")  
  })  
})
```

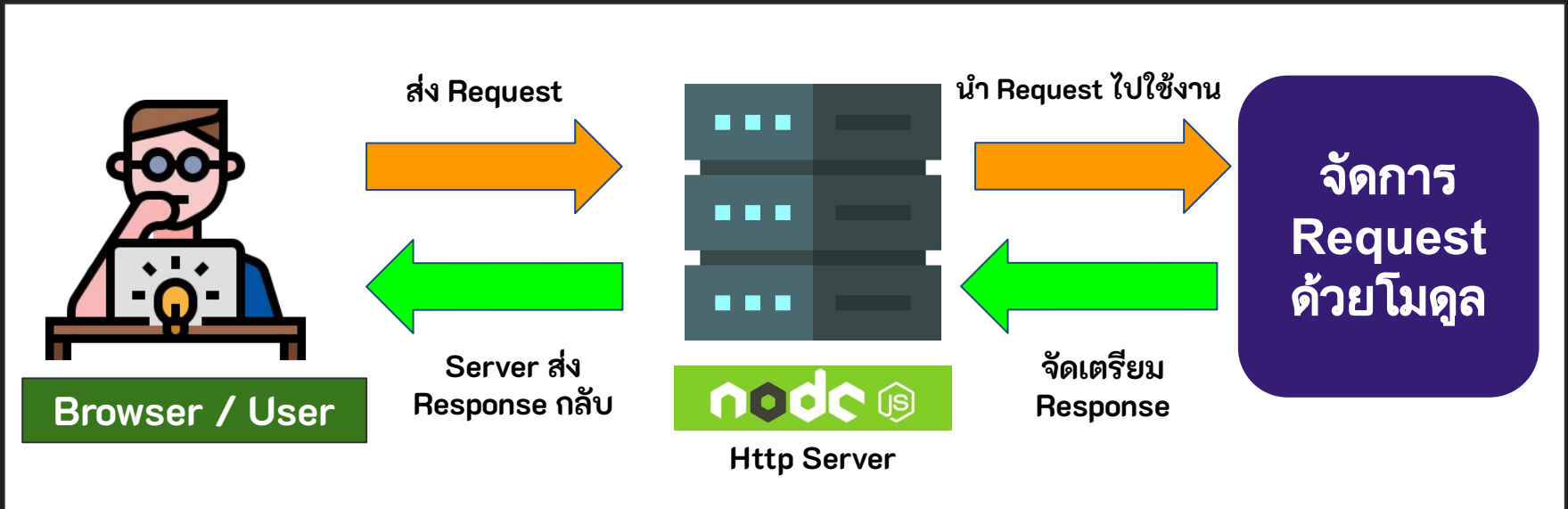
สร้าง Web Server ด้วย Node.js

คำศัพท์พื้นฐาน

- Server - ผู้ให้บริการ
- Client - ผู้ใช้บริการ (User/Browser)
- Request - คำขอในการเข้าถึง
- Response - ตอบกลับคำขอ



แผนภาพการทำงาน



สร้างแอปของ Node.js

- `npm init` หรือ `npm init -y` (ไม่ต้องใส่รายละเอียด)
- `package.json` - เป็นไฟล์ที่เก็บข้อมูลหรือรายละเอียดต่างๆ รวมถึง `module/package` ที่จะใช้ทำงานภายในโปรเจค

สร้าง Web Server ด้วย Node.js

http คือ โมดูลที่ใช้นำมาควบคุมการทำงานของ Server

http.createServer() คำสั่งสำหรับสร้าง Server โดยสร้างการเชื่อมต่อและรับส่งข้อมูลผ่าน Callback Function และรับค่าเข้ามาทำงาน 2 ค่า ได้แก่

- **req (request)** รับข้อมูลจากผู้ใช้ (Browser) มาที่ Server
- **res (response)** ส่งข้อมูลตอบกลับจาก Server ไปหาผู้ใช้

สร้าง Web Server ด้วย Node.js

response.write() เขียนผลลัพธ์ตอบกลับไปหาผู้ใช้ ระบุเป็นข้อความหรือ HTML ก็ได้ (เขียนหรือไม่เขียนก็ได้)

response.end() กำหนดจุดสิ้นสุดการรับส่งข้อมูลหรือระบุการตอบกลับไปหาผู้ใช้งาน

listen(3000) สั่งให้ Web Server เริ่มรันแล้วเชื่อมต่อที่ port หมายเลข 3000 หรือใช้ port หมายเลขอื่นก็ได้ เช่น 5000 , 8000 เป็นต้น

รันแอปด้วย Nodemon

ในกรณีที่มีการเปลี่ยนแปลงการทำงานในแอปที่พัฒนาด้วย Node.js ต้องทำการหยุดรันแอปโดยการกด Ctrl+C ทุกครั้งแล้วค่อยรันใหม่ ด้วยคำสั่ง `node index.js` จึงจะเห็นการเปลี่ยนแปลงที่เกิดขึ้น ซึ่งมันทำให้เกิดความยุ่งยากเพราะต้อง restart แอปทุกครั้ง ซึ่งสามารถจัดการปัญหาดังกล่าวได้โดยใช้

nodemon

รันแอปด้วย Nodemon

nodemon คือ โมดูลที่จะคอยติดตามการเปลี่ยนแปลงที่เกิดขึ้นกับไฟล์ JavaScript (.js) ทั้งหมดที่อยู่ในแอป เมื่อใดที่ไฟล์ดังกล่าวมีการเปลี่ยนแปลงและถูกบันทึกไฟล์เกิดขึ้น nodemon ก็จะ restart แอปให้โดยอัตโนมัติ โดยที่ไม่ต้อง restart แอปเอง

การติดตั้ง

```
npm install nodemon
```

รันแอปด้วย Nodemon

การรันแบบเดิม

```
node index.js
```

การรันแบบใหม่

```
npx nodemon index.js
```

ตั้งค่าใน script

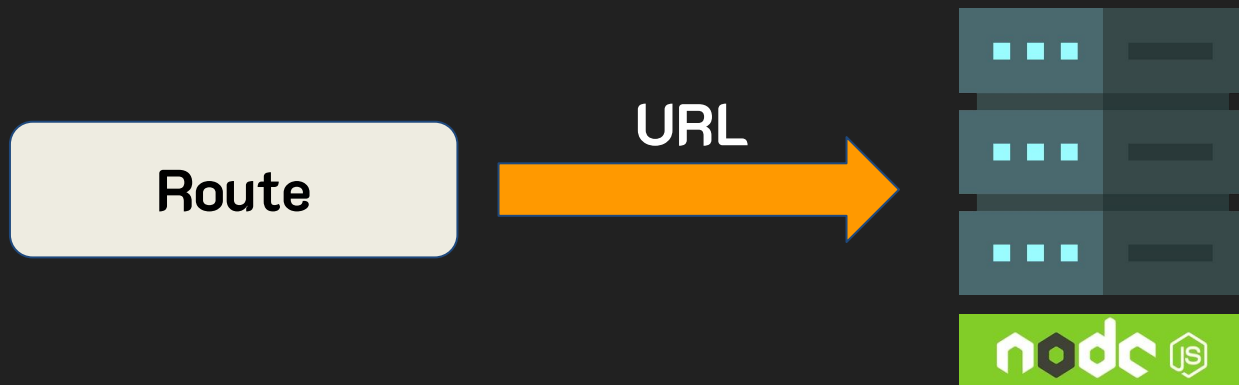
```
"start": "nodemon ./bin/www"
```

และรันโดยใช้

```
npm start
```



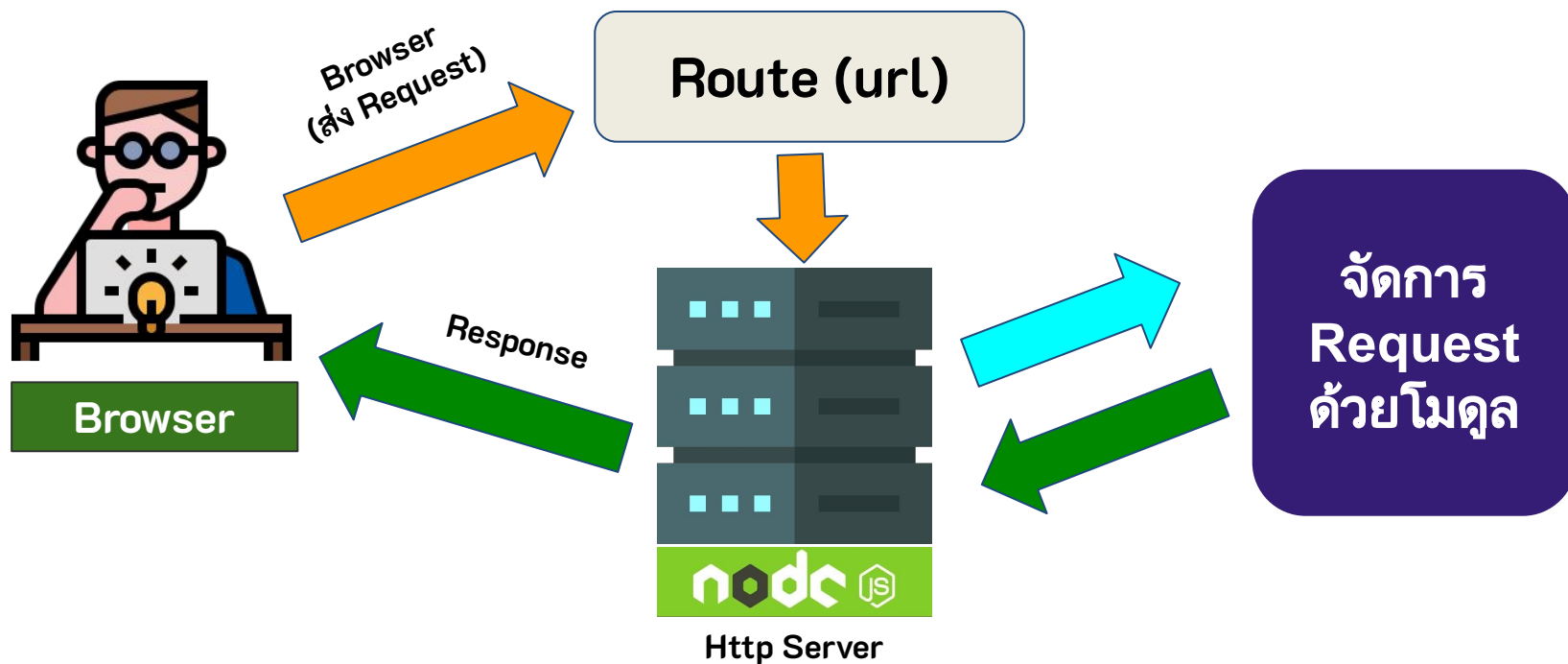
Routing



การกำหนดเส้นทางหรือ URL ในการอนุญาตให้เข้าถึงข้อมูล
รวมไปถึงตรวจสอบ URL Request เพื่อจะได้กำหนดรูปแบบการ
ทำงาน



แผนภาพการทำงาน



HTTP Status Code

เป็นรหัสที่บ่งบอกสถานะของ Request ตัวอย่างเช่น

- 200 Ok (ดำเนินการเสร็จสมบูรณ์)
- 201 Create (สร้างข้อมูลใหม่เรียบร้อยแล้ว)
- 400 Bad Request (Server ไม่เข้าใจว่า Request นี้เกี่ยวกับอะไร)
- 404 Not Found (หาข้อมูลที่เรียกไม่เจอหรือไม่สามารถใช้งานได้)
- 500 Internal Server Error (Request ถูกต้องแต่มีข้อผิดพลาดที่ฝั่ง Server)



Express

ปัญหาเมื่อใช้ Node.js ทำ Web Server

ในกรณีที่ใช้ Node.js ทำ Web ฟัง Server จะมีความยุ่งยากในการทำระบบหลายๆ อย่าง เช่น

ระบบ Routing จากหัวข้อที่ผ่านมาต้องอ่าน path จาก URL และนำค่ามาเปรียบเทียบกับเพื่อแสดงผลไฟล์หรือเพจที่ต้องการ ซึ่งวิธีดังกล่าวนี้ค่อนข้างมีความยุ่งยากและไม่เหมาะกับการใช้งานในระบบที่ใหญ่ มีความซับซ้อนสูง จึงได้มีการพัฒนา **Express.js** ขึ้นเพื่อ จัดการปัญหาการทำ Web Server ด้วย Node.js นั้นเอง

รู้จักกับ Express.js

Express.js เป็น Framework ของ JavaScript ที่ใช้งานร่วมกับ Node.js เพื่อสนับสนุนการทำงานของ Web Server ให้ความง่ายและสะดวกสบายมากยิ่งขึ้น เนื่องจากมี Feature หลากหลายที่น่าสนใจ เช่น Routing การจัดการ Request , Response และ Middleware เป็นต้น

ในปัจจุบันนิยมนำ Express.js มาสร้าง Web Server มากกว่าที่จะทำผ่าน Node.js โดยตรงแล้ว

The logo for Express.js, featuring the word "Express" in a white, sans-serif font inside a white rectangular box.

ติดตั้ง Express.js

คำสั่งที่ใช้

```
npm install express
```

การนำ express มาใช้งาน

```
const express = require('express')
```



การใช้งาน Express.js

```
const express = require('express') // นำ express เข้ามาทำงาน

const app = express() // เรียกใช้งาน express และเก็บลงในตัวแปร app

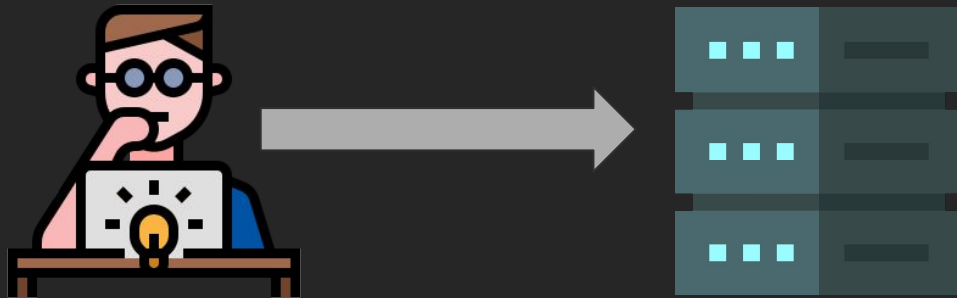
app.use((req,res)=>{ // path เริ่มต้น
    res.send("Hello Express.js")
})

app.listen(8080, () => { // รัน server ผ่าน port

    console.log('Start server at port 8080.')

})
```


Express & Routing เบื้องต้น



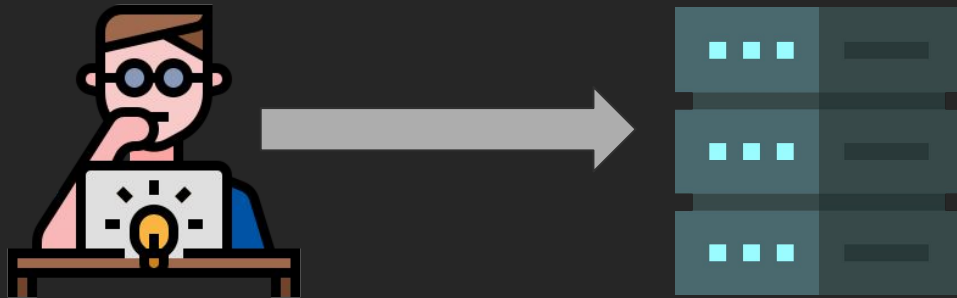
- ใช้คำสั่ง use (path เริ่มต้น)
- ใช้ http method (get ,post)

Express & Routing เบื้องต้น

ตัวอย่างคำสั่ง use

```
const express = require('express') // นำ express เข้ามาทำงาน  
const app = express() // เรียกใช้งาน express และเก็บลงในตัวแปร app  
app.use('/',(req,res)=>{ // ระบุ path เริ่มต้น  
    res.send("Hello Express.js")  
})
```

Express & Routing เบื้องต้น



โครงสร้างคำสั่ง

```
app.method(path, callback function)
```

Express & Routing เบื้องต้น

- **get()** เป็นเมธอดสำหรับกำหนดการทำงานตามเส้นทางที่ระบุ เมื่อ request ถูกส่งเข้ามาเช่น การส่งค่ามาพร้อมกับ URL เป็นต้น
- **callback function** คือ สำหรับกำหนดรูปแบบการตอบสนองที่เกิดขึ้นเพื่อส่ง request เข้ามาในเส้นทางดังกล่าว

Express & Routing เบื้องต้น

```
const express = require('express') // นำ express เข้ามาทำงาน

const app = express() // เรียกใช้งาน express และเก็บลงในตัวแปร app

app.get('/', (req, res) => { // ให้กำหนด url เป็น / ในรูปแบบของ get (http method)

    res.send('Hello Express.js') // response ข้อความไปหาผู้ใช้

})

app.get('/product', (req, res) => {

    res.send('Hello Product')

})
```

การใช้งานโมดูล path (20)

```
const path = require('path')

const indexPage = path.join(__dirname, 'index.html')

app.get('/', (req, res) => {

  res.status(200) // แจ้ง status code

  res.type('text/html') // กำหนดรูปแบบเนื้อหา

  res.sendFile(indexPage)

})
```

****ตรวจสอบเนื้อหา
ก่อนขึ้นหัวข้อใหม่**

รู้จักกับ Class Router

การทำระบบที่มีความซับซ้อนมากขึ้น มีเส้นทาง (route) มากขึ้น
การเขียนระบบเส้นทาง (routing) แบบเดิมอาจจะไม่ตอบโจทย์
ในหัวข้อนี้จะมาแนะนำการจัดการ Routing โดยใช้ Class ที่มีชื่อว่า

Router

รู้จักกับ Class Router

โครงสร้างคำสั่ง

```
const express = require('express')
```

```
const router = express.Router()
```

```
router.get('/', function(req, res) => {})
```

```
router.get('/product', function(req, res) => {})
```

```
app = express()
```

```
app.use(router) // นำ router ไปกำหนดเส้นทางในแอป
```

Router Parameter

การกำหนดพารามิเตอร์หรือตัวแปรส่งไปพร้อมกัน Path โดยใช้เครื่องหมาย : (colon) กำกับไว้ด้านหน้าชื่อพารามิเตอร์ ในแต่ละ Path สามารถกำหนดพารามิเตอร์ได้มากกว่า 1 ตัว เช่น

- product/:id
- product/:category/:id

Router Parameter

การรับค่าพารามิเตอร์

- request.params['ชื่อพารามิเตอร์'] หรือ
- request.params.ชื่อพารามิเตอร์



การเปลี่ยนเส้นทาง ด้วย Redirect

การทำ Routing นอกจากนิยามเส้นทางการรับ-ส่งข้อมูลแล้วสามารถที่จะเปลี่ยนเส้นทางการแสดงผลไปยัง Path อื่นๆได้โดยใช้คำสั่ง

- `response.redirect(path)`
- `response.redirect(URL)`

****ตรวจสอบเนื้อหา
ก่อนขึ้นหัวข้อใหม่**

จัดการ Static File



<https://www.youtube.com/c/KongRuksiamOfficial/>

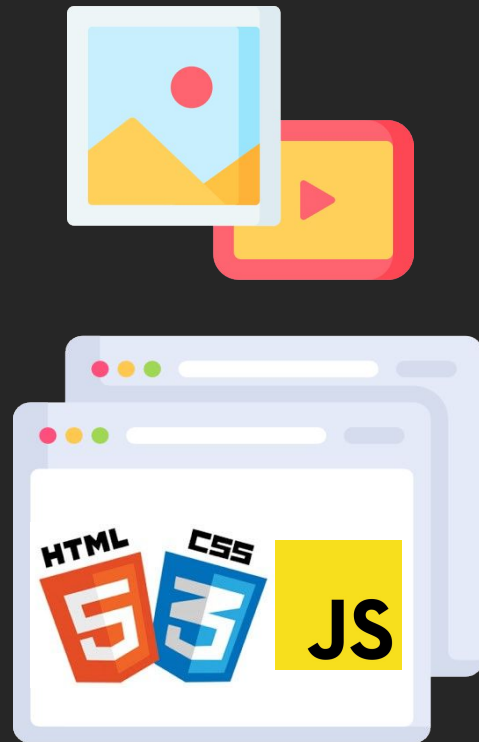


<https://www.facebook.com/KongRuksiamTutorial/>

Static File

คือ ไฟล์ที่ไม่มีการเปลี่ยนแปลงเนื้อหา เช่น ไฟล์ภาพ วิดีโอ เสียง หรือไฟล์โค้ดบางชนิด เช่น ไฟล์ html , css , js (Static Webpage) เป็นไฟล์ที่มีเนื้อหาตายตัว

หากนำไฟล์ดังกล่าวมารันก็จะแสดงผลตามที่ได้กำหนดไว้ในไฟล์หรือถ้ามีการเขียนโปรแกรมกำหนดเงื่อนไขขึ้นมาไฟล์นั้นก็จะไม่มีการเปลี่ยนแปลงเนื้อหาตามเงื่อนไขที่กำหนดขึ้น



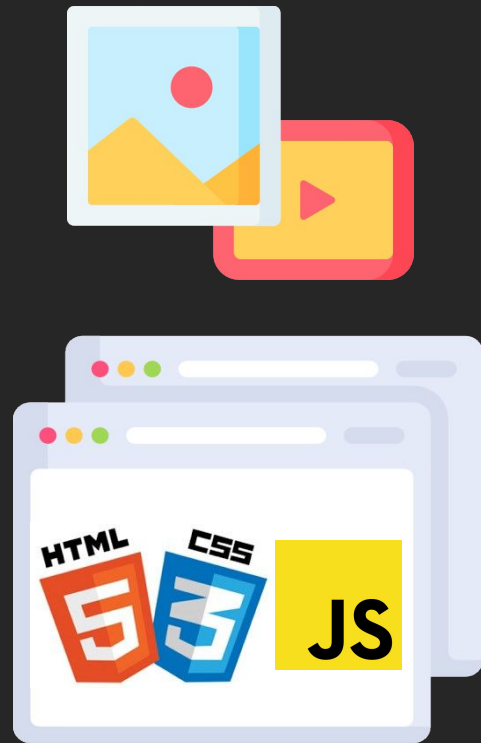
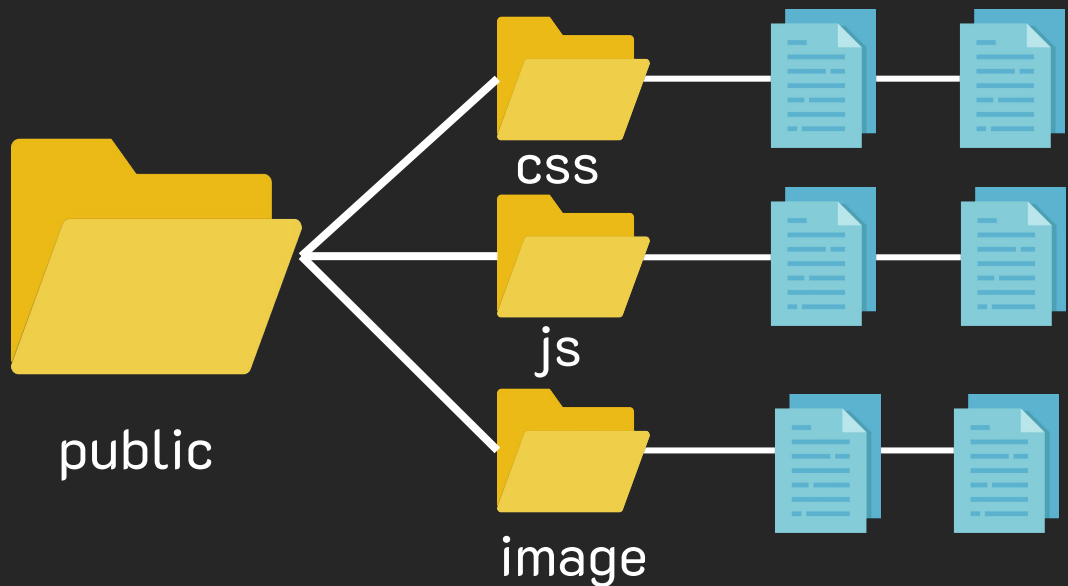
Static File

จากหัวข้อที่ผ่านมาเราจัดการ Static ไฟล์โดยให้จัดเก็บอยู่ในโฟลเดอร์ที่เราคิดหรือสร้างขึ้นเอง ยกตัวอย่าง เช่น โฟลเดอร์ Webpage , Template เป็นต้น ถ้าอยากเรียกใช้งานก็ดำเนินการผ่านโมดูล

แต่ถ้าไปใช้กับโปรเจคใหญ่ๆ ทำงานกันเป็นที่มาอาจจะต้องมีการกำหนดมาตรฐานในการจัดการกับ Static File ใหม่ เพื่อให้เกิดรูปแบบเดียวกัน ง่ายต่อการจัดการและเรียกใช้งาน มาตรฐานหลักที่ชาวโลกใช้กัน คือ เก็บไฟล์ Static ในโฟลเดอร์ที่มีชื่อว่า **public**

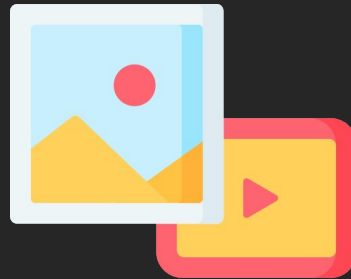
การจัดเก็บ Static File

ไฟล์รูปแบบ static จะถูกเก็บลงในโฟลเดอร์ public



การตั้งค่าใช้งาน Static File

```
const express = require('express')  
  
const app = express()  
  
app.use(express.static('public'))  
  
app.listen(8080,()=>{  
  })
```



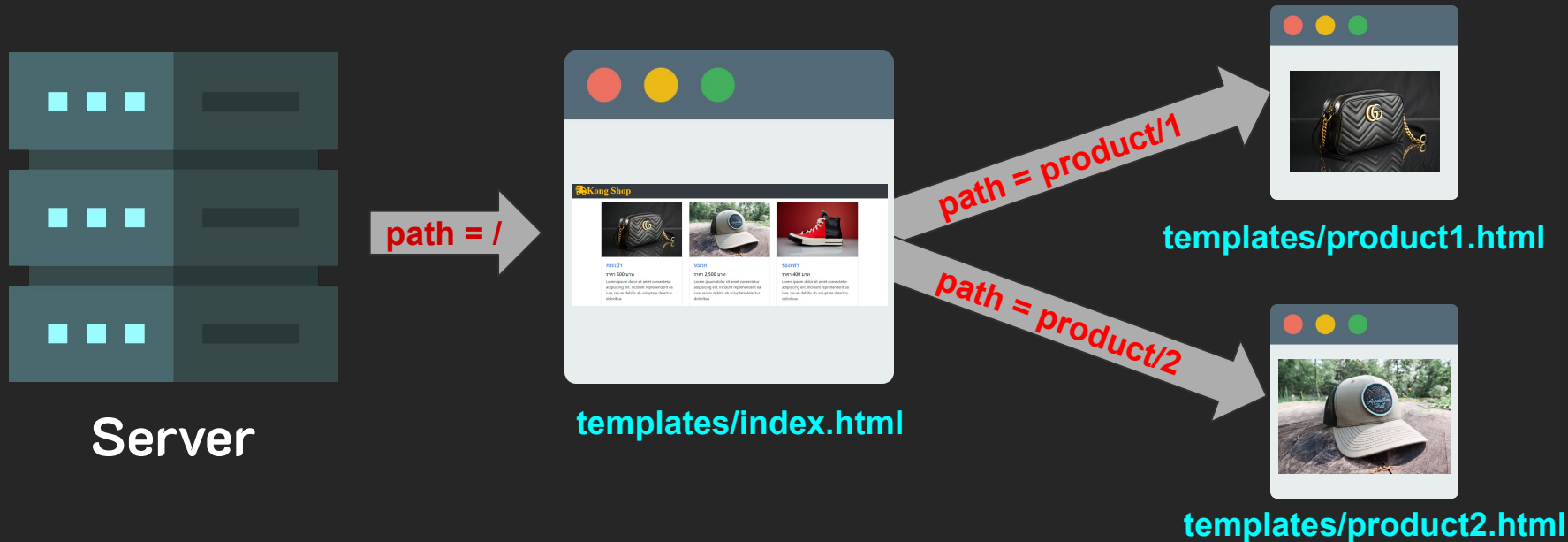
ทำไมต้องอ้างอิง Static File ?

Static File แทบจะไม่มีผลกับการทำงานฝั่ง Server เลย กล่าวคือ เป็นไฟล์ที่มีเนื้อหาตายตัว สามารถเข้าถึงและแสดงผลได้เลยโดยไม่ต้องพึ่งการทำงานฝั่ง Server หรือ อาจจะไม่ต้องอาศัยระบบ Routing มาอ้างอิงการทำงานก็ได้ แค่ว่าที่อยู่ของไฟล์ในโฟลเดอร์ public ก็ทำงานได้เลย



กระบวนการทำงานแบบเดิม

Server -> Routing -> กำหนด URL -> อ้างอิงตำแหน่งไฟล์ .html (static)



กระบวนการทำงานแบบเดิม

Server -> public -> อ้างอิงตำแหน่งไฟล์ .html (static)



โหลด Template K-Shop (kongruksiam-shop)



<https://github.com/kongruksiamza/data-nodejs-basic>

โครงสร้างไฟล์ K-Shop

▼ k-shop templates

> css

> images

> js

<> 404.html

<> form.html

<> index.html

<> manage.html

<> product.html



<https://www.youtube.com/c/KongRuksiamOfficial/>



<https://www.facebook.com/KongRuksiamTutorial/>

โครงสร้างไฟล์ K-Shop

▼ k-shop templates

> css

> images

> js

<> 404.html

<> form.html

<> index.html

<> manage.html

<> product.html

css - สำหรับจัดเก็บไฟล์ .css

images - สำหรับเก็บไฟล์ภาพ

js - สำหรับเก็บไฟล์ .js

****ทุกไฟล์จะถูกนำมาทำงานฝั่ง**

Frontend /Client ทั้งหมด

View & Template Engine



<https://www.youtube.com/c/KongRuksiamOfficial/>



<https://www.facebook.com/KongRuksiamTutorial/>

Dynamic File

คือ ไฟล์ที่มีการเปลี่ยนแปลงเนื้อหาหรือไฟล์ที่มีความยืดหยุ่นไม่มีเนื้อหาตายตัวและทำงานตามเงื่อนไขที่กำหนดขึ้นมา เช่น ให้แสดงผลเนื้อหาอ้างอิงตามข้อมูลที่อยู่ในฐานข้อมูล หรือ เนื้อหาอ้างอิงตามพื้นที่ของผู้ใช้งาน เป็นต้น จะเรียกไฟล์ดังกล่าวว่า Dynamic File หรือเรียกอีกชื่อคือ

View & Templates



View & Template

View คือ ส่วนที่ใช้แสดงเนื้อหาใน Web Browser ในรูปแบบของ Dynamic Web Page โดยเนื้อหาที่แสดงผลจะสามารถเปลี่ยนแปลงได้ เช่น ทำการดึงข้อมูลจากที่อื่นมาแสดงผล แล้วแทรกลงไปในส่วนที่ต้องการจะเรียกรูปแบบนี้ว่า **Template**



View & Template

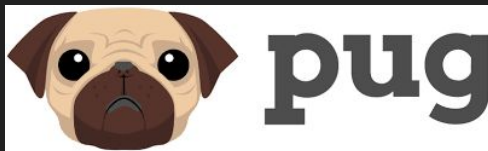


Template คือ หน้าตาแอปพลิเคชันเป็นส่วนที่ไว้ใช้
แสดงผลข้อมูลผลลัพธ์จากการประมวลผลข้อมูลในหน้า
เว็บร่วมกับ HTML , CSS ,JavaScript

ไฟล์ของ Template จะเก็บไว้ในโฟลเดอร์ที่มีชื่อว่า
Views เท่านั้น (หรือโฟลเดอร์อื่นต้องไปตั้งค่าเพิ่มเติม)

View & Template

<%= EJS %>



เนื่องจากการใช้งาน Node.js และ Express.js ยังไม่รองรับการใช้งาน Template โดยตรง จะต้องมีการติดตั้งส่วนที่จัดการกับ Template เรียกว่า “**Template Engine**” ซึ่งมีอยู่หลายโมดูลด้วยกัน เช่น EJS , Pug ,Jade เป็นต้น

ติดตั้ง EJS Template Engine



EJS เป็น Template Engine ที่มีรูปแบบโครงสร้าง
การเขียนคล้ายกับ HTML เพียงแต่สามารถเขียน
คำสั่ง JavaScript เข้าไปใน HTML ได้โดยที่ไฟล์นั้น
มีนามสกุล .ejs และสามารถติดตั้งโดยใช้คำสั่ง

`npm install ejs`



<https://www.youtube.com/c/KongRuksiamOfficial/>



<https://www.facebook.com/KongRuksiamTutorial/>

การตั้งค่าใช้งาน Template



```
const express = require('express')
```

```
const ejs = require('ejs')
```

```
const app = express()
```

```
// ตั้งค่าให้เป็น Engine สำหรับรันแอฟ
```

```
app.set('view engine', 'ejs')
```



การแสดง Template



```
router.get('/data',(req,res)=>{  
    res.render("ชื่อ Template")  
})
```


การส่งข้อมูลไปทำงานที่ Template



<https://www.youtube.com/c/KongRuksiamOfficial/>



<https://www.facebook.com/KongRuksiamTutorial/>

การส่งข้อมูลไปที่ Template



```
res.render("ชื่อ Template",{  
    property: value,  
    property:value  
})
```

การแสดงผลที่ Template



<%= ชื่อ Propertie %>

ในการนี้ส่งเป็นรูปแบบ html ระบุเป็น

<%- ชื่อ Propertie %>

โครงสร้างควบคุม (เงื่อนไข)



```
<% if (condition) { %>
```

```
<% }else{ %>
```

```
<% } %>
```



โครงสร้างควบคุม (การทำซ้ำ)



```
<% for (i=1;i<10;i++) { %>
```

```
// คำสั่งที่ต้องการทำซ้ำ
```

```
<% } %>
```

การส่ง Array ไปที่ Template



```
res.render("ชื่อ Template",{  
    products:[  
    }  
})
```

โครงสร้างควบคุม (การทำซ้ำ)



```
<% for (item in arr) { %>
```

```
    arr[item]
```

```
<% } %>
```



การแทรกไฟล์ใน Template (Include)



ในกรณีที่หน้าเว็บมีองค์ประกอบบางส่วนที่เหมือนกัน เช่น Header , Footer เมื่อมีการปรับปรุงแก้ไขคำสั่งในพื้นที่หรือองค์ประกอบดังกล่าว เราต้องปรับโค้ดทุกๆหน้าเพื่อให้ได้ผลลัพธ์เหมือนกัน วิธีแก้ปัญหาคือแยกส่วนประกอบดังกล่าวสร้างเป็นไฟล์แล้วนำเข้ามาทำงานในภายหลัง

```
<%- include("ชื่อไฟล์")%>
```


การรับ-ส่งข้อมูลผ่าน แบบฟอร์ม(Form)



<https://www.youtube.com/c/KongRuksiamOfficial/>



<https://www.facebook.com/KongRuksiamTutorial/>

HTML FORM

```
<form method="get|post" action="path">
```

```
// Element
```

```
</form>
```

รูปแบบการรับส่งข้อมูล



method="get หรือ post" (รูปแบบการส่งข้อมูล)

- get ส่งข้อมูลพร้อมแนบข้อมูลไปพร้อมกับ url (ไม่มีความปลอดภัยเพราะข้อมูลถูกมองเห็นและไม่ควรใช้งานร่วมกับข้อมูลที่เป็นแบบ sensitive data)
- post ส่งข้อมูลพร้อมซ่อนค่าข้อมูลระหว่างทางที่ส่งไป (มีความปลอดภัย)

รูปแบบการรับส่งข้อมูล



action = “path” ระบุ path ปลายทางเพื่อรับ
ข้อมูลที่ส่งไปจากฟอร์ม

การรับค่าข้อมูลจากฟอร์ม (Express)

- `app.get()` // รับข้อมูลจากฟอร์มแบบ get method
- `app.post()` // รับข้อมูลจากฟอร์มแบบ post method

จัดการข้อมูลที่ส่งแบบ GET

การส่งค่าจากแบบฟอร์ม

```
<form method="get" action="path"></form>
```

การรับค่า

```
app.get('path',(req,res)=>{
```

```
    console.log(req.query) // object ข้อมูลที่ส่งจากฟอร์ม
```

```
})
```

จัดการข้อมูลที่ส่งแบบ POST

การส่งค่าจากแบบฟอร์ม

```
<form method="post" action="path"></form>
```

การรับค่า

```
app.post('path',(req,res)=>{
```

```
    console.log(req.body) // object ข้อมูลที่ส่งจากฟอร์ม
```

```
})
```

จัดการข้อมูลที่ส่งแบบ POST

การตั้งค่าใน app.js

```
app.use(express.urlencoded({ extended: false }));
```

```
// ต้องระบุก่อน router
```



รู้จักกับ MongoDB และ Mongoose



<https://www.youtube.com/c/KongRuksiamOfficial/>



<https://www.facebook.com/KongRuksiamTutorial/>

MongoDB คืออะไร



เป็นฐานข้อมูลเชิงเอกสาร (Document Store) สำหรับเก็บข้อมูลขนาดใหญ่ที่มีความยืดหยุ่นสูง ง่ายต่อการปรับขนาดและทำงานข้าม Platform ได้ การจัดเก็บข้อมูลไม่ได้อยู่ในรูปแบบตาราง แต่จะอยู่ในรูปแบบเอกสาร JSON แล้วเซฟเก็บไว้ในเอกสารรูปแบบไบนารี BSON

โครงสร้างการจัดเก็บข้อมูลของ MongoDB



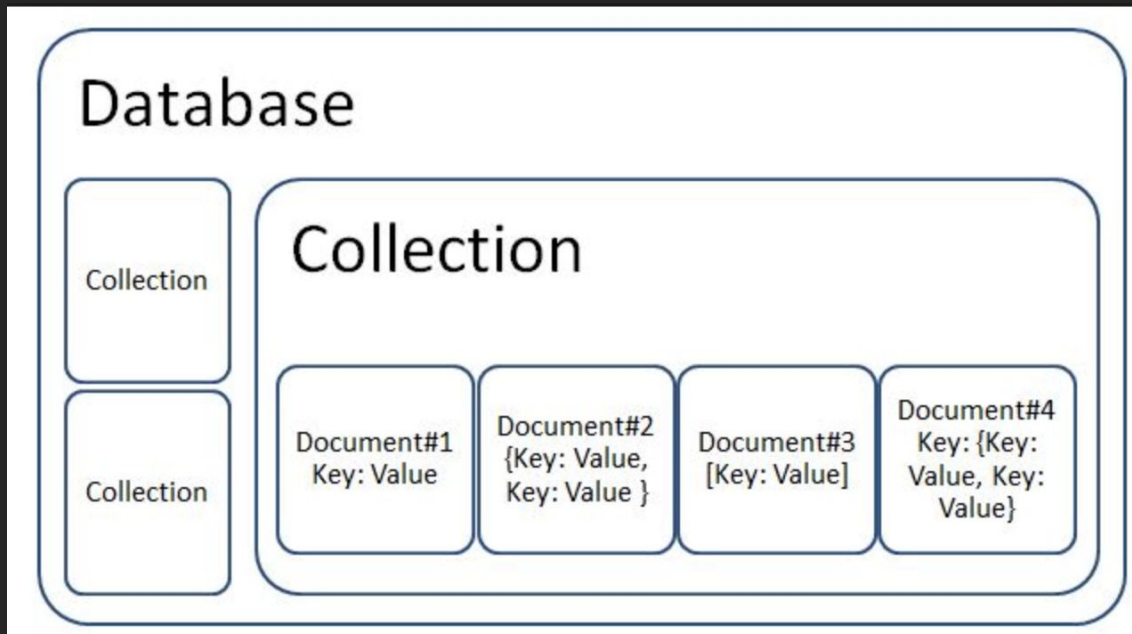
มีองค์ประกอบอยู่ 3 ส่วน ได้แก่

- Database
- Collection
- Documents



<https://phoenixnap.com/kb/wp-content/uploads/2021/05/document-database-illustration.png>

โครงสร้างการจัดเก็บข้อมูลของ MongoDB



https://miro.medium.com/max/1313/1*vhKORnX6ZQ5HNUaeqxbQGg.png



<https://www.youtube.com/c/KongRuksiamOfficial/>



<https://www.facebook.com/KongRuksiamTutorial/>

โครงสร้างการจัดเก็บข้อมูลของ MongoDB



- Database (ฐานข้อมูล) เป็นส่วนที่ใช้เก็บ Collection หรือชุดข้อมูล



โครงสร้างการจัดเก็บข้อมูลของ MongoDB

- Collection หรือชุดข้อมูลเทียบได้กับตารางในฐานข้อมูลเชิงสัมพันธ์
- Document เอกสารที่จัดเก็บข้อมูลของคู่คีย์ (Key) และค่า (Value)



จัดการฐานข้อมูลด้วย MongoDB

จัดการฐานข้อมูลด้วย

MongoDB NoSQL

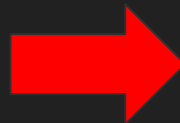
สำหรับผู้เริ่มต้น [7 ชั่วโมงเต็ม]

KongPhukorn Or เล่นทั้งหมด Kalam Studio

ปูพื้นฐาน MongoDB | NoSQL
สำหรับจัดการฐานข้อมูล [2021]

วิดีโอ 35 รายการ • การดู 952 ครั้ง • อัปเดตแล้ววันนี้

≡+ ↶ ↷ ...



สแกนเพื่อเข้าเรียน

mongoose

elegant **mongodb** object modeling for **node.js**



<https://www.youtube.com/c/KongRuksiamOfficial/>



<https://www.facebook.com/KongRuksiamTutorial/>

เชื่อมต่อ MongoDB ด้วย Mongoose

ถ้าต้องการให้โปรเจกต์เชื่อมโยงกับฐานข้อมูล MongoDB
ต้องดำเนินการผ่านตัวกลางที่เรียกว่า Driver ในหัวข้อนี้จะใช้
เครื่องมือตัวหนึ่งที่มีชื่อว่า

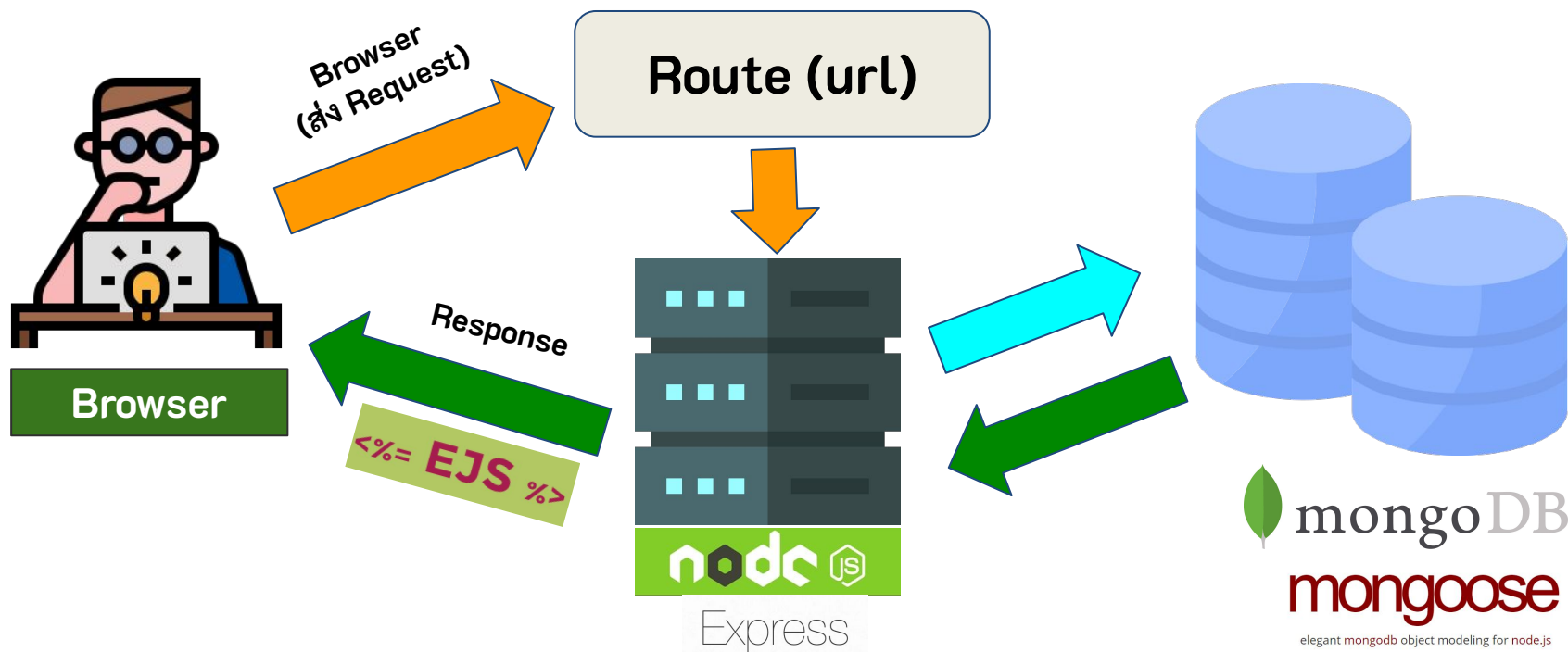
Mongoose

Mongoose คืออะไร

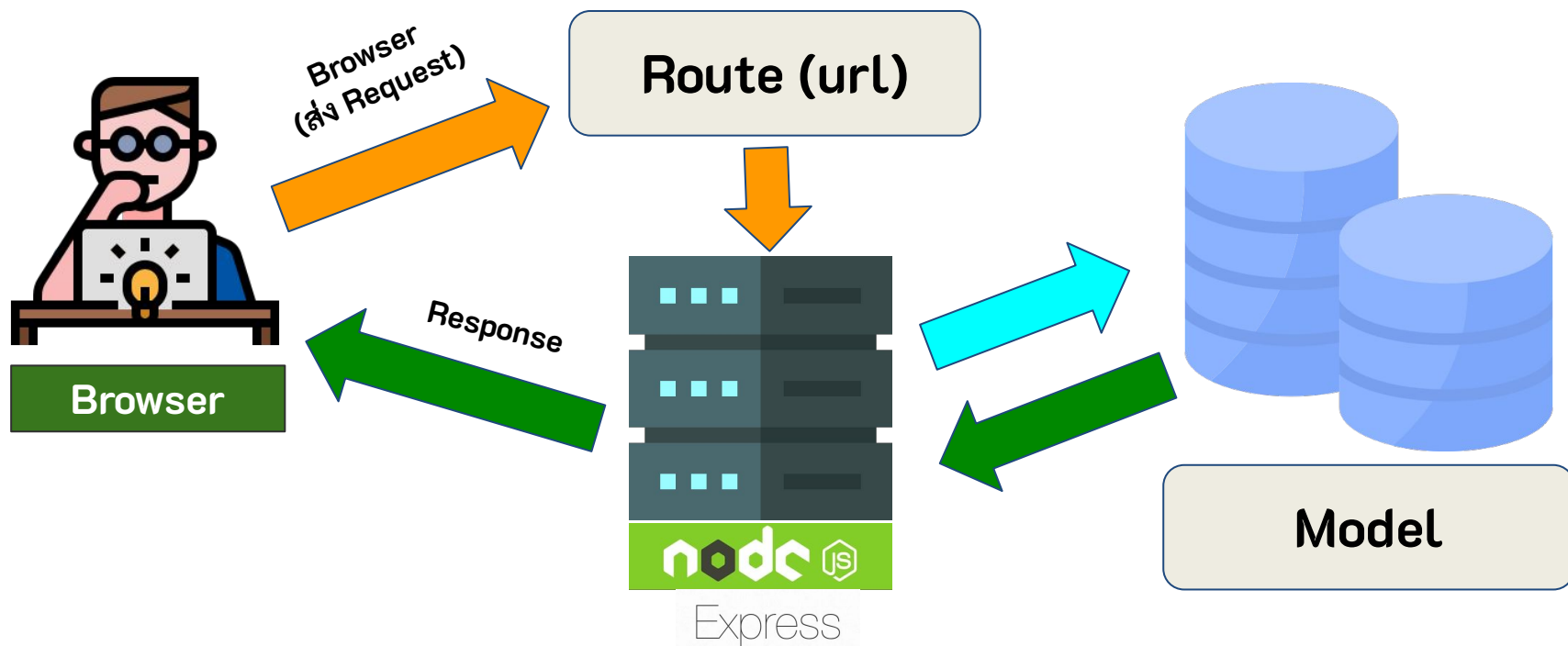
- เป็น Framework รูปแบบ ODM (Object Document Model) สำหรับติดต่อและดำเนินการกับฐานข้อมูลในรูปแบบ Document ที่มีข้อมูลด้านในเป็น Object
- กำหนดโครงสร้างของ Collection เป็น Schema
- จัดการข้อมูลผ่าน Model (เพิ่ม , ลบ , แก้ไข , สอบถาม)



ภาพรวมระบบ



ภาพรวมระบบ



การสร้างโมเดล

1. ติดตั้ง Mongoose
2. เชื่อม MongoDB ด้วย Mongoose
3. ออกแบบ Schema และ Model
4. นำโมเดลไปใช้งาน



ติดตั้ง Mongoose

- `npm install mongoose`

โครงสร้างคำสั่ง

```
mongoose.connect('mongodb://<hostname:port>/<database>')
```



เชื่อม MongoDB ด้วย Mongoose

```
const mongoose = require('mongoose')  
mongoose.connect('mongodb://<hostname:port>/<database>', {  
  useNewUrlParser:true,  
  useUnifiedTopology:true  
}).catch(err=>console.log(err))
```

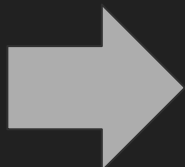
การสร้าง Schema และ Model

```
mongoose.Schema({
```

```
  field : type,
```

```
  field : type
```

```
})
```



```
const productSchema = mongoose.Schema({
```

```
  name :String,
```

```
  price: Number,
```

```
  description: String,
```

```
  imagePath:String
```

```
})
```

Schema คือ โครงสร้างในการเก็บข้อมูล

การสร้าง Schema และ Model

```
mongoose.model ('ชื่อ Collection', schema)
```

Model คือส่วนที่ใช้จัดการเกี่ยวกับข้อมูล เช่น

```
let Product = mongoose.model ('ชื่อ Collection', productSchema)
```

```
module.exports = Product; // export ออกไปใช้งาน
```


การบันทึกข้อมูล

```
const Product=require('../models/product')  
  
let doc =new Product({  
    name:req.body.name,  
    price:req.body.price,  
    description:req.body.description,  
    image:req.body.image  
})
```

```
Product.createProduct(doc,function(err){  
    if(err) console.log(err);  
    res.redirect("/")  
});
```

```
module.exports.createProduct=function(m  
odel,data){  
    model.save(data)  
}
```

อัปโหลดไฟล์ด้วย Multer

ประกอบด้วย 5 ขั้นตอนดังนี้

1. ติดตั้ง Multer
2. ตั้งค่า Form
3. กำหนด Option (ตำแหน่งจัดเก็บไฟล์ , ชื่อไฟล์ (ชื่อไฟล์ห้ามซ้ำกัน))
4. กำหนดการอัปโหลดไฟล์
5. บันทึกข้อมูล

อัปโหลดไฟล์ด้วย Multer

ติดตั้ง Multer

- `npm install multer`

อัปโหลดไฟล์ด้วย Multer

ตั้งค่า Form

```
<form class="form-horizontal"  
enctype="multipart/form-data">
```

อัปโหลดไฟล์ด้วย Multer

```
const multer = require('multer');  
const storage = multer.diskStorage({  
  destination: function(req, file, cb) { // ตำแหน่งเก็บไฟล์ภาพ  
    cb(null, './public/images/products');  
  },  
  filename: function(req, file, cb) {  
    cb(null, Date.now() + ".jpg"); // กำหนดชื่อไฟล์ไม่ซ้ำกันโดยใช้ Date  
  }  
});
```

อัปโหลดไฟล์ด้วย Multer

กำหนดการอัปโหลดไฟล์

```
const upload = multer({  
  storage: storage  
});
```

บันทึกข้อมูล

```
router.post('/insert',upload.single('image'),  
(req,res)=>{  
  image:req.file.filename  
})
```

การแสดงผลข้อมูล

```
Product.find().exec((err,doc)=>{  
    res.render('index',{products:doc})  
})
```

ลบข้อมูลและยืนยันการลบ

```
<a href="/delete/<%=item._id%>"  
onclick="return confirm('คุณต้องการลบหรือไม่ ? ')">  
ลบ</a>
```


ลบข้อมูลและยืนยันการลบ

```
router.get('/delete/:id',(req,res)=>{  
    Product.findByIdAndDelete(req.params.id,  
    {useFindAndModify:false}).exec(err=>{  
        res.redirect('/manage')  
    })  
});
```

แสดงรายละเอียดสินค้า



<https://www.youtube.com/c/KongRuksiamOfficial/>



<https://www.facebook.com/KongRuksiamTutorial/>

แก้ไขข้อมูล



<https://www.youtube.com/c/KongRuksiamOfficial/>



<https://www.facebook.com/KongRuksiamTutorial/>

อัปเดตข้อมูล



<https://www.youtube.com/c/KongRuksiamOfficial/>



<https://www.facebook.com/KongRuksiamTutorial/>

Fixed Template



<https://www.youtube.com/c/KongRuksiamOfficial/>



<https://www.facebook.com/KongRuksiamTutorial/>

รู้จักกับ Cookie



<https://www.youtube.com/c/KongRuksiamOfficial/>



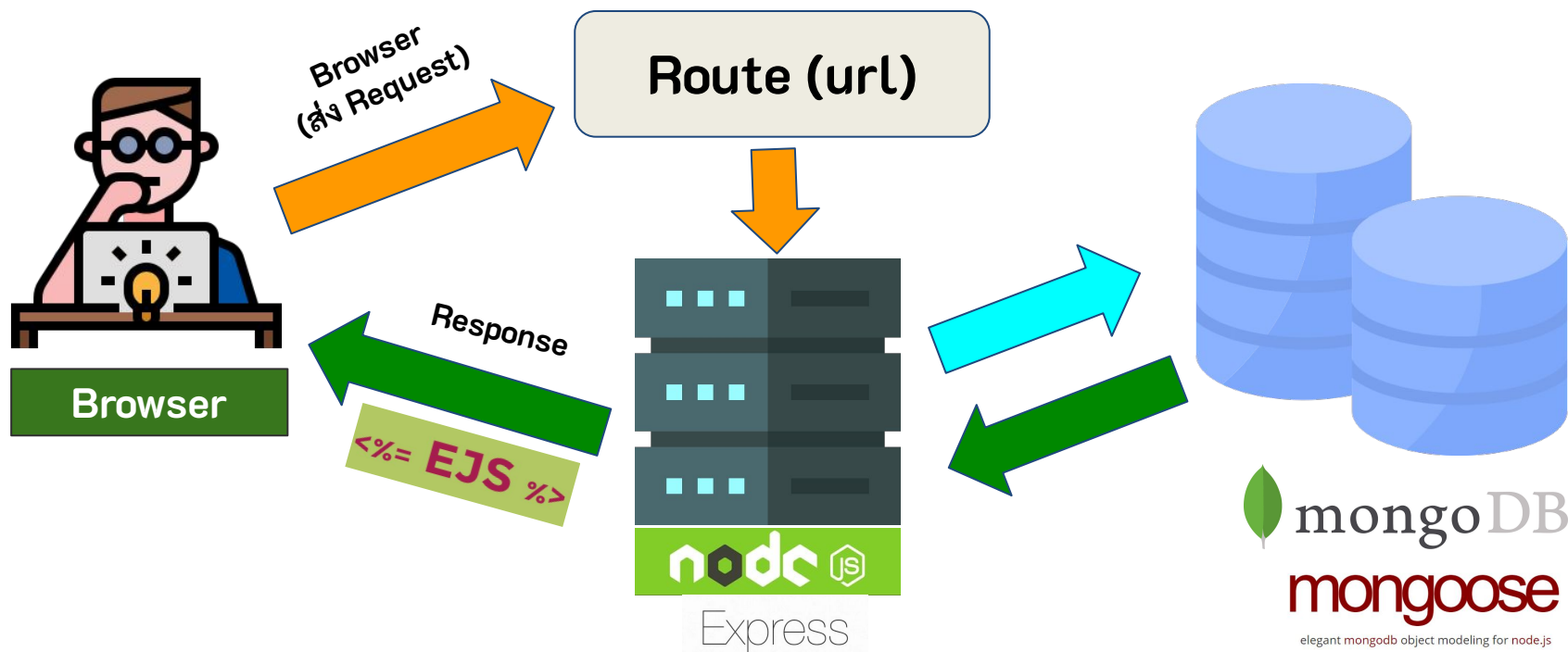
<https://www.facebook.com/KongRuksiamTutorial/>

รู้จักกับ Cookie

การทำงานระหว่าง Browser และ Server นั้นจะเกิดขึ้นเมื่อ Browser ส่ง Request ไปที่ Server และ Server Responseกลับมามีการทำงานก็จะสิ้นสุดลงและเมื่อปิด Browser ข้อมูลต่างๆที่แสดงผลใน Web Page ก็จะถูกทำลายไป แต่ถ้าเราต้องการอยากจะทำให้ Web Page คงสถานะหรือเชื่อมโยงการทำงานระหว่าง Web Page เข้าด้วยกันได้ จะอาศัยส่วนที่เรียกว่า Cookie และ Session



รู้จักกับ Cookie

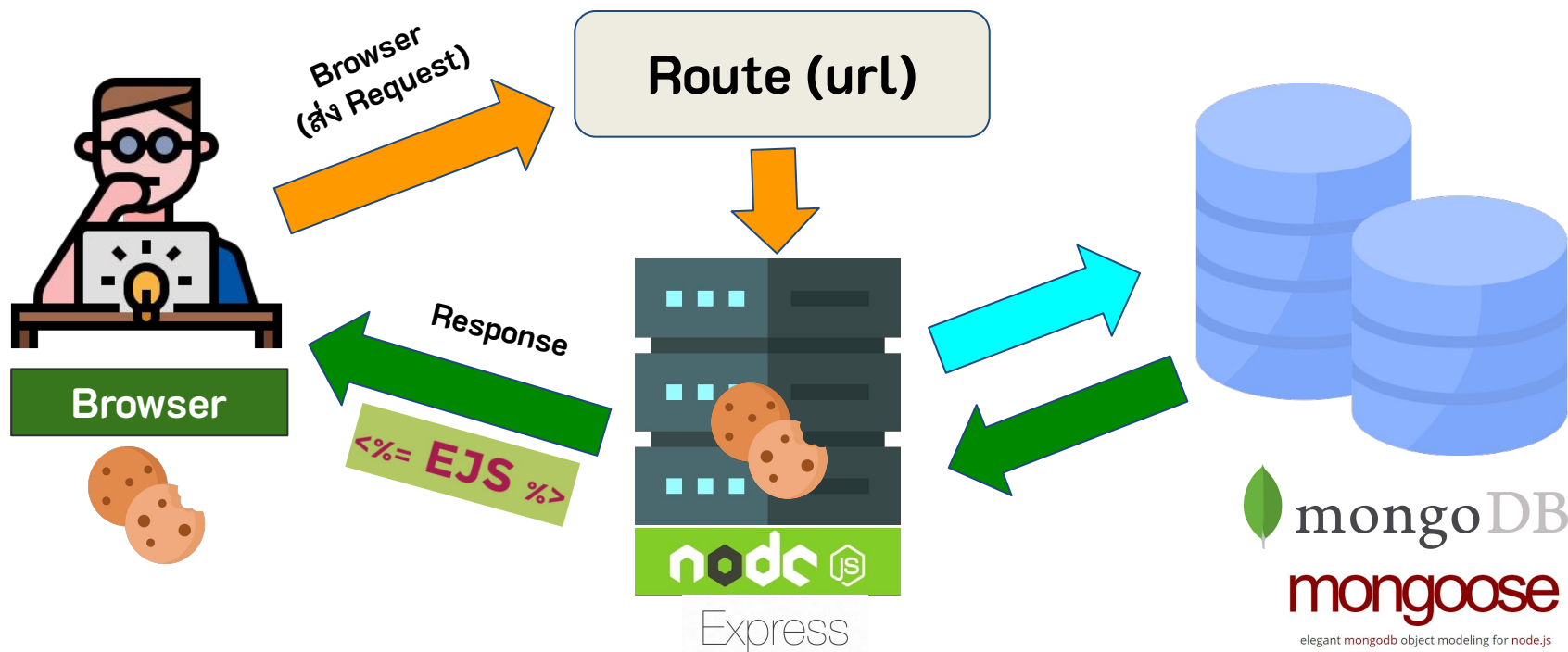


Cookie (คุกกี้)

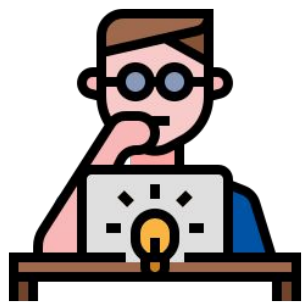
เป็นรูปแบบการเก็บข้อมูลบางอย่างไว้ในเครื่องผู้ใช้
เพื่อที่จะนำข้อมูลดังกล่าวมาใช้ในภายหลังถึงจะปิด
Browser ไปแต่ข้อมูลใน Cookie ก็ยังคงอยู่



Cookie (คุกกี้)



กำหนดสิทธิ์ในการเข้าถึงข้อมูล



Browser



cookie : admin



จัดการสินค้า

cookie : admin



บันทึกสินค้า

คุณสมบัติของ Cookie (คุกกี้)

- ข้อมูลคุกกี้จะเก็บไว้ในไฟล์ Text ธรรมดา ซึ่งคุกกี้ของแต่ละเว็บจะถูกแยกคนละไฟล์
- คุกกี้มีระดับความปลอดภัยที่ค่อนข้างต่ำ เนื่องจากเก็บที่ฝั่งผู้ใช้ (Client/Browser) จึงเหมาะจะใช้เก็บข้อมูลที่ไม่เป็นความลับ
- ในกรณีที่คุกกี้หมดอายุหรือไฟล์ที่เก็บข้อมูลคุกกี้เสียหายก็จะไม่สามารถใช้งานคุกกี้ได้

การใช้งาน Cookies



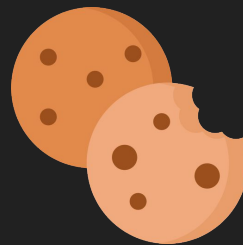
ติดตั้งโมดูล

- `npm install cookie-parser`

การตั้งค่าใช้งาน

```
const cookieParser = require('cookie-parser')
```

```
app.use(cookieParser())
```



การเก็บข้อมูล Cookie

```
res.cookie("ชื่อ cookie", "ค่าใน Cookie")
```

การอ่านข้อมูลใน Cookie

```
req.cookie.ชื่อ cookie
```



การกำหนดอายุ Cookie

```
res.cookie("ชื่อ cookie", "ค่าใน Cookie", {maxAge:หน่วยมิลลิวินาที})
```

การลบ Cookie

```
res.clearCookie('ชื่อ cookie ที่ต้องการลบ')
```



Session (เซสชัน)

เป็นรูปแบบการเก็บข้อมูลบางอย่างไว้ใน Server
เพื่อที่จะนำข้อมูลดังกล่าวมาใช้ในภายหลัง



คุณสมบัติของ Session (เซสชัน)

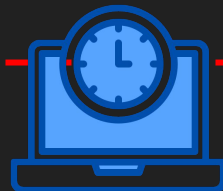
- เมื่อผู้ใช้ทำการเชื่อมต่อกับเว็บไซต์หรือทำงานกับ Server ตัว Server จะสร้างรหัสสำหรับอ้างอิงผู้ใช้คนนั้น โดยรหัสดังกล่าวจะเรียกว่า Session ID โดยผู้ใช้ที่เชื่อมต่อจะมี Session ID ที่มีค่าไม่ซ้ำกันสำหรับอ้างอิงตัวผู้ใช้งานเอง
- Session ID จะถูกอ้างอิงระหว่างที่ผู้ใช้งานเชื่อมต่อกับ Server และเมื่อยกเลิกการเชื่อมต่อค่า Session ID ก็จะถูกยกเลิกด้วย เช่น เมื่อผู้ใช้ทำการปิด Browser และเมื่อมีการเชื่อมต่อใหม่อีกครั้งค่า Session ID ก็จะถูกสร้างขึ้นมาใหม่ด้วยเช่นกัน

คุณสมบัติของ Session (เซสชัน)

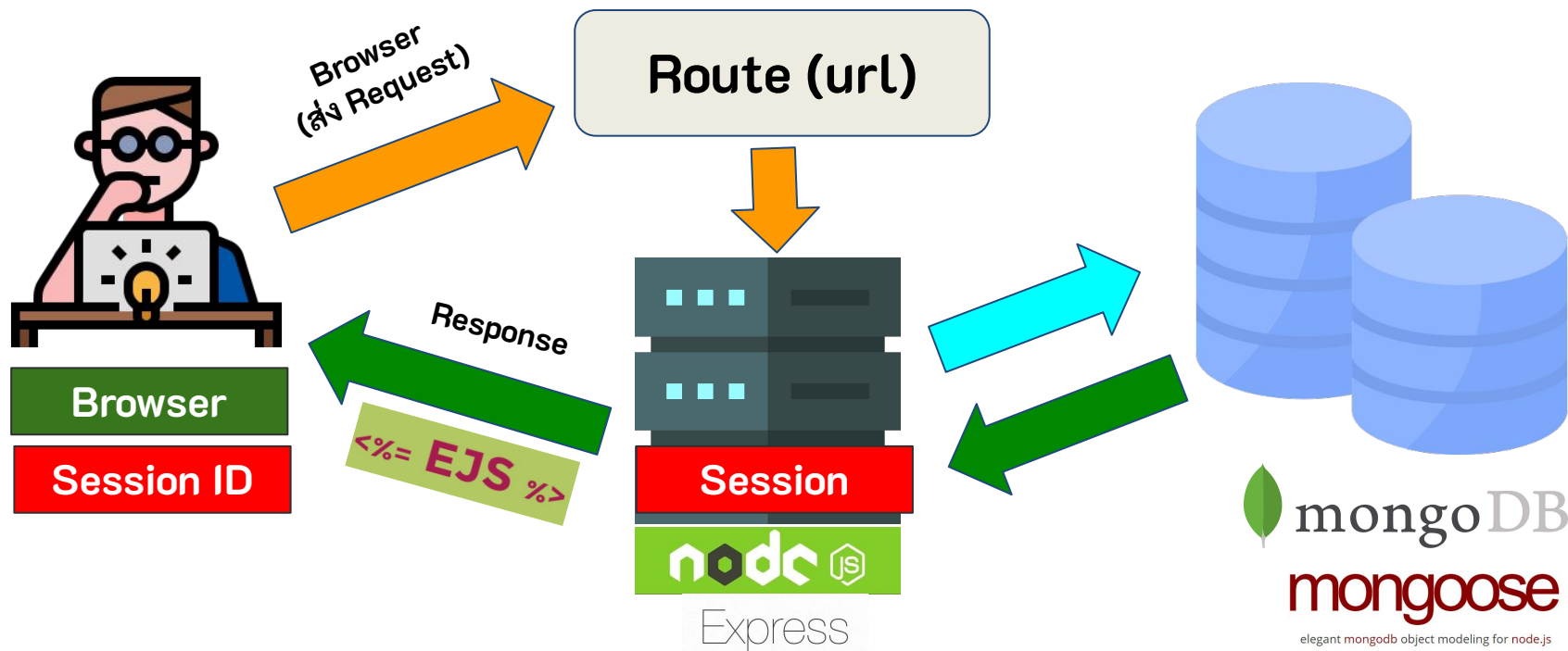
- ข้อมูลใน Session จะนำมาจำแนกผู้ใช้แต่ละคนออกจากกัน มันจะใช้ได้เฉพาะกับผู้ใช้คนๆนั้นไม่สามารถใช้งานร่วมกับคนอื่นได้
- ค่า Session ID ไม่สามารถใช้ร่วมกันระหว่าง Browser ได้ เช่น Session ที่ทำงานใน Google Chrome ไม่สามารถนำไปใช้กับ Firefox หรือ Safari ได้
- ข้อมูล Session ที่สร้างขึ้น สามารถนำไปทำงานในแต่ละ Page ได้เหมือนกับ Cookie เลย

คุณสมบัติของ Session (เซสชัน)

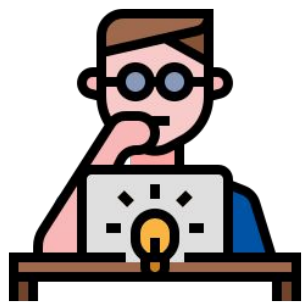
- ข้อมูล Session จะใช้งานได้แค่ชั่วคราวเท่านั้น คือเชื่อมต่อเมื่อเวลาเปิด Browser แต่เมื่อปิด Browser ไปข้อมูลก็จะถูกยกเลิกการเชื่อมต่อ
- ถ้าต้องการให้สภาพการเชื่อมต่อยังคงอยู่หรือการทำงานเหมือน Cookie จำเป็นต้องอาศัยการจัดเก็บ Session ID ไว้ที่เครื่องผู้ใช้และเก็บข้อมูล Session ไว้ที่ Server แทน



Session (เซสชัน)



Session (เซสชัน)



Browser

Session ID

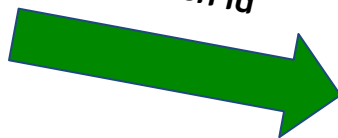


session id



จัดการสินค้า

session id

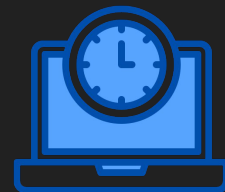


บันทึกสินค้า

เปรียบเทียบ Cookie กับ Session

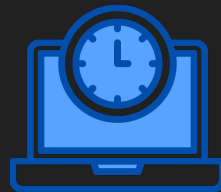
| Cookie | Session |
|---|---|
| เก็บข้อมูลผู้ใช้งานระบบ | เก็บข้อมูลผู้ใช้งานระบบ |
| เก็บไว้ที่เครื่องผู้ใช้ ไม่สิ้นเปลืองทรัพยากรของระบบ | เก็บไว้ที่เครื่อง Server อาจจะสิ้นเปลืองทรัพยากรของระบบมากกว่า แต่เก็บค่า Session ID ในรูปแบบ Cookie ได้เพื่ออ้างอิงข้อมูลผู้ใช้งาน |
| นำข้อมูลใน Cookie มาใช้งานได้ตลอดถ้า Cookie ไม่หมดอายุ | นำข้อมูลใน Session มาใช้งานได้ตลอดถ้า Session ไม่หมดอายุ |
| สำหรับเก็บข้อมูลที่ไม่เป็นความลับ เพราะสามารถดูข้อมูลใน Cookie ได้ที่ฝั่งผู้ใช้ | สำหรับเก็บข้อมูลที่เป็นความลับ เพราะข้อมูลเก็บไว้ที่ Server |

การใช้งาน Session



ติดตั้งโมดูล

- `npm install express-session`



การตั้งค่าใช้งาน

```
const session = require('express-session')
```

```
app.use(session({
```

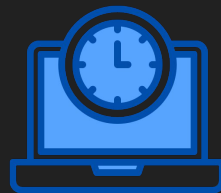
```
  secret:"key สำหรับสร้าง session id",
```

```
  resave:false,
```

```
  saveUninitialized:false
```

```
  }
```

```
))
```

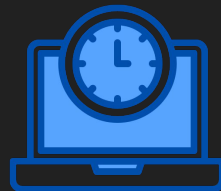


การเก็บข้อมูล Session

`req.session.ชื่อ session = ค่าใน session`

การอ่านข้อมูลใน Session

`req.session.ชื่อ session`



การกำหนดอายุ Session

```
req.session.cookie.maxAge = หน่วยมิลลิวินาที
```

การลบ Session

```
res.session.destroy((err)=>{
```

การทำงานเมื่อลบ Session เสร็จ

```
})
```



เข้าเรียนเนื้อหา Nodejs เบื้องต้น 2021 ได้ที่

<https://bit.ly/3LH1Kzu>