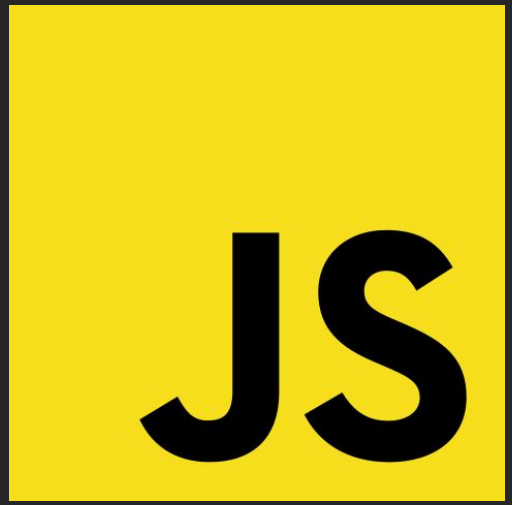
Nodejs เบื้องต้น

ผู้เรียนต้องมีพื้นฐานอะไรบ้าง

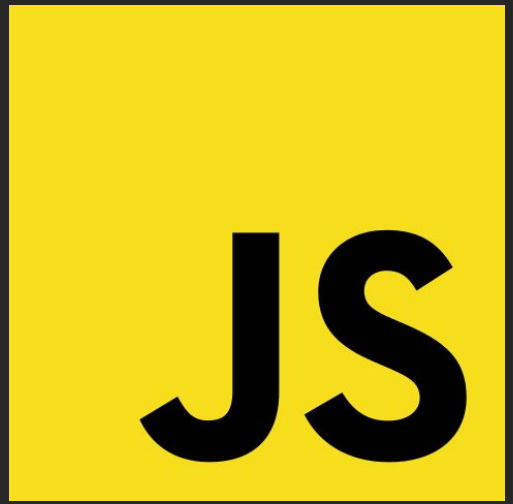
- HTML5 เบื้องต้น
- CSS3 เบื้องต้น
- JavaScript เบื้องต้น
- JavaScript ES6
- MongoDB เบื้องต้น

รู้จักกับ JavaScript และ Nodejs



JavaScript คือ ภาษาคอมพิวเตอร์ที่ใช้ในการพัฒนาเว็บร่วมกับ HTML เพื่อให้เว็บมีลักษณะแบบไดนามิก หมายถึง เว็บสามารถตอบสนองกับผู้ใช้งานหรือแสดงเนื้อหาที่แตกต่างกันไปโดยจะอ้างอิงตามเว็บบราวเซอร์ที่ผู้เข้าชมเว็บใช้งานอยู่

รู้จักกับ JavaScript และ Nodejs



เป็นภาษาที่ทำงานฝั่งผู้ใช้ (Client Side Script) โดยเว็บเบราว์เซอร์จะทำหน้าที่ประมวลผลคำสั่งที่ถูกเขียนขึ้นมาและตอบสนองต่อผู้ใช้ได้ทันที เช่น การแสดงข้อความแจ้งเตือน (Alert) การตรวจสอบข้อมูลที่ผู้ใช้ป้อน (Validation) เป็นต้น

รู้จักกับ JavaScript และ Nodejs

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

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

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

รู้จักกับ JavaScript และ Nodejs

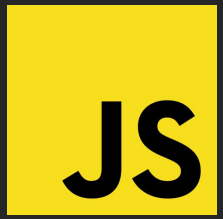
ในปัจจุบันได้มีการพัฒนาชุดแปลคำสั่งของ JavaScript ขึ้นมาใหม่โดยที่ไม่ต้องขึ้นอยู่กับ Web Browser นั่นก็คือ

รู้จักกับ JavaScript และ Nodejs

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

รู้จักกับ JavaScript และ Nodejs

ส่งผลให้สามารถรันโค้ด JavaScript ด้วย Nodejs ได้เลย โดยไม่จำเป็นต้องสร้างเป็นเว็บเพจแล้วนำเว็บเพจไปรันใน Web Browser นั่นเอง (ไม่ง้อ Web Browser)



รู้จักกับ JavaScript และ Nodejs

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 หน้าตาของแอพพลิเคชั่น) โดยผู้ใช้งานสามารถมองเห็นและมีส่วนร่วมหรือโต้ตอบภายใน Web Browser ได้

Backend คือ การพัฒนาโปรแกรมหลังบ้านหรือการทำงานเบื้องหลังในแอพ เช่น การทำงานกับฐานข้อมูล เป็นต้น โดยผู้ใช้งานไม่สามารถมีส่วนร่วมหรือโต้ตอบได้

ข้อดีของ Nodejs

- Nodejs ใช้ JavaScript ในการพัฒนาเว็บทั้งฝั่ง Frontend และ Backend
- ทำงานแบบ Non-Blocking I/O โดยใช้วิธีการแบบ Asynchonous โดยจะไม่รอการตอบสนองแต่ละ Request ให้แล้วเสร็จ แต่จะทำการย้ายการทำงานไปอยู่เบื้องหลัง แล้วรอรับ Request ต่อๆไปทันที
- 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

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 Asynchonous โดยจะไม่รอการตอบสนองแต่ละ Request ให้แล้วเสร็จ แต่จะทำการย้ายการทำงานไปอยู่เบื้องหลัง แล้วรอรับ Request ต่อๆไปทันที
- แบ่งการทำงานออกเป็นแต่ละโมดูล แล้วนำมาใช้เฉพาะส่วนที่จำเป็น ทำให้โค้ดที่ต้องประมวลผลมีขนาดเล็กลง

ข้อดีของ Node.js

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

Non-Blocking I/O

โดยทั่วไปกระบวนการทำงานภายในโปรแกรมจะเป็นไปตามลำดับขั้นตอนที่กำหนดไว้ หมายถึง **ต้องทำงานส่วนนี้ให้เสร็จก่อนถึงจะไปทำงานถัดไปได้ (Blocking)** แต่การทำงานบางอย่างอาจจะต้องใช้เวลาต้องรอจนกว่างานนั้นจะเสร็จ ถึงจะเริ่มต้นทำงานถัดไป ซึ่งทำให้เกิดความล่าช้าในการทำงาน **ใน JavaScript จะจัดการปัญหาล่าช้าดังกล่าวโดยวิธีการทำงานที่เรียกว่า**

Non-Blocking หรือ Asynchronous

Non-Blocking I/O

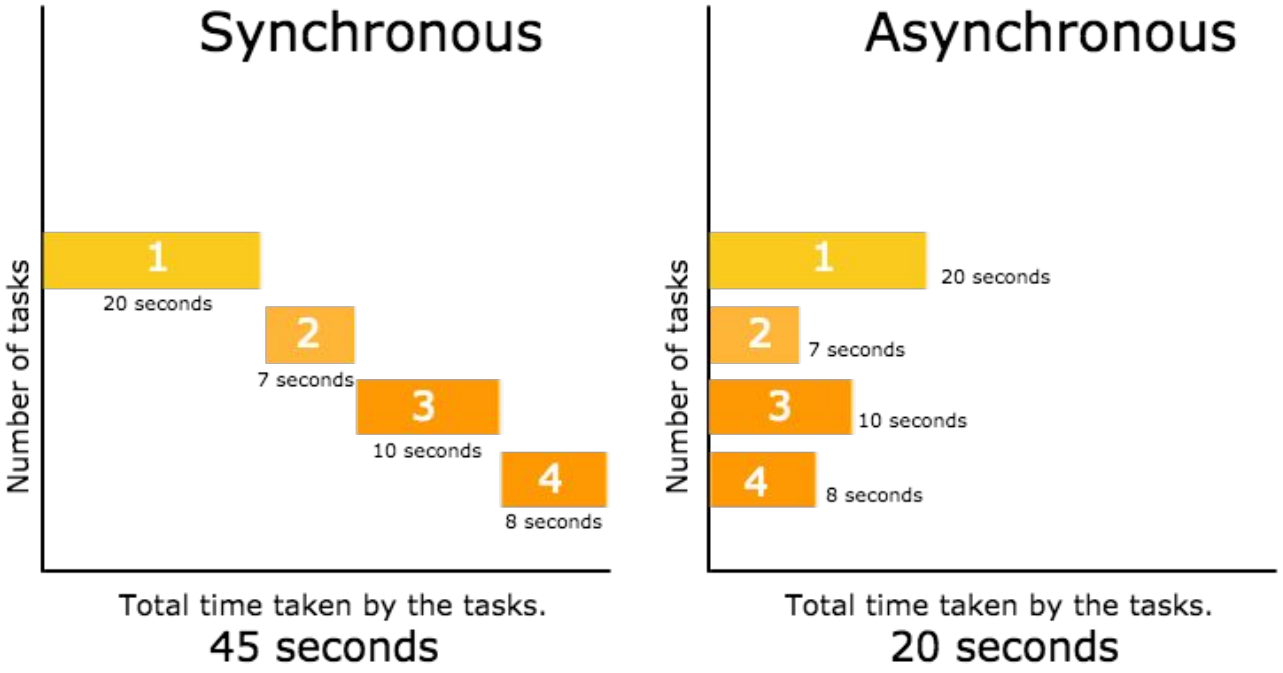
Non-Blocking หรือ Asynchronous

หมายถึง เวลาที่ระบบเริ่มต้นกระบวนการประมวลผลหรือทำงานไปแล้ว **ถ้างานนั้นยังทำไม่เสร็จหรือรอแล้วใช้ระยะเวลานาน จะข้ามไปทำงานถัดไปได้เลย แล้วค่อยกลับมาทำส่วนที่รอในภายหลัง** วิธีนี้จะช่วยให้การทำงานมีความรวดเร็วมากยิ่งขึ้น เพราะไม่ต้องเสียเวลารอในบางขั้นตอน

Blocking และ Non-Blocking

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

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



http://www.phpmind.com/blog/2017/05/synchronous-and-asynchronous/

Non-Blocking หรือ Asynchronous ในชีวิตประจำวัน

- **ทานข้าว -** เราสามารถที่จะทานข้าวไปพร้อมๆกับดูคลิปในยูทูปได้ในเวลาเดียวกัน
- **สั่งอาหาร -** เราสามารถสั่งให้คนมาส่งอาหารให้เราได้พร้อมๆกับนั่งเขียนโค้ดไปด้วยในเวลาเดียวกัน ระหว่างรอคนส่งของมาถึง งานเราก็ใกล้จะเสร็จพอดีในเวลาเดียวกัน
- **โปรแกรมแชท -** เราสามารถที่จะอัพโหลดและส่งภาพในช่องแชทได้ ระหว่างที่รออัพโหลดและส่งภาพไปเราก็ยังสามารถพิมพ์แชทได้ในเวลาเดียวกัน โดยที่ไม่ต้องรอให้อัพโหลดภาพและส่งภาพเสร็จค่อยแชทได้

Blocking

เป็นการทำงานที่เกิดการรอเกิดขึ้น ไม่ว่าจะมีการทำงานหลายๆงานพร้อมกัน ต้องรอทำงานเก่าจนเสร็จก่อนแล้วค่อยไปทำงานใหม่ลำดับถัดไป

ข้อดี

- มีการทำงานที่เป็นลำดับขั้นตอนและเข้าใจง่าย

ข้อเสีย

- สิ้นเปลืองทรัพยากร

Non - Blocking

เป็นการทำงานโดยที่ไม่ต้องรอการทำงานของงานก่อนหน้าเสร็จ ก็สามารถไปทำงานอื่นๆได้เลยทันที

ข้อดี

- ทำงานเร็วเพราะไม่ต้องรอทำงานตามลำดับ

ข้อเสีย

- การทำงานไม่เป็นลำดับอาจจะมีความเข้าใจยาก

แนวคิดและการทำงานของ Node.js

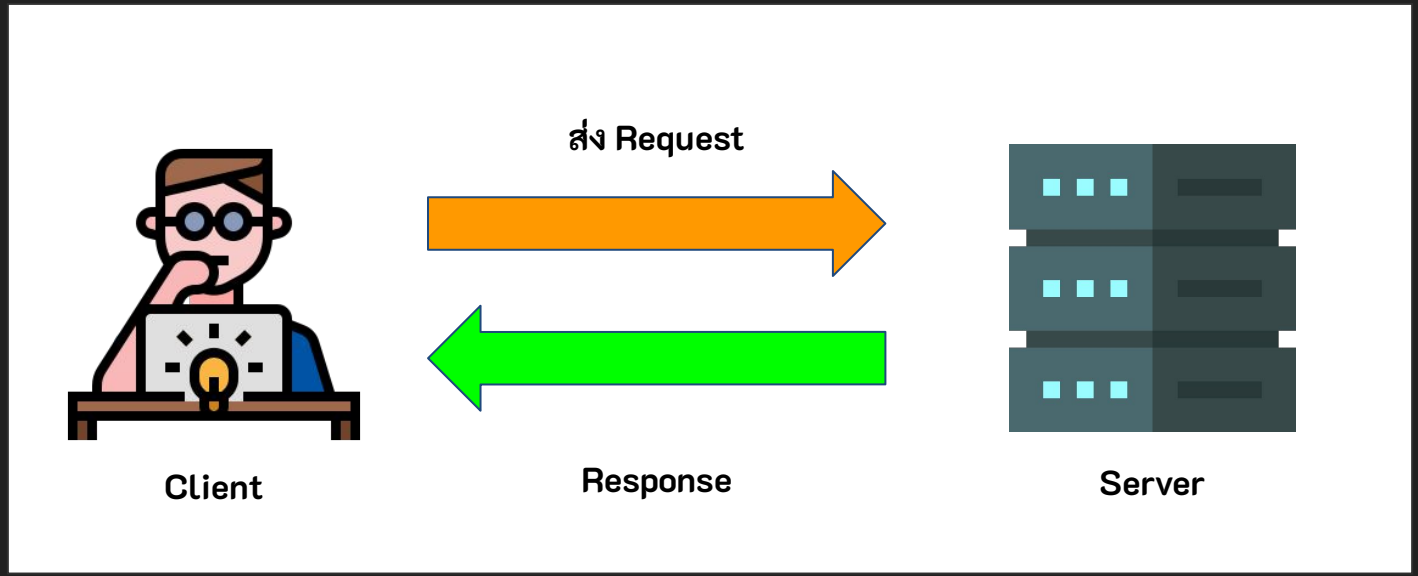
https://www.youtube.com/c/KongRuksiamOfficial/

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

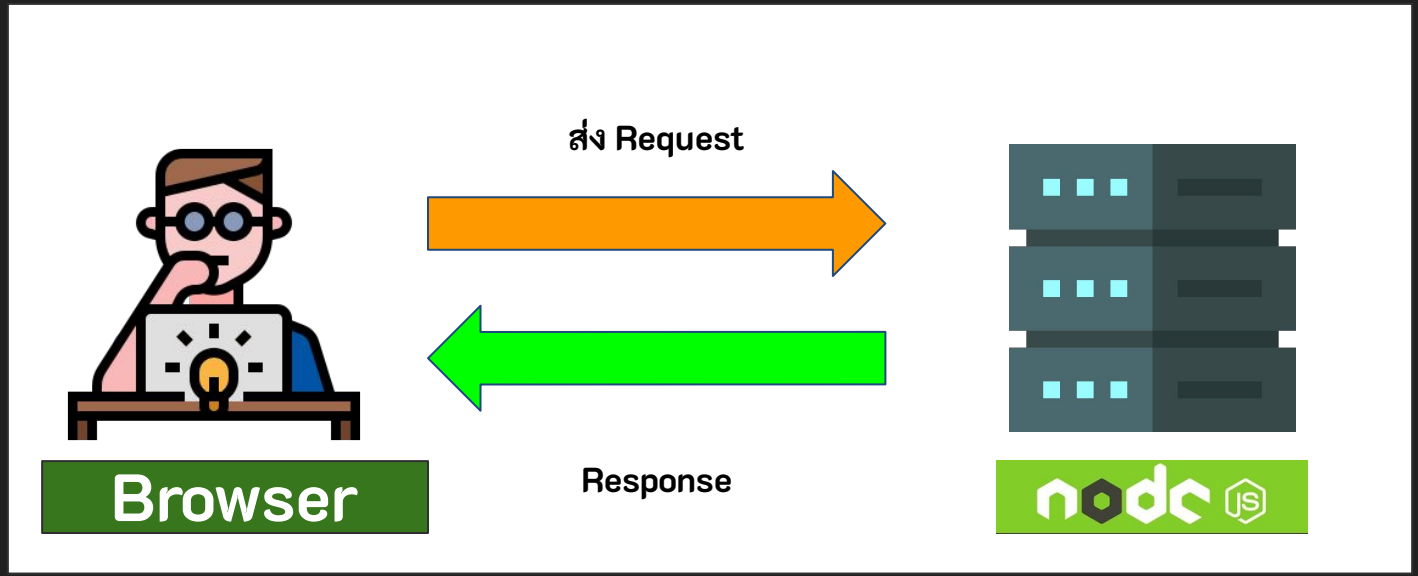
แผนภาพการทำงาน



คำศัพท์พื้นฐาน

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

แผนภาพการทำงาน



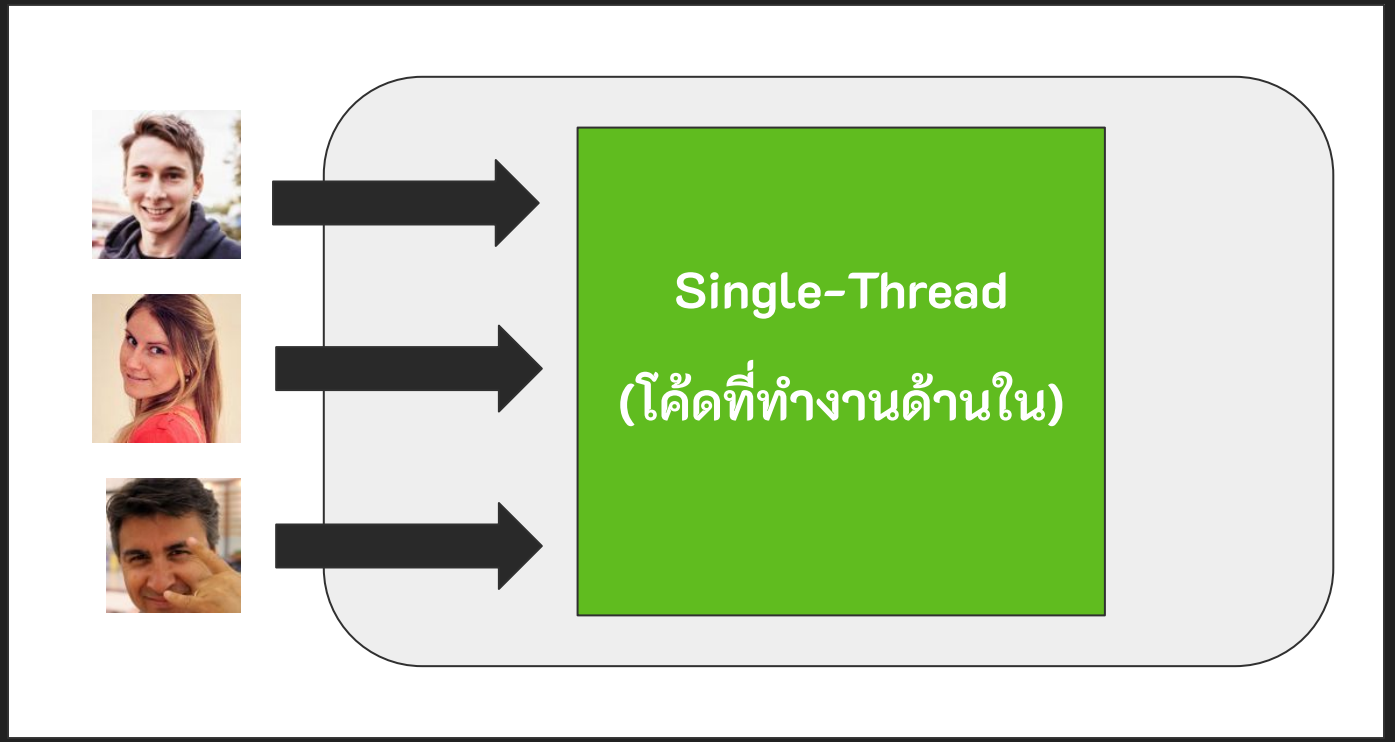
รูปแบบการทำงานของ Node.js

รูปแบบการทำงานของ Node.js

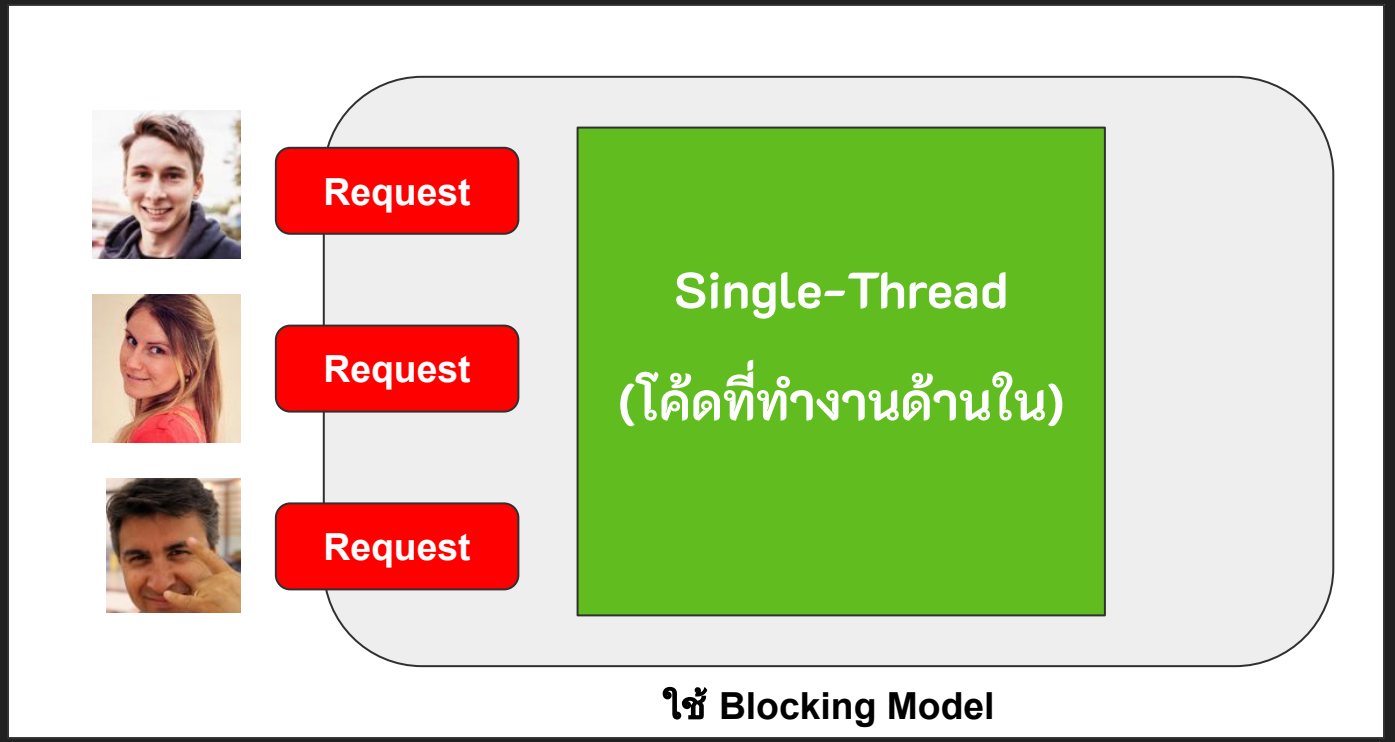
Single-Thread

(โค้ดที่ทำงานด้านใน)

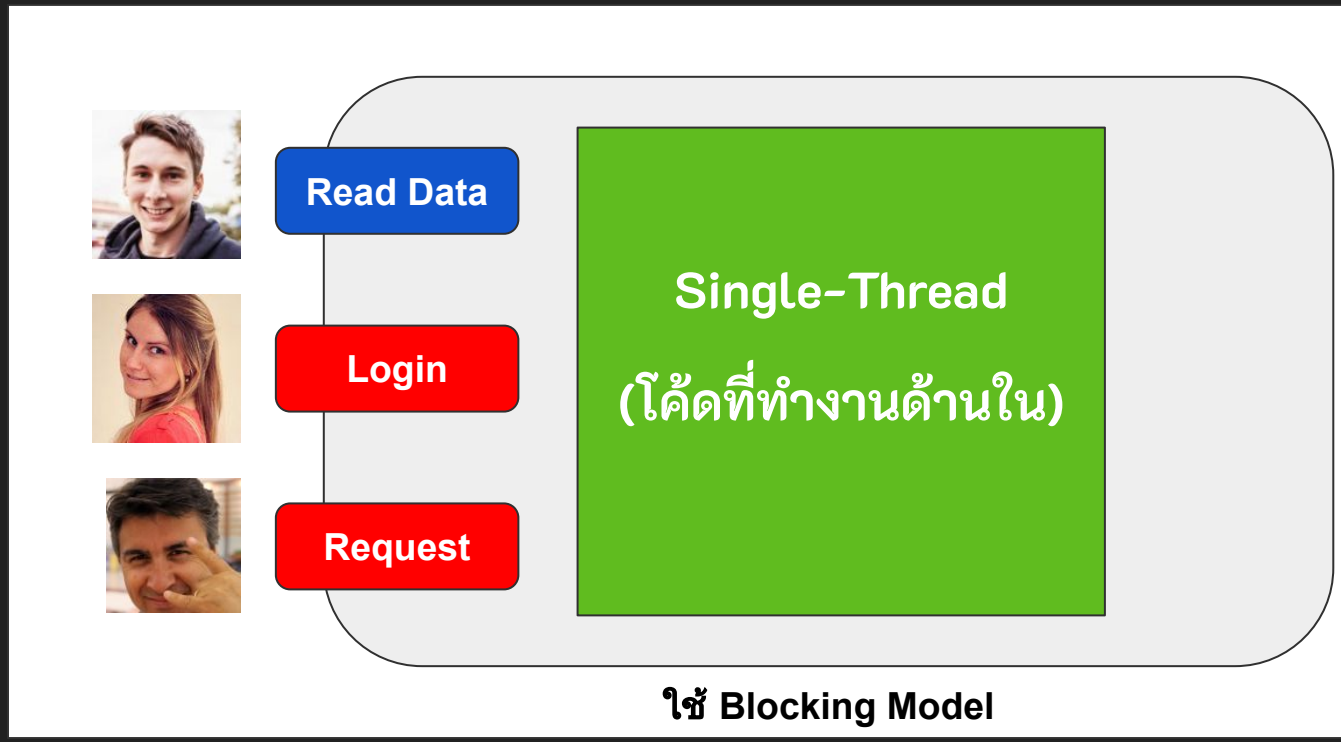
รูปแบบการทำงานของ Node.js



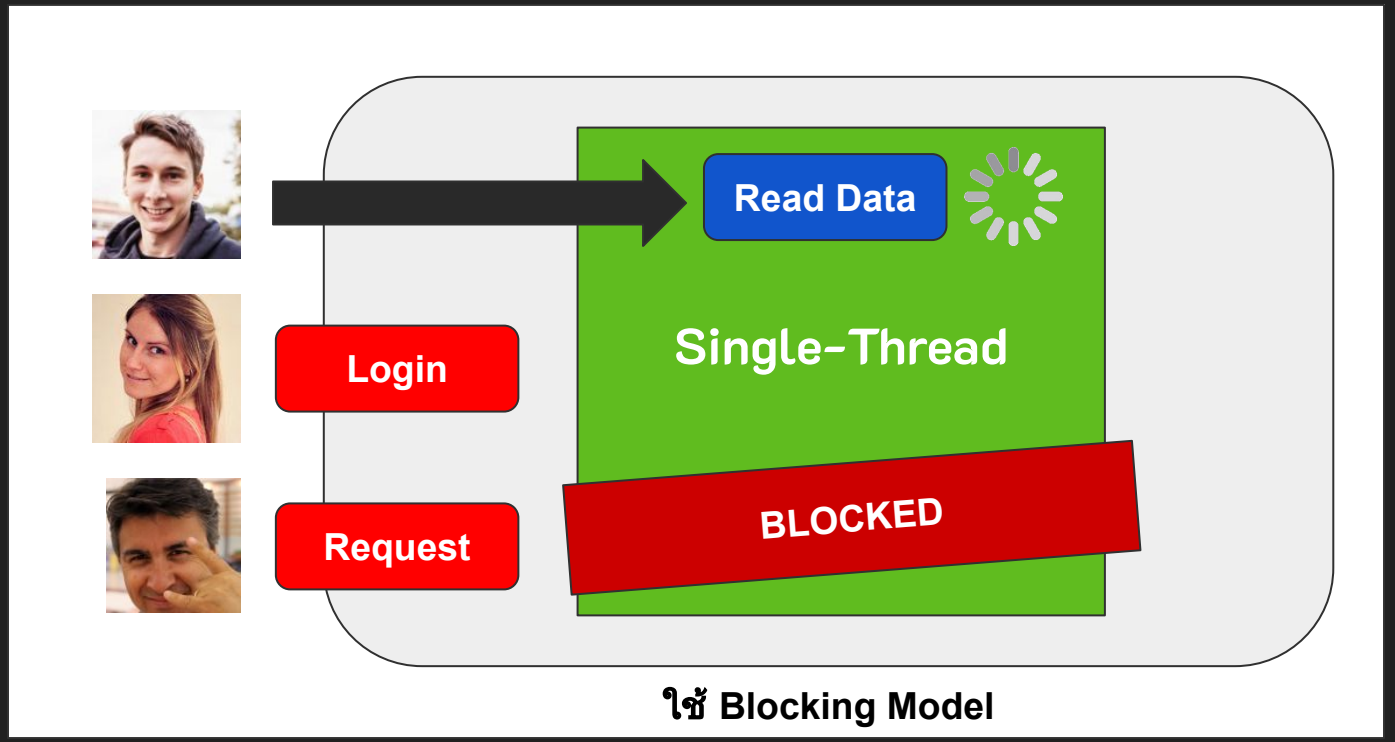
รูปแบบการทำงานของ Node.js



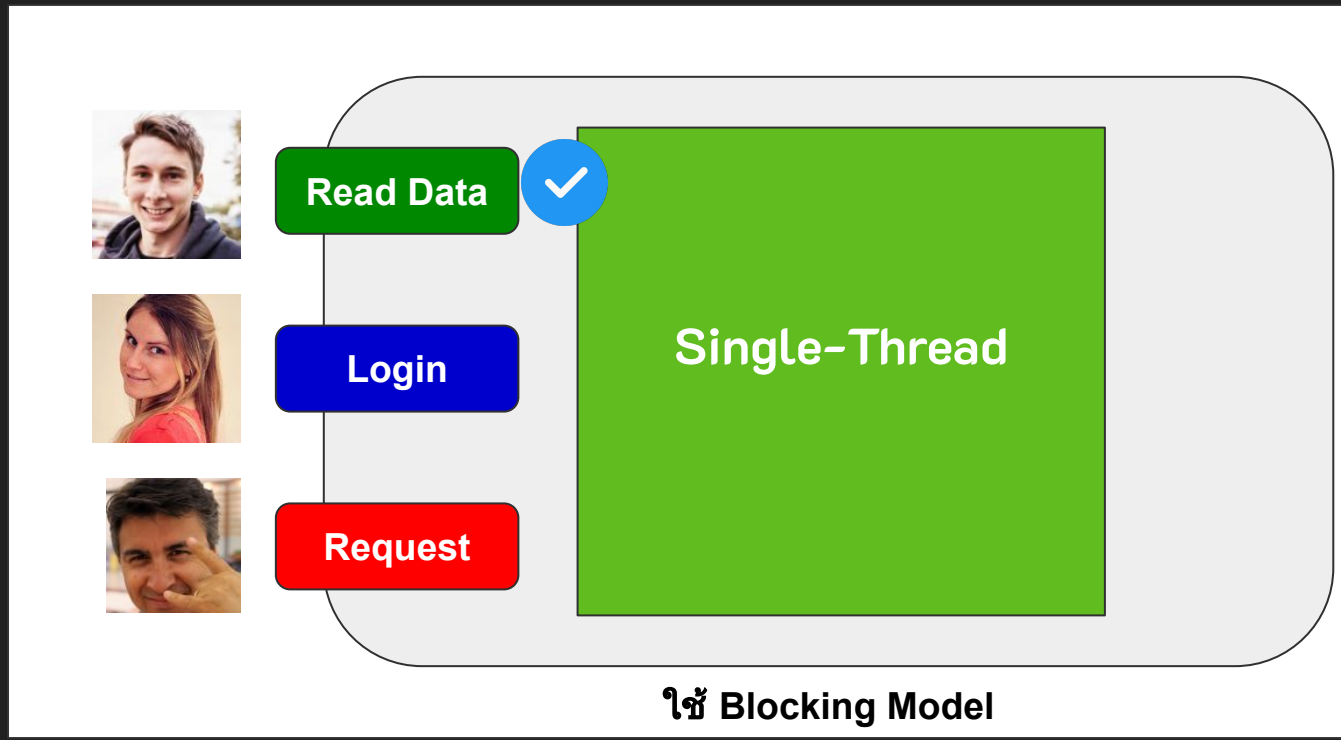
รูปแบบการทำงานของ Node.js



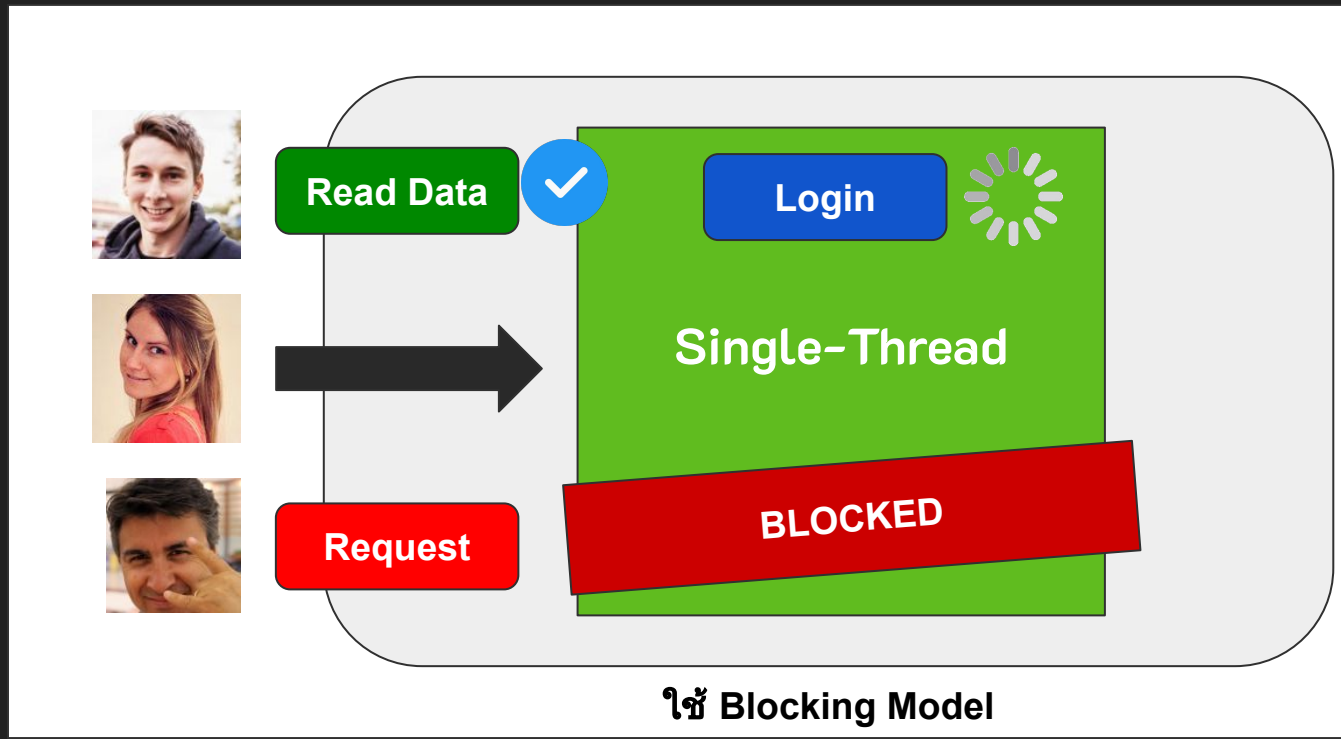
รูปแบบการทำงานของ Node.js



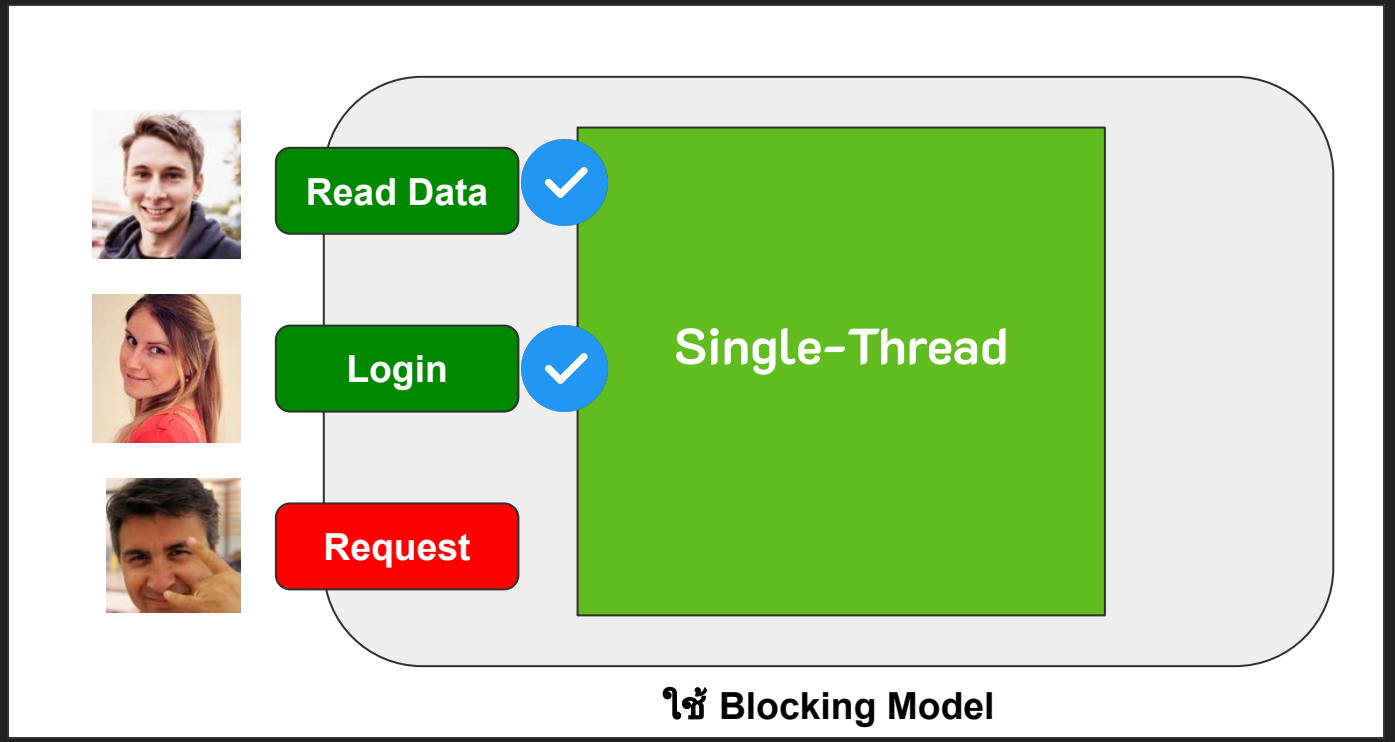
รูปแบบการทำงานของ Node.js



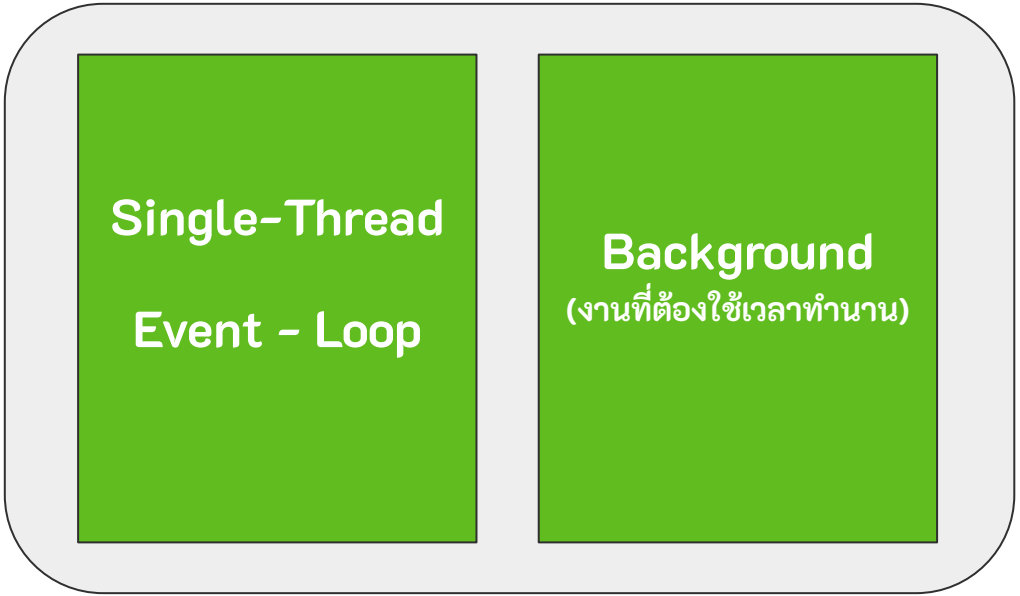
รูปแบบการทำงานของ Node.js



รูปแบบการทำงานของ Node.js



รูปแบบการทำงานของ Node.js

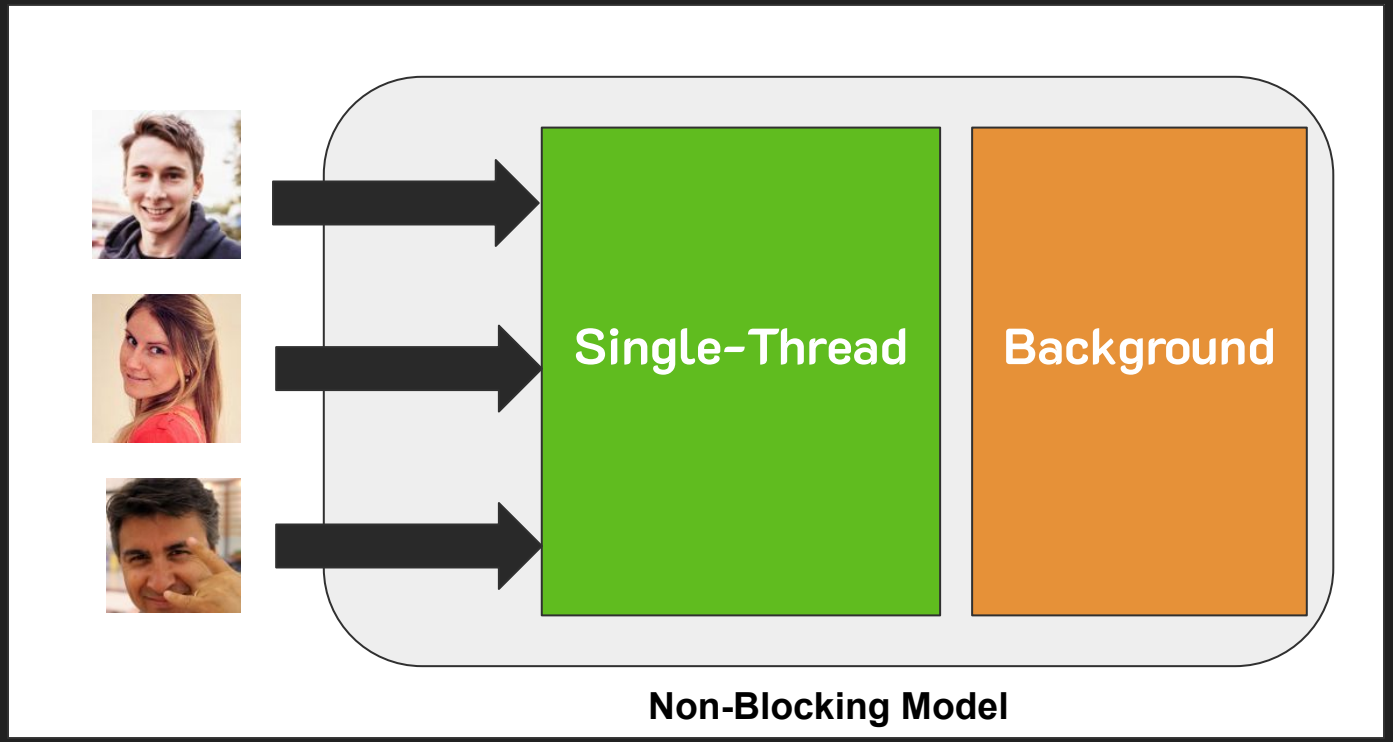


Non-Blocking Model

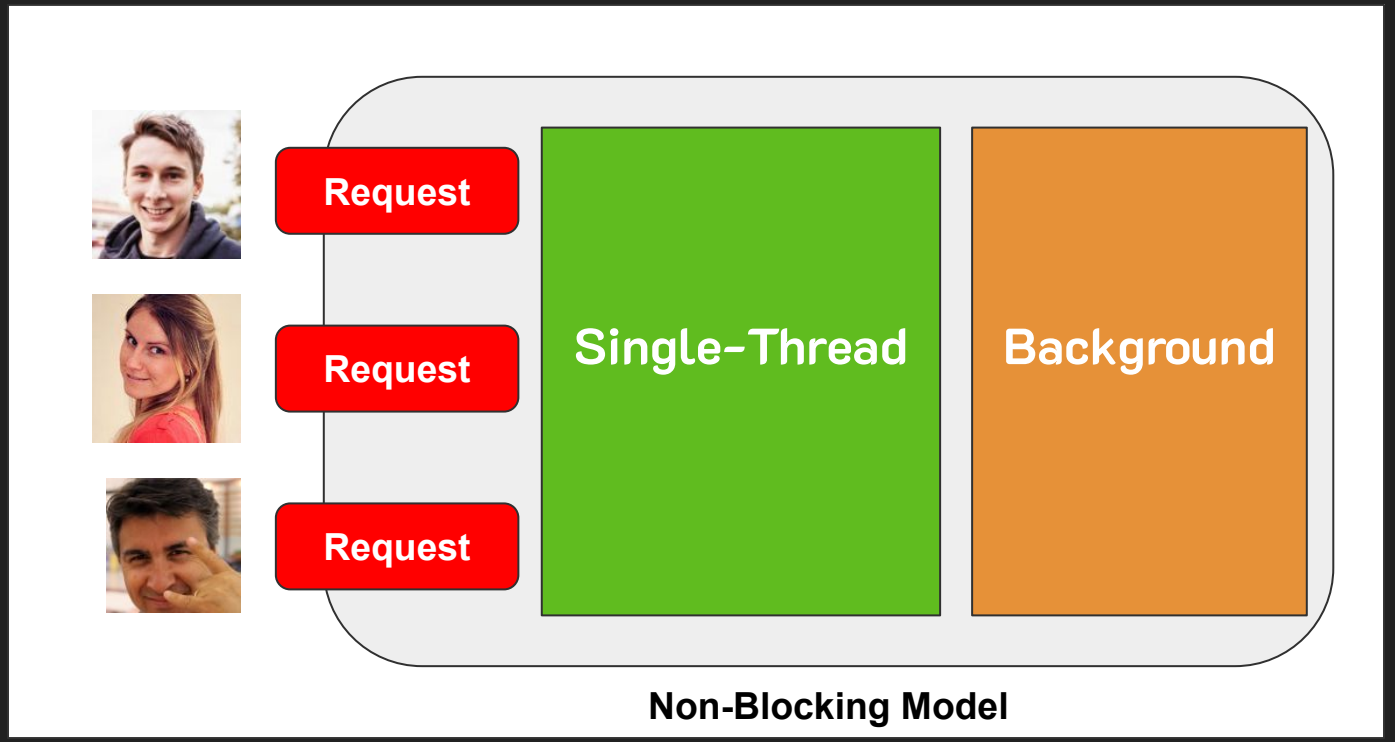
Event Loop คืออะไร

- 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

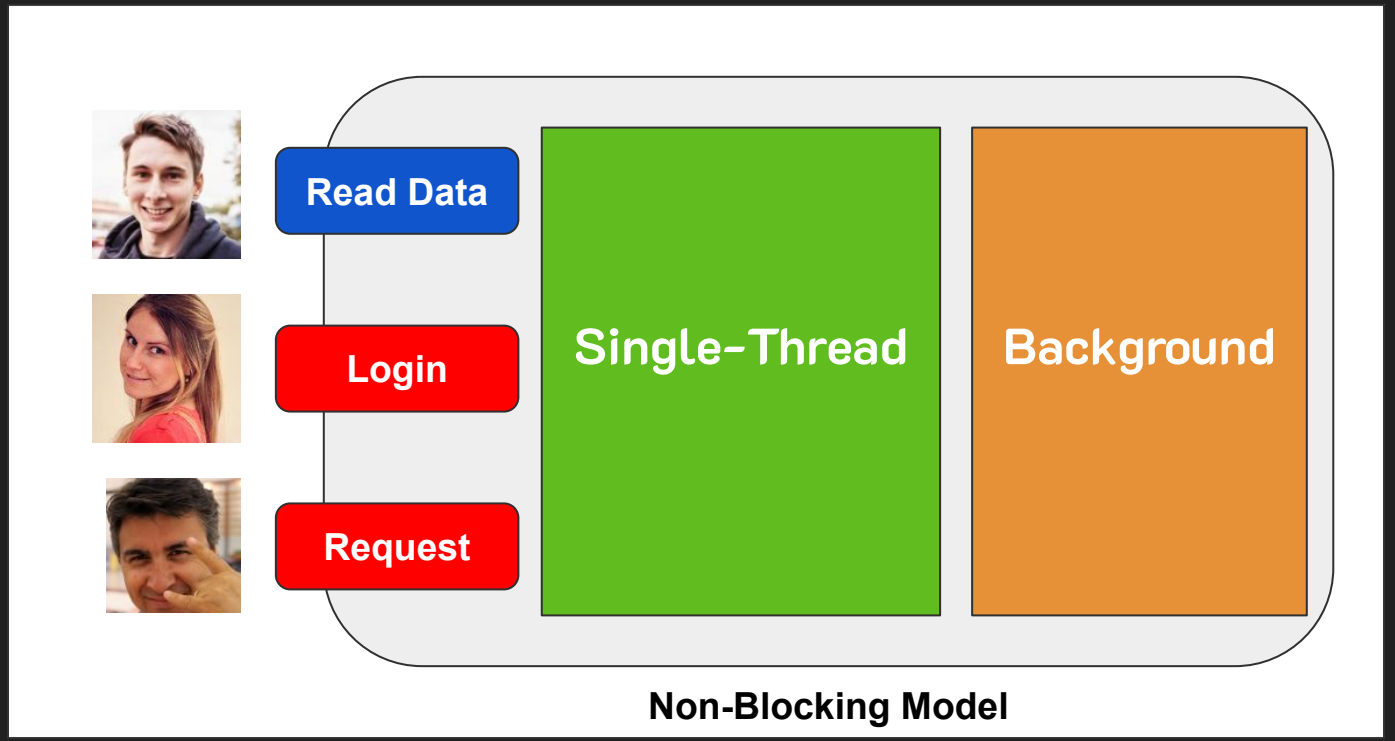
รูปแบบการทำงานของ Node.js



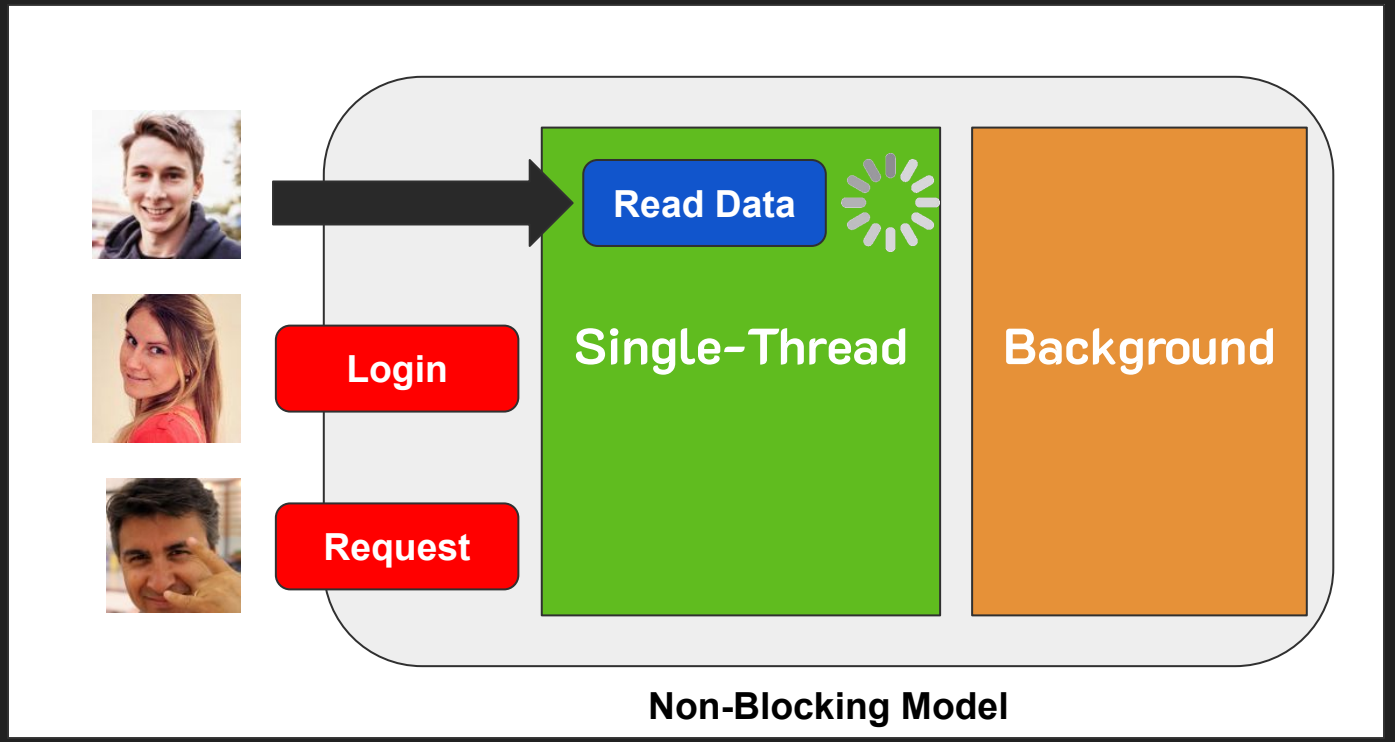
รูปแบบการทำงานของ Node.js



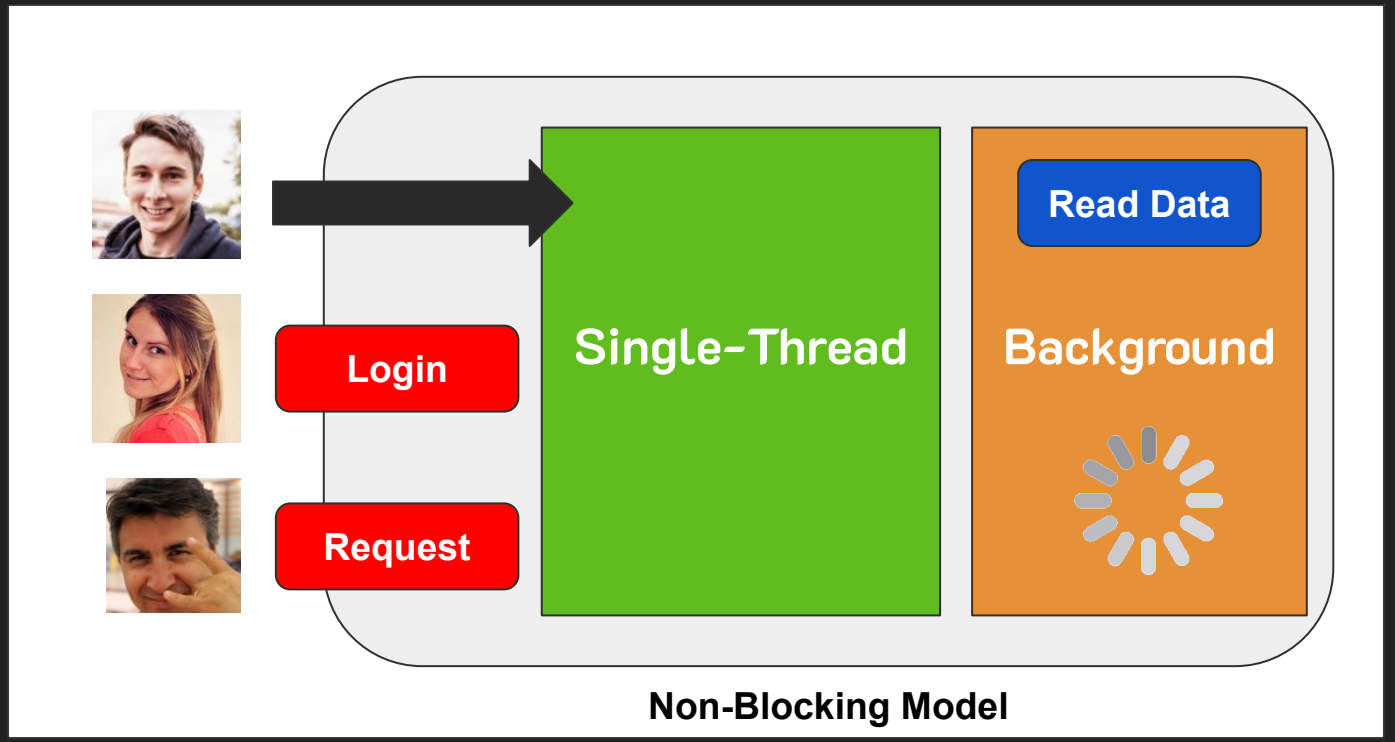
รูปแบบการทำงานของ Node.js



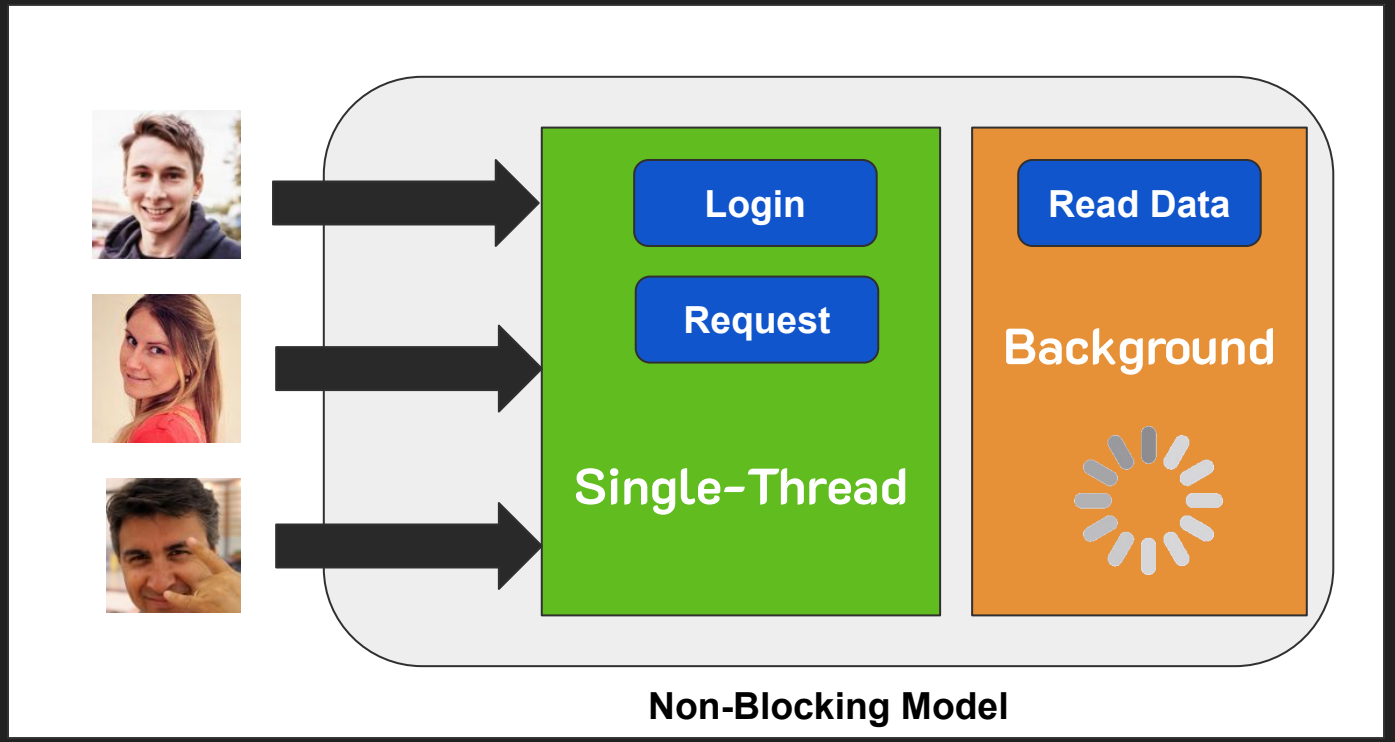
รูปแบบการทำงานของ Node.js



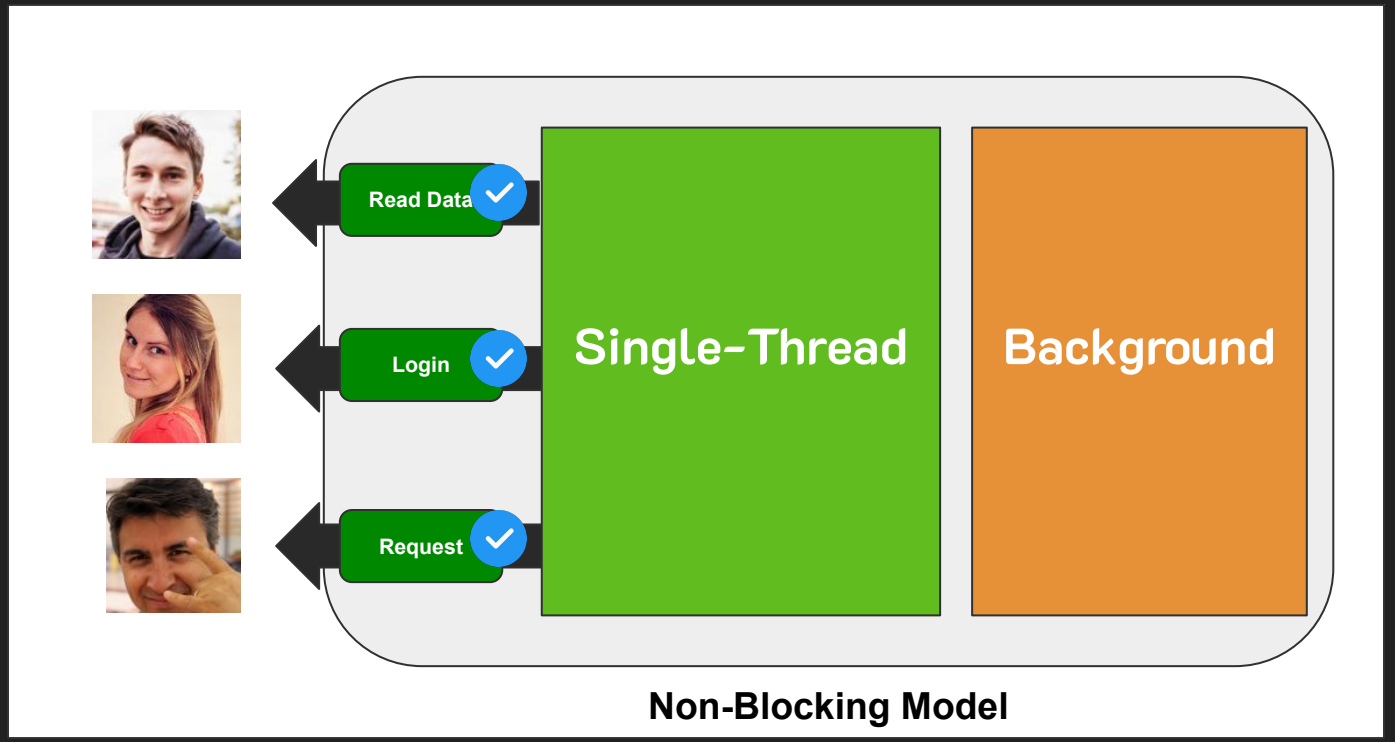
รูปแบบการทำงานของ Node.js



รูปแบบการทำงานของ Node.js



รูปแบบการทำงานของ Node.js



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

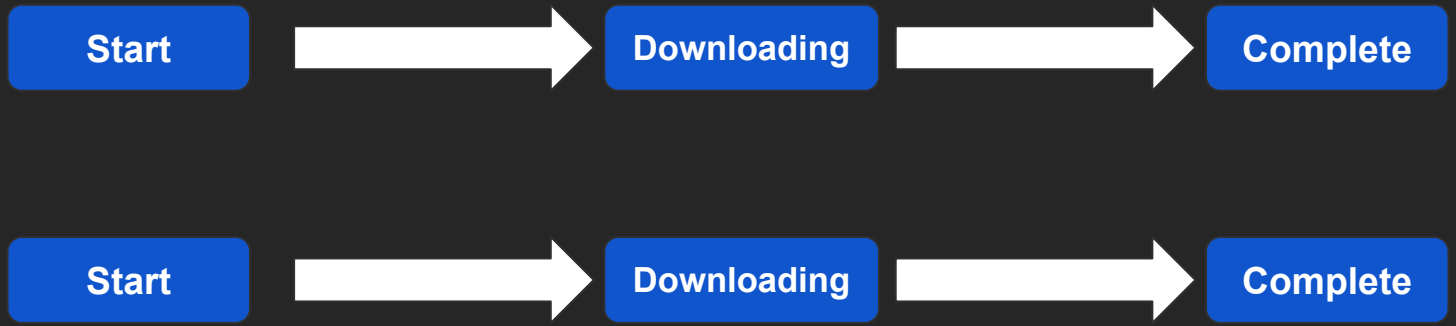
ตัวอย่างโค้ด

```
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

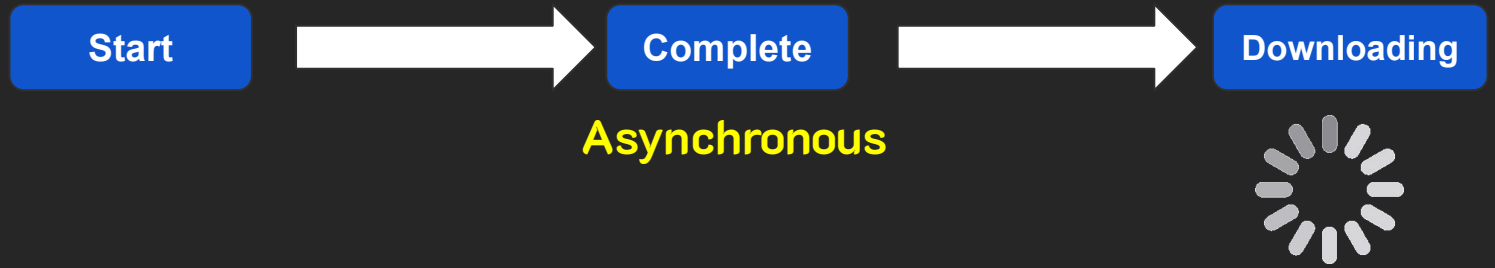
รู้จักกับ CallBack

CallBack

อาศัย CallBack Function คือ ฟังก์ชั่นที่จะถูกเรียกใช้งานเมื่ออีกฟังก์ชั่นทำงานเสร็จ

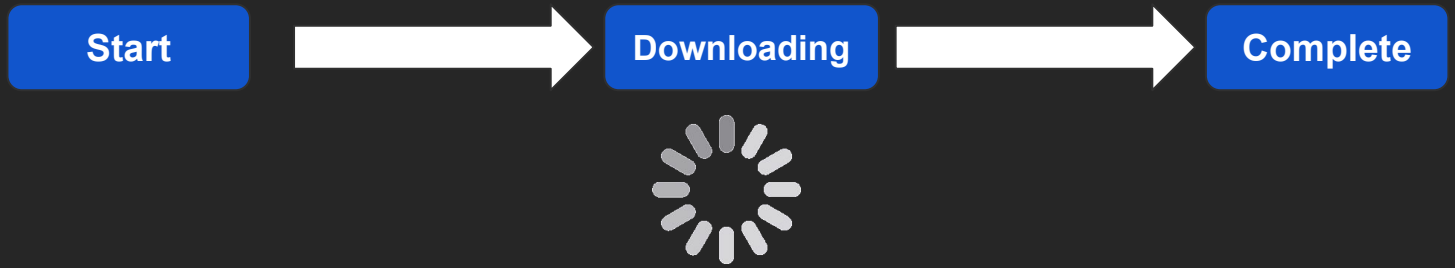
CallBack

อาศัย CallBack Function คือ ฟังก์ชั่นที่จะถูกเรียกใช้งานเมื่ออีกฟังก์ชั่นทำงานเสร็จ



CallBack

อาศัย CallBack Function คือ ฟังก์ชั่นที่จะถูกเรียกใช้งานเมื่ออีกฟังก์ชั่นทำงานเสร็จ



CallBack

อาศัย CallBack Function คือ ฟังก์ชั่นที่จะถูกเรียกใช้งานเมื่ออีกฟังก์ชั่นทำงานเสร็จ

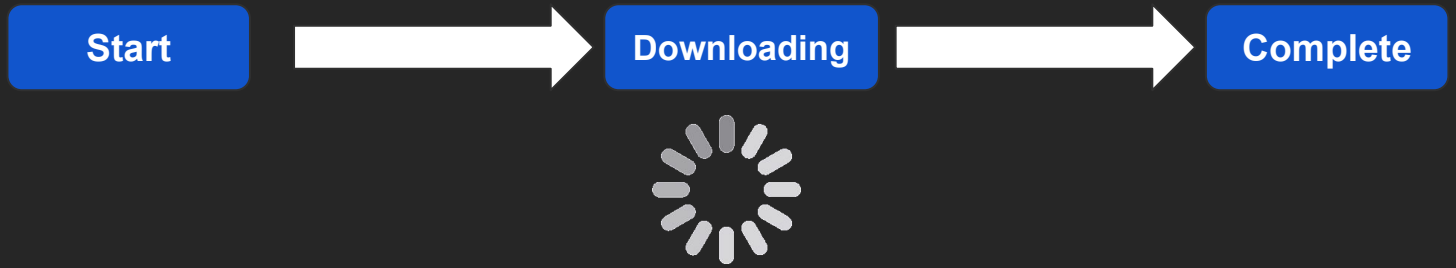
CallBack มาควบคุมลำดับการทำงาน

ทบทวน CallBack

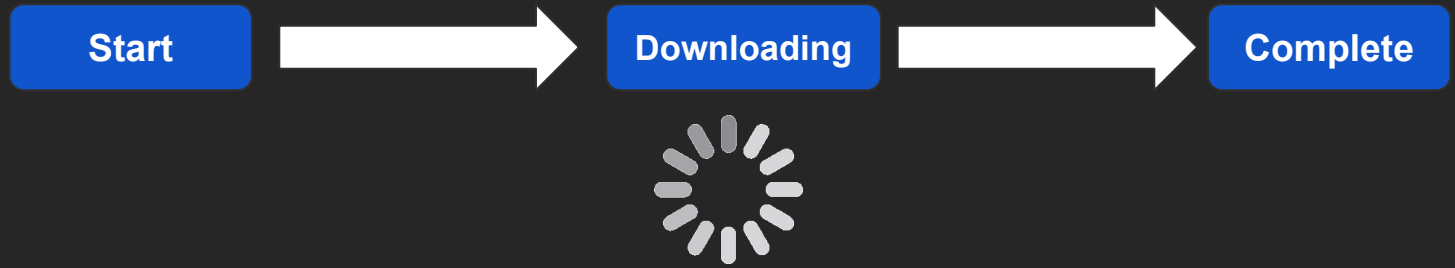
เขียนโค้ดโปรแกรมบวกเลขแบบปกติและลดรูป

1.ใช้ CallBack มาควบคุมการทำงานแบบ Asynchronous ให้คำสั่งทำงานตามลำดับที่ต้องการ

ส่ง url ของไฟล์เป็นพารามิเตอร์เข้าไปทำงาน



กำหนด url ไฟล์ตัวอย่างให้กับการ Download ไฟล์ในฟังก์ชั่น
โดยการโหลดไฟล์ตั้งแต่ไฟล์ที่ 1 ไปจนถึงไฟล์ที่ 5 เรียงลำดับ



CallBack Hell

การใช้ Callback อาจก่อให้เกิดปัญหาที่เรียกว่า Callback Hell ซึ่งก็คือการนำฟังก์ชั่น Callback มาซ้อนกันไปเรื่อยๆแล้วส่งผลทำให้เกิดความสับสนกับการเช็คค่าลำดับการทำงานเลยเกิดวิธีแก้ปัญหาดังกล่าวโดยการใช้ “Promise”

รู้จักกับ Promise

Promise (พรอมิส)

Promise ถูกนำมาใช้งานเกี่ยวกับการทำงานแบบ Asynchronous คือ ให้รอในระหว่างที่ผลลัพธ์ยังไม่เกิดขึ้น ใช้กับงานที่มีลักษณะการหน่วงเวลา (Delay) หรืองานที่ต้องทำเบื้องหลัง แล้วจะมีผลเกิดขึ้นในเวลาต่อมาและถูกนำมาใช้แก้ปัญหา CallBack Hell

Promise (พรอมิส)

การสร้าง Promise

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


})
```

CallBack Function - ใช้กำหนดการกระทำบางอย่าง

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("ดาวน์โหลดเสร็จเรียบร้อย");
    } else {
        reject("เกิดข้อผิดพลาดระหว่าง Download");
    }
});
```

Promise (พรอมิส)

```
let connect = true
const downloading = new Promise(function (resolve, reject) {
    setTimeout(()=>{
        if (connect) {
        resolve("ดาวน์โหลดเสร็จเรียบร้อย");
        } 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

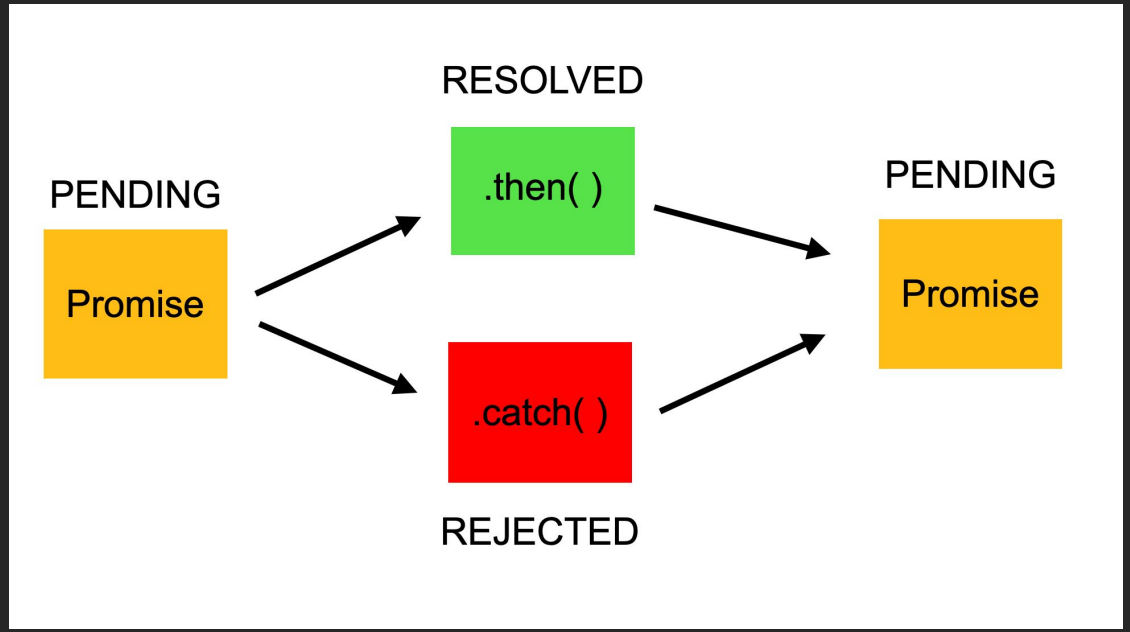
เมธอด then() , catch() , finally()

```
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)



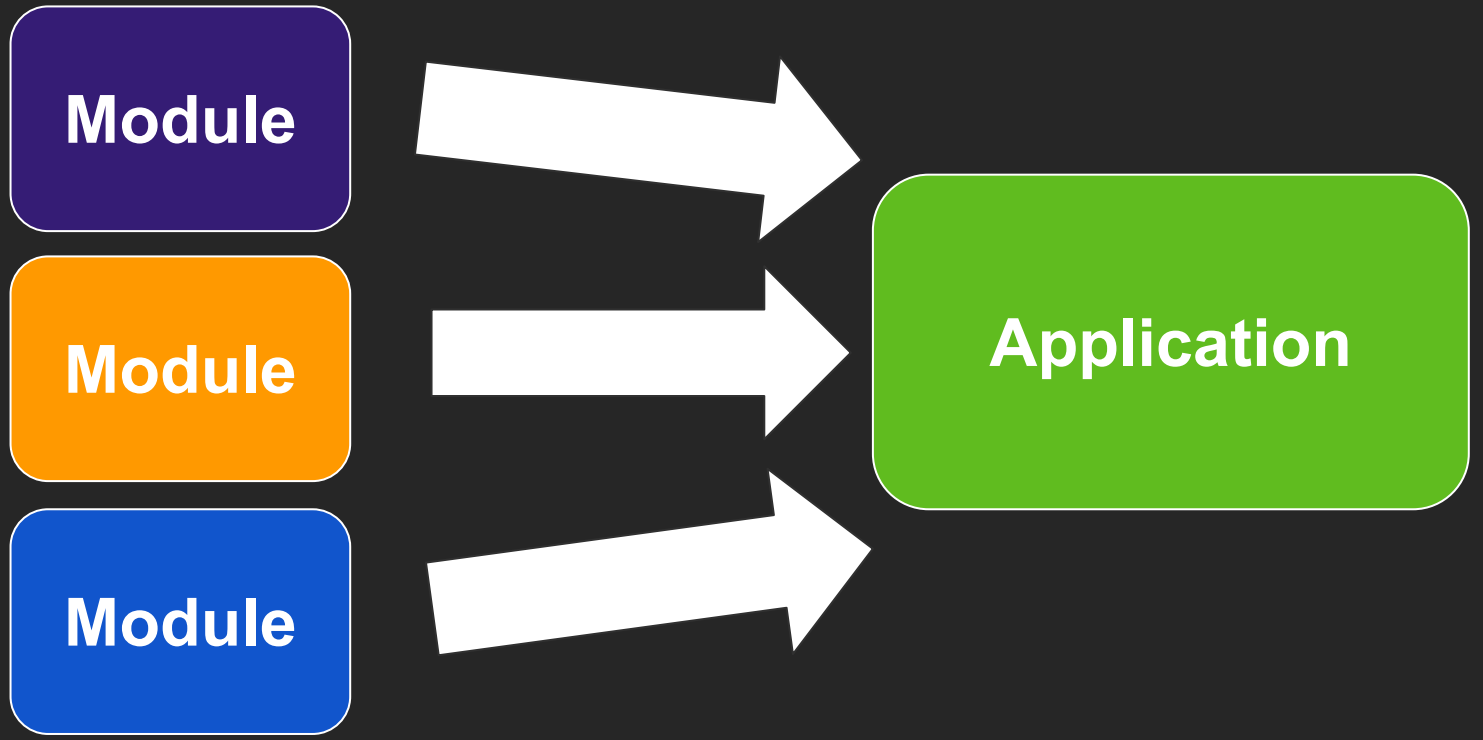
https://www.freecodecamp.org/news/content/images/2020/06/Ekran-Resmi-2020-06-06-12.21.27.png

โมดูล (Module)

Module คือ ไฟล์ที่จัดเก็บโค้ดของ JavaScript ซึ่งประกอบด้วยการทำงานต่างๆ ได้แก่ ตัวแปร ฟังก์ชั่น , 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>)

const util = require(‘mymodule’)

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)
```

การอ่านและเขียนไฟล์

การอ่านและเขียนไฟล์แบบ Synchronous (Blocking)

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

fs.writeFileSync('ตำแหน่งไฟล์',data)

console.log("เขียนไฟล์สำเร็จ!")
```

การอ่านและเขียนไฟล์

จากโค้ดดังกล่าว ถ้ายังอ่านไฟล์ไม่เสร็จจะไปทำงานในส่วนของการเขียนไฟล์ต่อไม่ได้ เนื่องจากเป็นรูปแบบ Synchronous แล้วถ้าไฟล์มีขนาดใหญ่ก็จะต้องใช้เวลาอ่านไฟล์นานและการทำงานของระบบก็จะช้าไปด้วย สามารถจัดการปัญหาดังกล่าวโดยใช้รูปแบบ Asynchronous

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)

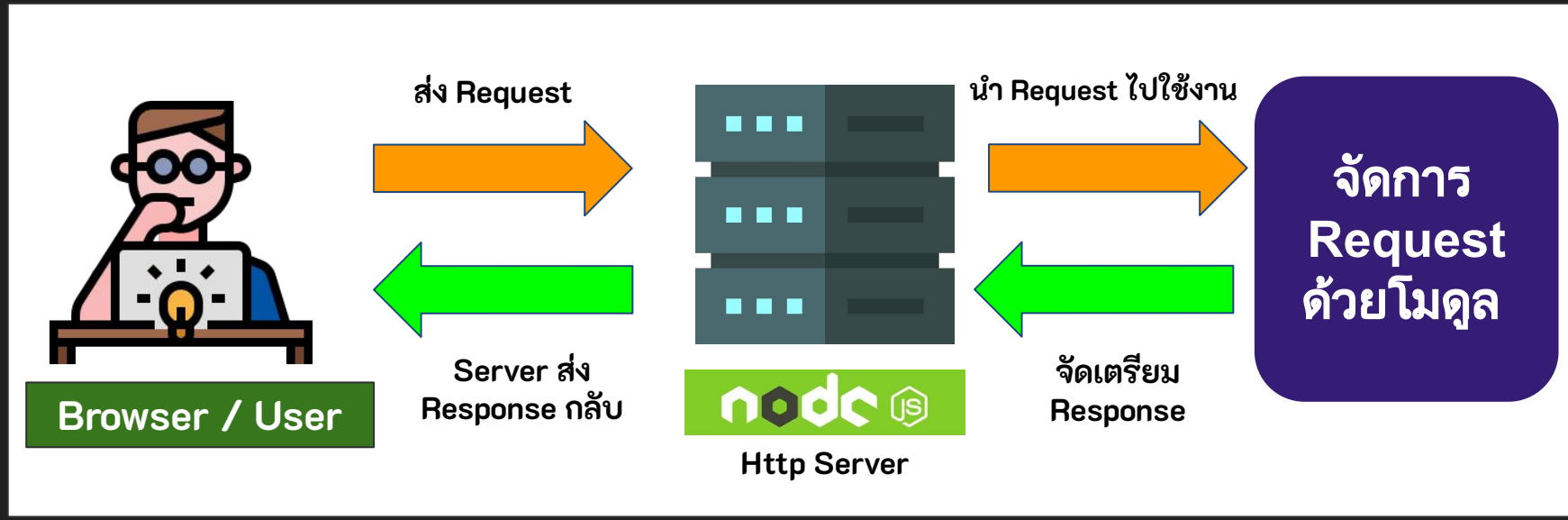
```
fs.readFile('ตำแหน่งไฟล์',encoding,(err,data){
    fs.writeFile('ตำแหน่งไฟล์',data,(err)=>{
        if(err) return  console.log(err)
        console.log("เขียนไฟล์เรียบร้อย")
    })
})
```

สร้าง Web Server ด้วย Node.js

คำศัพท์พื้นฐาน

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

แผนภาพการทำงาน



สร้างแอพของ 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

รันแอพด้วย Nodemon

ในกรณีที่มีการเปลี่ยนแปลงการทำงานในแอพที่พัฒนาด้วย Node.js ต้องทำการหยุดรันแอพโดยการกด Ctrl+C ทุกครั้งแล้วค่อยรันใหม่ด้วยคำสั่ง node index.js จึงจะเห็นการเปลี่ยนแปลงที่เกิดขึ้น ซึ่งมันทำให้เกิดความยุ่งยากเพราะต้อง restart แอพทุกครั้ง ซึ่งสามารถจัดการปัญหาดังกล่าวได้โดยใช้

nodemon

รันแอพด้วย Nodemon

nodemon คือ โมดูลที่จะคอยติดตามการเปลี่ยนแปลงที่เกิดขึ้นกับไฟล์ JavaScript (.js) ทั้งหมดที่อยู่ในแอพ เมื่อใดที่ไฟล์ดังกล่าวมีการเปลี่ยนแปลงและถูกบันทึกไฟล์เกิดขึ้น nodemon ก็จะ restart แอพให้โดยอัตโนมัติ โดยที่ไม่ต้อง restart แอพเอง

การติดตั้ง

```
npm install nodemon
```

รันแอพด้วย Nodemon

การรันแบบเดิม

```
node index.js
```

การรันแบบใหม่

```
npx nodemon index.js
```

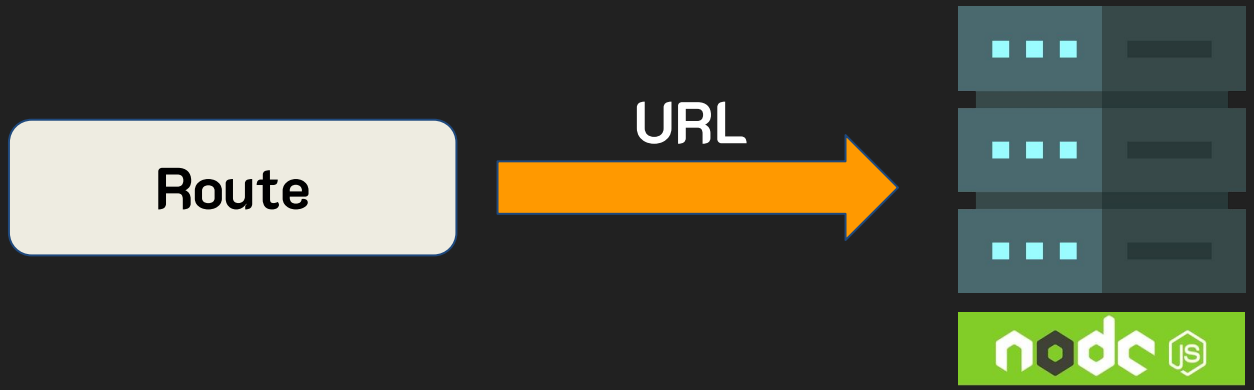
ตั้งค่าใน script

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

และรันโดยใช้

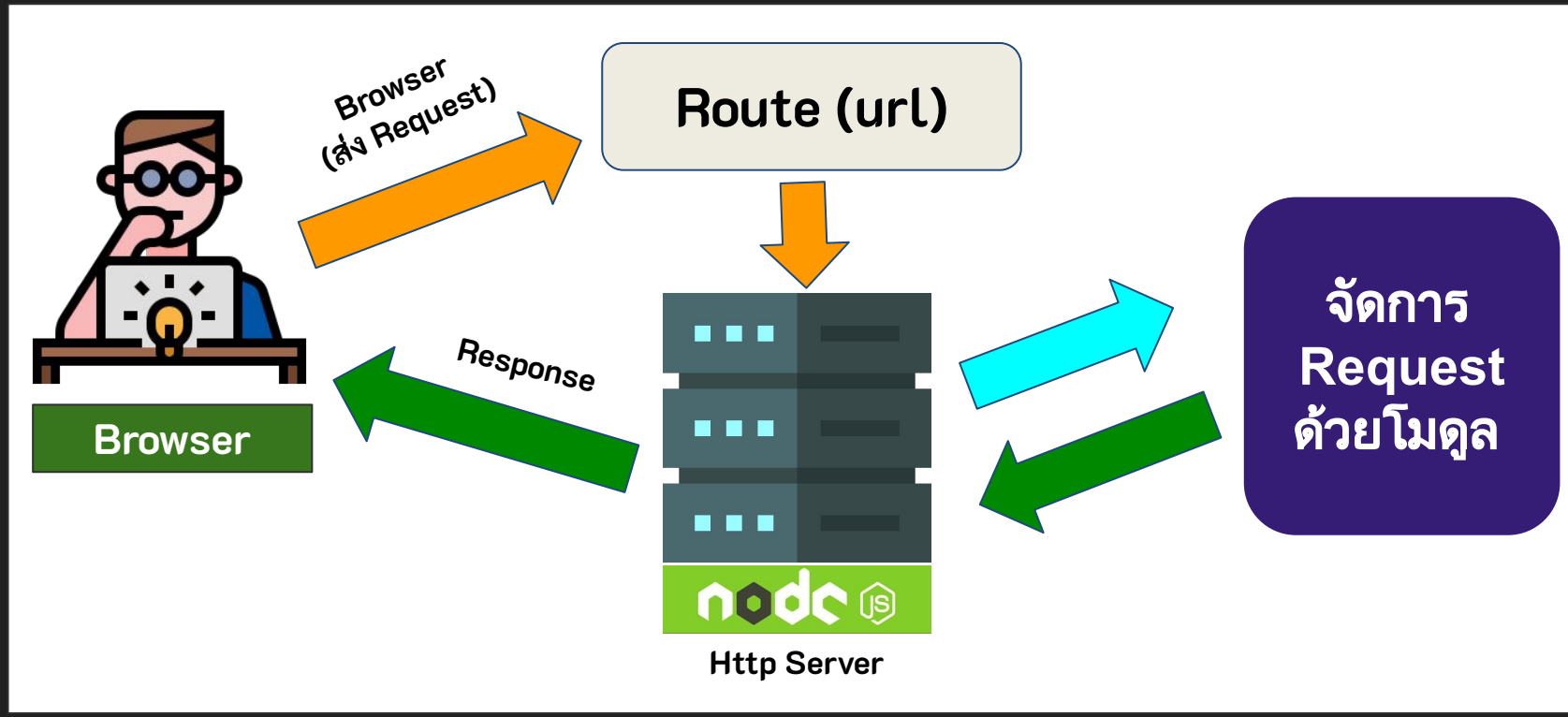
```
npm start
```

Routing



การกำหนดเส้นทางหรือ URL ในการอนุญาตให้เข้าถึงข้อมูล รวมไปถึงตรวจสอบ URL Request เพื่อจะได้กำหนดรูปแบบการทำงาน

แผนภาพการทำงาน



HTTP Status Code

เป็นรหัสที่บ่งบอกสถานะของ Request ตัวอย่างเช่น

- 200 Ok (ดำเนินการเสร็จสมบูรณ์)
- 201 Create (สร้างข้อมูลใหม่เรียบร้อย)
- 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 ฝั่ง Server จะมีความยุ่งยากในการทำระบบหลายๆ อย่าง เช่น

ระบบ Routing จากหัวข้อที่ผ่านมาต้องอ่าน path จาก URL และนำค่ามาเปรียบเทียบเพื่อแสดงผลไฟล์หรือเพจที่ต้องการ ซึ่งวิธีดังกล่าวนั้นค่อนข้างมีความยุ่งยากและไม่เหมาะกับการใช้งานในระบบที่ใหญ่ มีความซับซ้อนสูง จึงได้มีการพัฒนา Express.js ขึ้นเพื่อ จัดการปัญหาการทำ Web Server ด้วย Node.js นั่นเอง

รู้จักกับ Express.js

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

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

ติดตั้ง Express.js

คำสั่งที่ใช้

```
npm install express
```

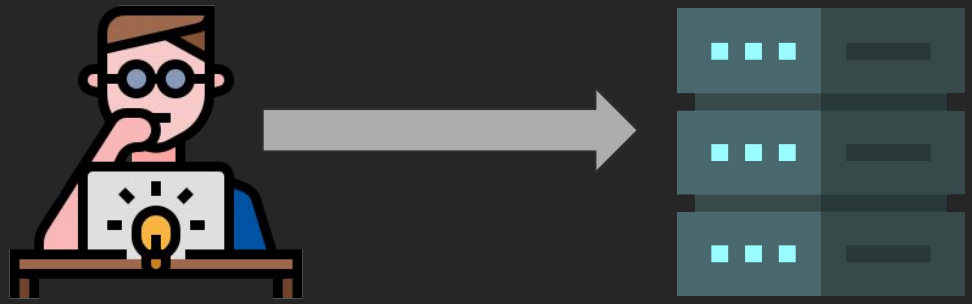
การนำ express มาใช้งาน

```
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



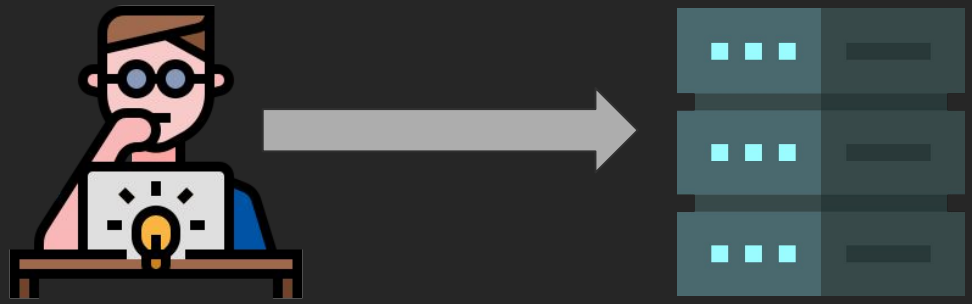
- ใช้คำสั่ง use (path เริ่มต้น)
- ใช้ http method (get ,post)

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

```
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)

})
```

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 ไปกำหนดเส้นทางในแอพ
```

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

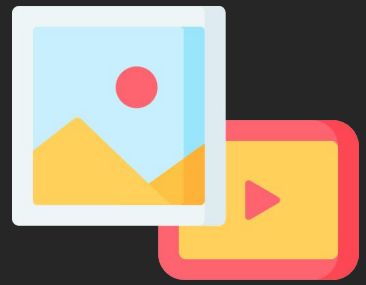
- response.redirect(path)
- response.redirect(URL)

จัดการ
Static File

Static File

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

หากนำไฟล์ดังกล่าวมารันก็จะได้ผลลัพธ์ตามที่ได้กำหนดไว้ในไฟล์หรือถ้ามีการเขียนโปรแกรมกำหนดเงื่อนไขขึ้นมาไฟล์นั้นก็จะไม่มีการเปลี่ยนแปลงเนื้อหาตามเงื่อนไขที่กำหนดขึ้น



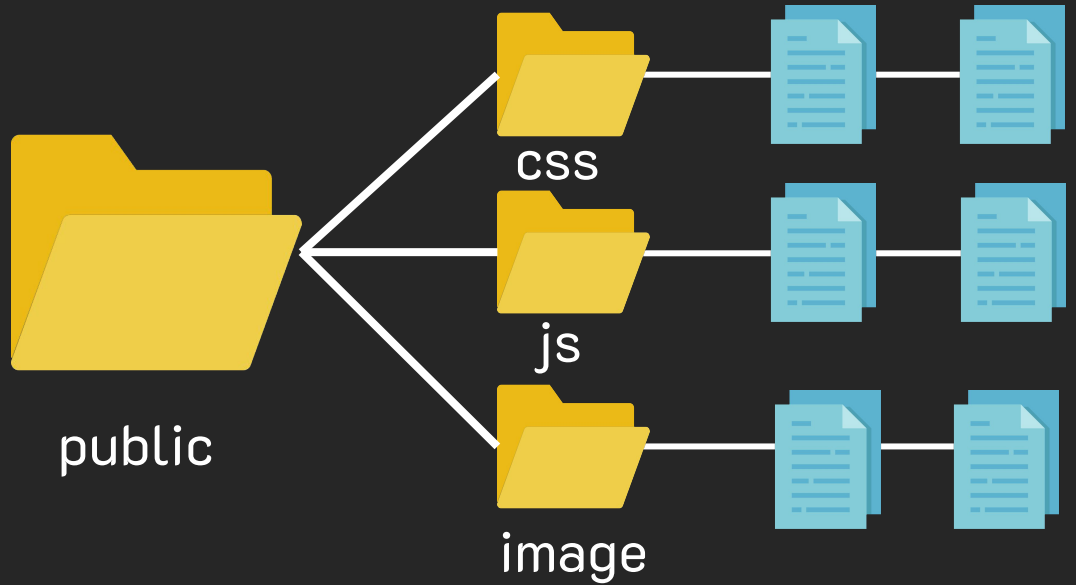
Static File

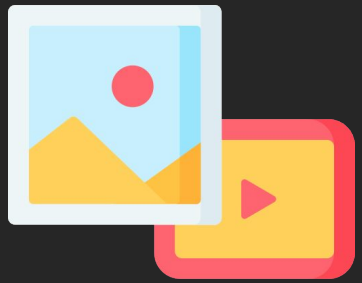
จากหัวข้อที่ผ่านมาเราจัดการ Static ไฟล์โดยให้จัดเก็บอยู่ในโฟลเดอร์ที่เราคิดหรือสร้างขึ้นเอง ยกตัวอย่า งเช่น โฟลเดอร์ Webpage , Template เป็นต้น ถ้าอยากเรียกใช้งานก็ดำเนินการผ่านโมดูล

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

การจัดเก็บ Static File

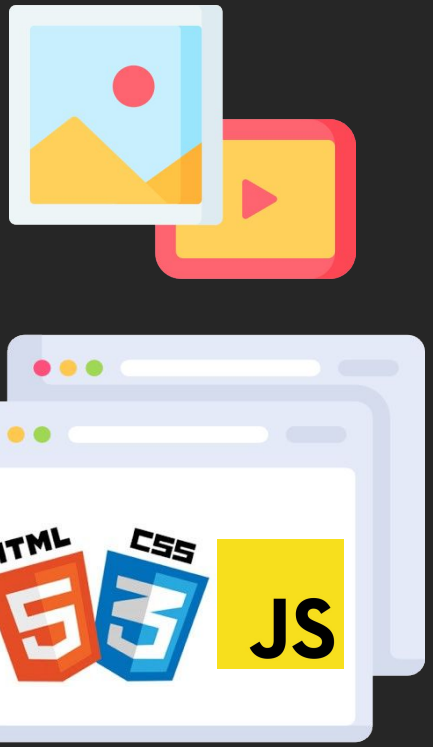
ไฟล์รูปแบบ static จะถูกเก็บลงในโฟลเดอร์ public





การตั้งค่าใช้งาน Static File

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

const app = express()

app.use(express.static('public'))

app.listen(8080,()=>{

})
```

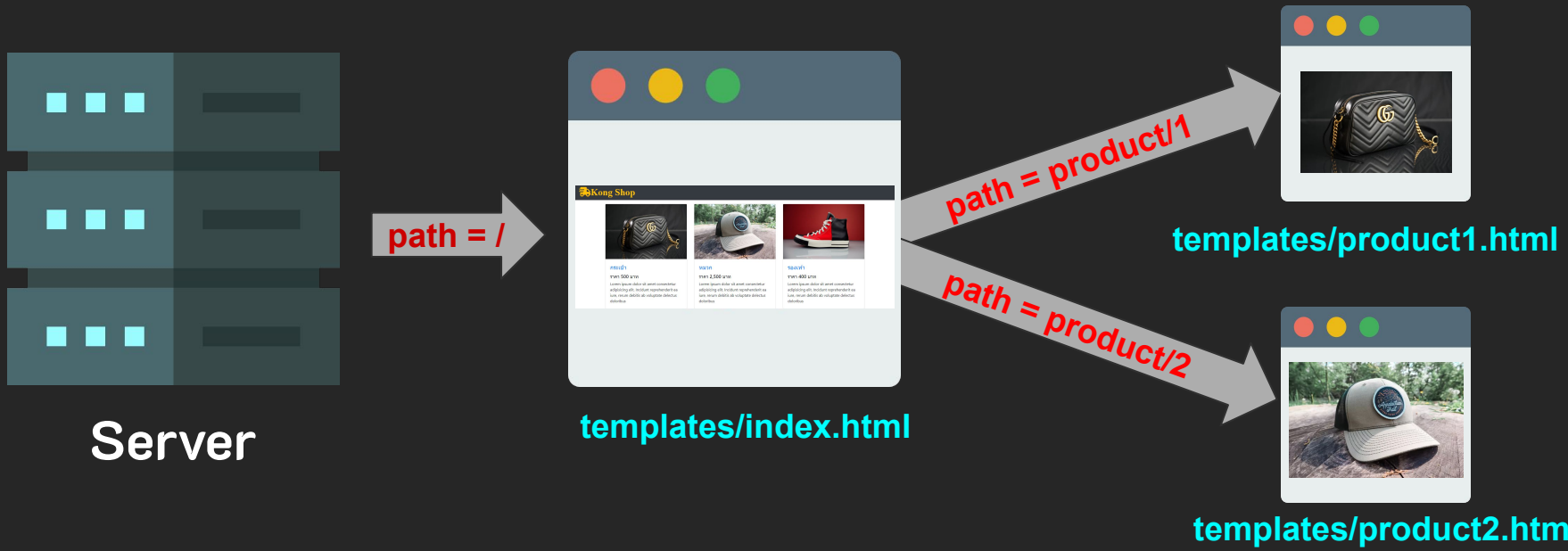


ทำไมต้องอ้างอิง Static File ?

Static File แทบจะไม่มีผลกับการทำงานฝั่ง Server เลย กล่าวคือ เป็นไฟล์ที่มีเนื้อหาตายตัว สามารถเข้าถึงและแสดงผลได้เลยโดยที่ไม่ต้องพึงการทำงานฝั่ง Server หรือ อาจจะไม่ต้องอาศัยระบบ Routing มาอ้างอิงการทำงานก็ได้ แค่รู้ที่อยู่ของไฟล์ในโฟลเดอร์ public ก็ทำงานได้เลย

กระบวนการทำงานแบบเดิม

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



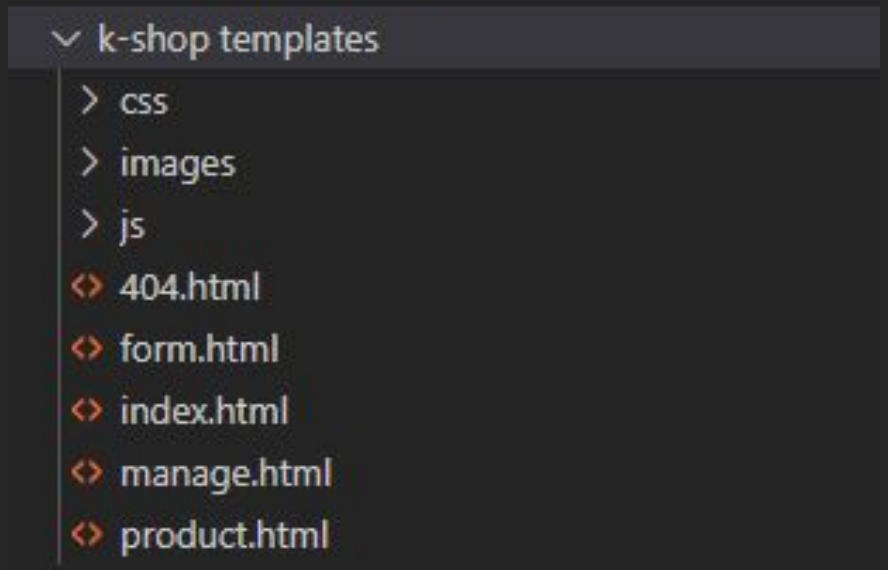
กระบวนการทำงานแบบเดิม

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

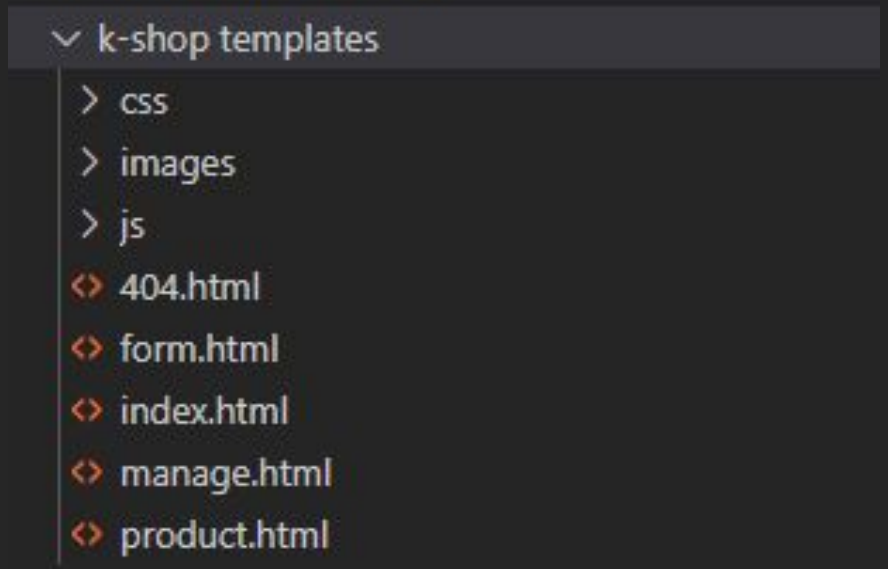
โหลด Template K-Shop (kongruksiam-shop)

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

โครงสร้างไฟล์ K-Shop



โครงสร้างไฟล์ K-Shop



css - สำหรับจัดเก็บไฟล์ .css

images - สำหรับเก็บไฟล์ภาพ

js - สำหรับเก็บไฟล์ .js

*****ทุกไฟล์จะถูกนำมาทำงานฝั่ง Frontend /Client ทั้งหมด***

View & Template Engine

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 หน้าตาแอพพลิเคชั่นเป็นส่วนที่ไว้ใช้แสดงผลข้อมูลผลลัพธ์จากการประมวลผลข้อมูลในหน้าเว็บร่วมกับ HTML , CSS ,JavaScript

ไฟล์ของ Template จะเก็บไว้ในโฟลเดอร์ที่มีชื่อว่า Views เท่านั้น (หรือโฟลเดอร์อื่นต้องไปตั้งค่าเพิ่มเติม)

View & Template

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 ที่มีรูปแบบโครงสร้างการเขียนคล้ายกับ HTML เพียงแต่สามารถเขียนคำสั่ง JavaScript เข้าไปใน HTML ได้โดยที่ไฟล์นั้นมีนามสกุล .ejs และสามารถติดตั้งโดยใช้คำสั่ง

```
npm install ejs
```

การตั้งค่าใช้งาน Template


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

//ตั้งค่าให้เป็น Engine สำหรับรันแอพ

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

การแสดง Template


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

    res.render("ชื่อ Template")

})
```

การส่งข้อมูลไปทำงานที่ Template

การส่งข้อมูลไปที่ Template


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

การแสดงข้อมูลที่ Template

<%= ชื่อ Propertie %>

ในกรณีส่งเป็นรูปแบบ 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-ส่งข้อมูลผ่านแบบฟอร์ม(Form)

HTML FORM

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

// Element

</form>
```

รูปแบบการรับส่งข้อมูล

method=”get หรือ post” (รูปแบบการส่งข้อมูล)

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

รูปแบบการรับส่งข้อมูล

action = “path” ระบุ path ปลายทางเพื่อรับข้อมูลที่ส่งไปจากฟอร์ม

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

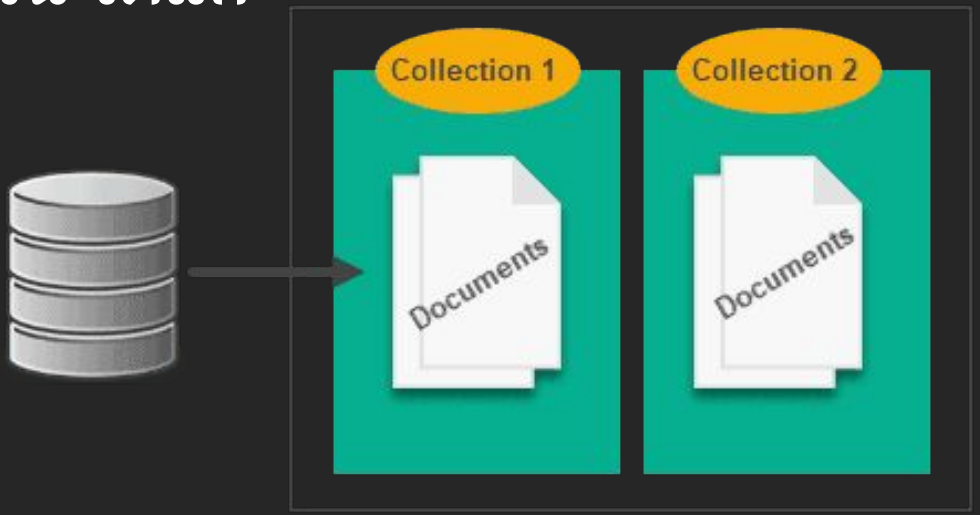
MongoDB คืออะไร

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

โครงสร้างการจัดเก็บข้อมูลของ MongoDB

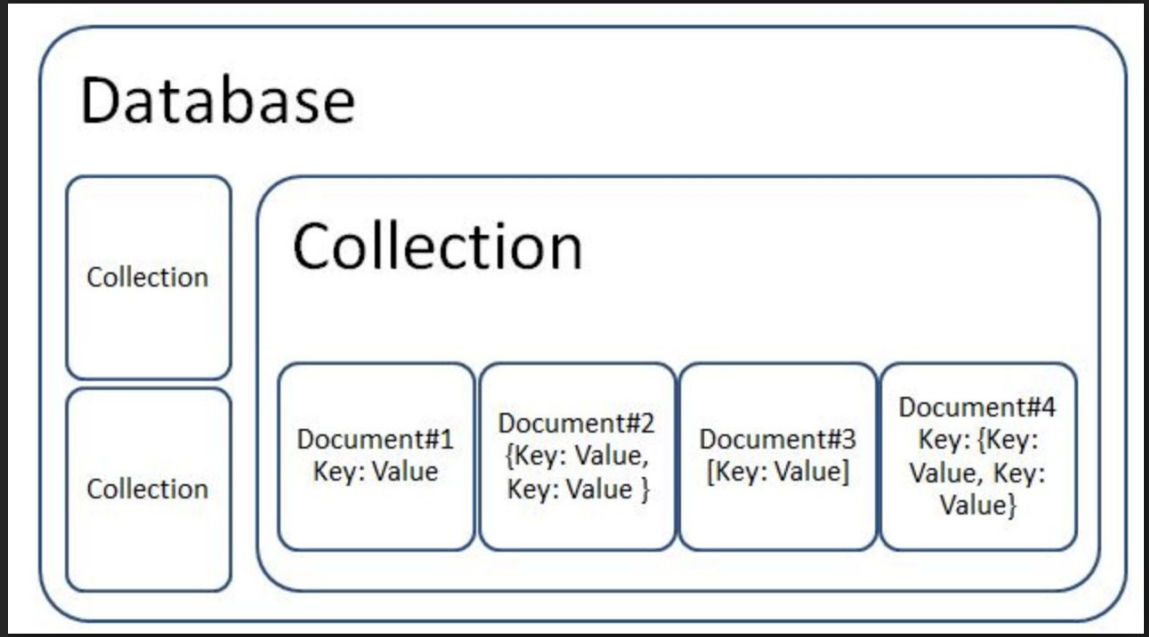
มีองค์ประกอบอยู่ 3 ส่วน ได้แก่

- Database
- Collection
- Documents



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

โครงสร้างการจัดเก็บข้อมูลของ MongoDB



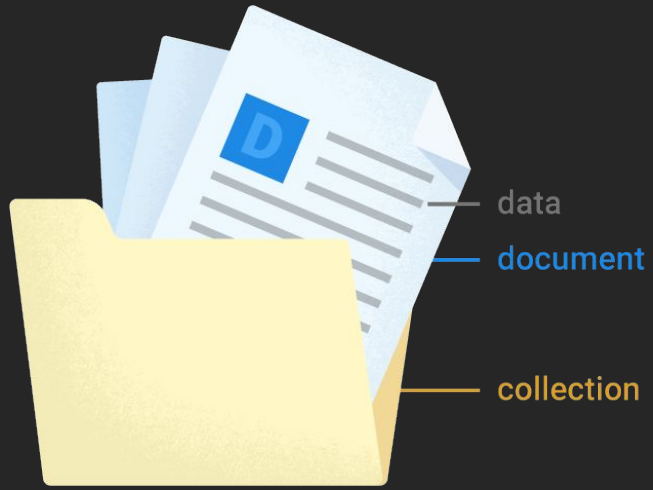
https://miro.medium.com/max/1313/1*vhKORnX6ZQ5HNUaeqxbQGg.png

โครงสร้างการจัดเก็บข้อมูลของ MongoDB

- Database (ฐานข้อมูล) เป็นส่วนที่ใช้เก็บ Collection หรือชุดข้อมูล

โครงสร้างการจัดเก็บข้อมูลของ MongoDB

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




mongoose
elegant mongodb object modeling for node.js

เชื่อมต่อ MongoDB ด้วย Mongoose

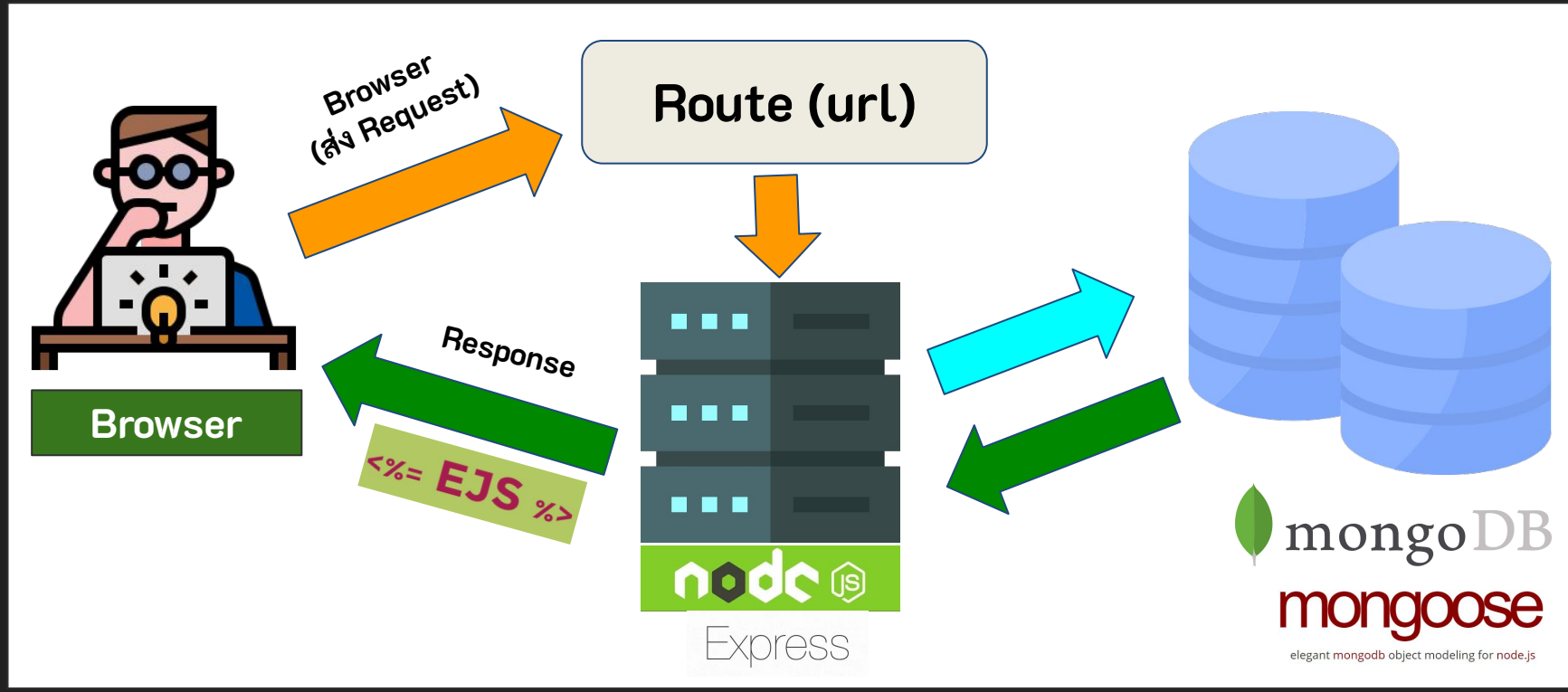
ถ้าต้องการให้โปรเจคเชื่อมโยงกับฐานข้อมูล MongoDB ต้องดำเนินการผ่านตัวกลางที่เรียกว่า Driver ในหัวข้อนี้จะใช้เครื่องมือตัวหนึ่งที่มีชื่อว่า

Mongoose

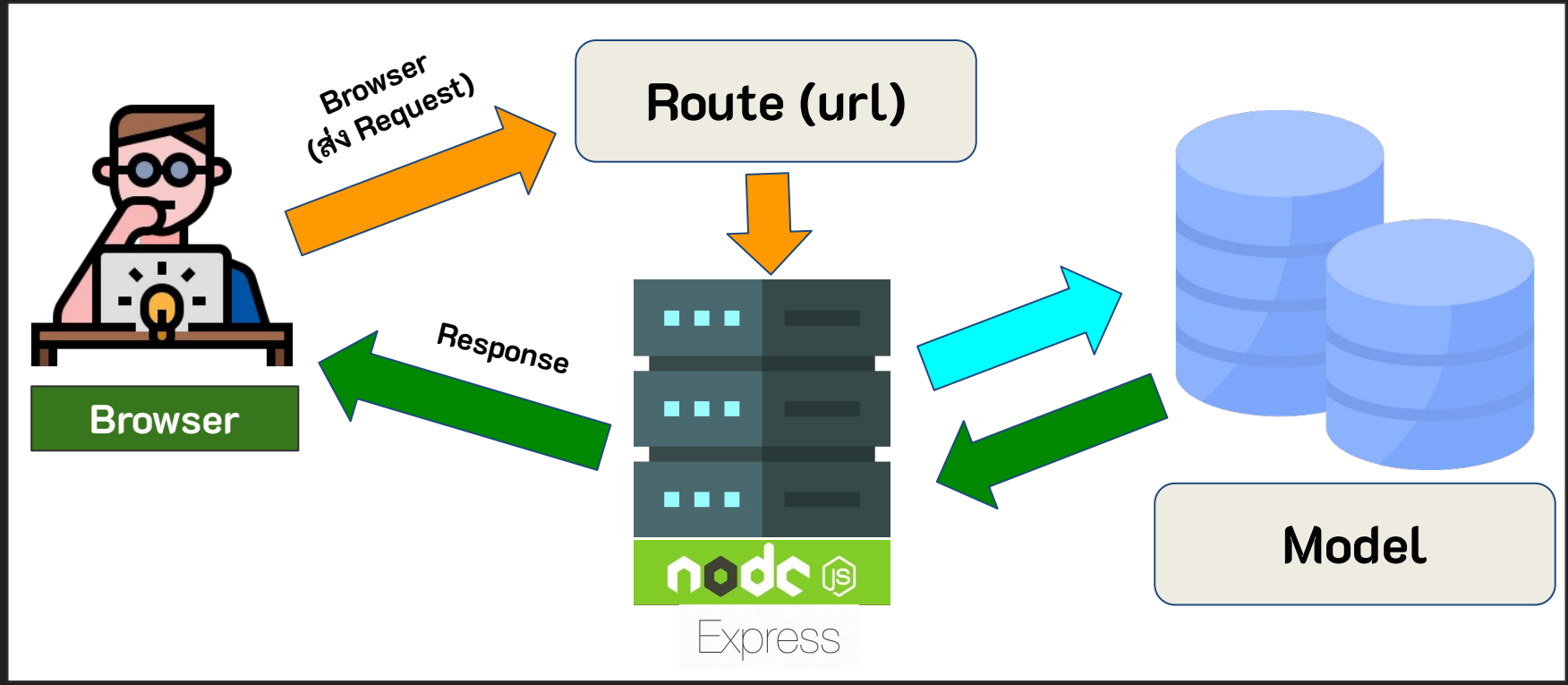
Mongoose คืออะไร

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

ภาพรวมระบบ


Browser
(ส่ง Request)
Route (url)
Browser
Response
<%= EJS %>
Express
mongoDB
mongoose
elegant mongodb object modeling for node.js

ภาพรวมระบบ


Browser
(ส่ง Request)
Route (url)
Response
Browser
Express
Model

การสร้างโมเดล

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

ติดตั้ง Mongoose

- npm install mongoose

โครงสร้างคำสั่ง

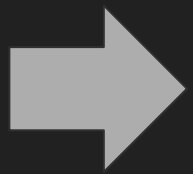
mongoose.connect(‘mongodb://<hostname:port>/<database>’)

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(model,data){
        model.save(data)
}
```

อัพโหลดไฟล์ด้วย Multer

ประกอบด้วย 5 ขั้นตอนดังนี้

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

อัพโหลดไฟล์ด้วย Multer

ติดตั้ง Multer

- npm install multer

อัพโหลดไฟล์ด้วย Multer

ตั้งค่า Form

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

อัพโหลดไฟล์ด้วย 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
    }
});
```

อัพโหลดไฟล์ด้วย Multer

กำหนดการอัพโหลดไฟล์

```
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')
    })
});
```

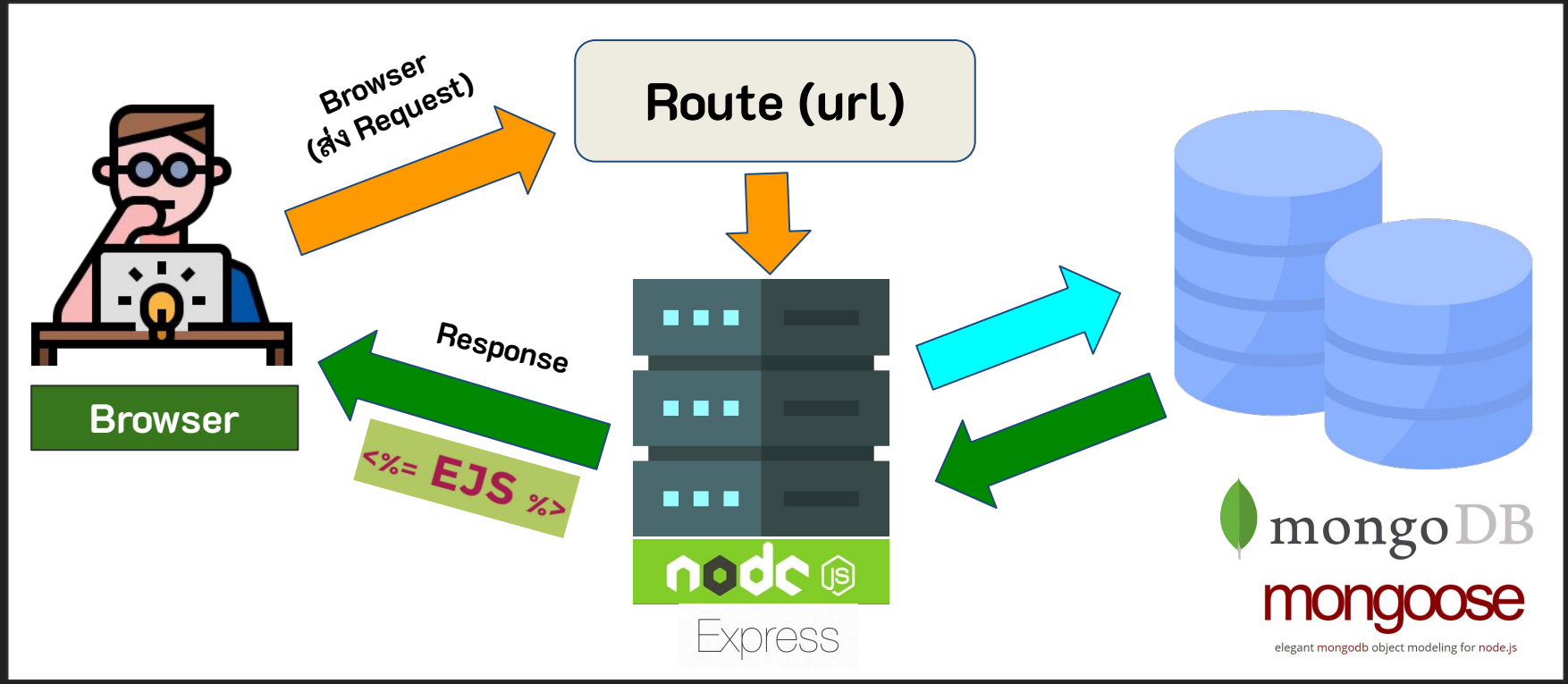
รู้จักกับ Cookie

รู้จักกับ Cookie

การทำงานระหว่าง Browser และ Server นั้นจะเกิดขึ้นเมื่อ Browser ส่ง Request ไปที่ Server และ Server Response กลับมา การทำงานก็จะสิ้นสุดลงและเมื่อปิด Browser ข้อมูลต่างๆที่แสดงผลใน Web Page ก็จะถูกทำลายไป แต่ถ้าเราต้องการอยากจะทำให้ Web Page คงสถานะหรือเชื่อมโยงการทำงานระหว่าง Web Page เข้าด้วยกันได้

จะอาศัยส่วนที่เรียกว่า Cookie และ Session

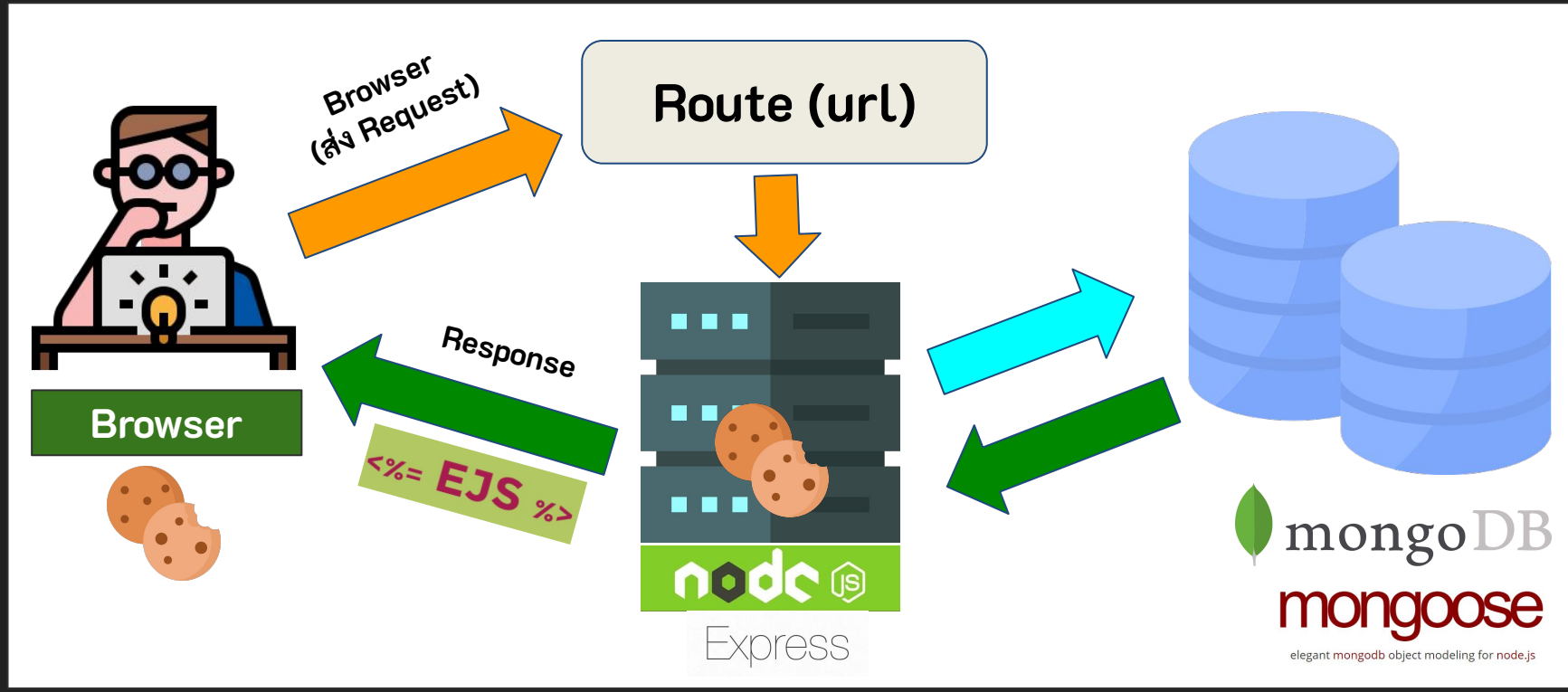
รู้จักกับ Cookie



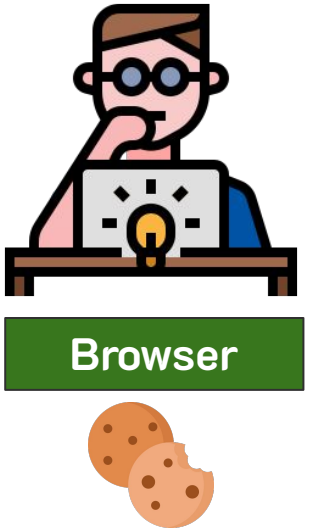
Cookie (คุกกี้)

เป็นรูปแบบการเก็บข้อมูลบางอย่างไว้ในเครื่องผู้ใช้ เพื่อที่จะนำข้อมูลดังกล่าวมาใช้ในภายหลังถึงจะปิด Browser ไปแต่ข้อมูลใน Cookie ก็จะยังคงอยู่

Cookie (คุกกี้)


Browser
(ส่ง Request)
Route (url)
Response
Browser
<%= EJS %>
Express
mongoDB
mongoose
elegant mongodb object modeling for node.js

กำหนดสิทธิ์ในการเข้าถึงข้อมูล



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 (เซสชั่น)

เป็นรูปแบบการเก็บข้อมูลบางอย่างไว้ใน Server เพื่อที่จะนำข้อมูลดังกล่าวมาใช้ในภายหลัง

คุณสมบัติของ Session (เซสชั่น)

- 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 (เซสชั่น)

- ข้อมูลใน Session จะนำมาจำแนกผู้ใช้แต่ละคนออกจากกัน มันจะใช้ได้เฉพาะกับผู้ใช้คนๆนั้นไม่สามารถใช้งานร่วมกับคนอื่นได้

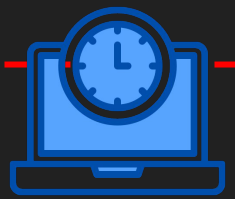
- ค่า Session ID ไม่สามารถใช้ร่วมกันระหว่าง Browser ได้ เช่น Session ที่ทำงานใน Google Chrome ไม่สามารถนำไปใช้กับ Firebox หรือ Safari ได้

- ข้อมูล Session ที่สร้างขึ้น สามารถนำไปทำงานในแต่ละ Page ได้เหมือนกับ Cookie เลย

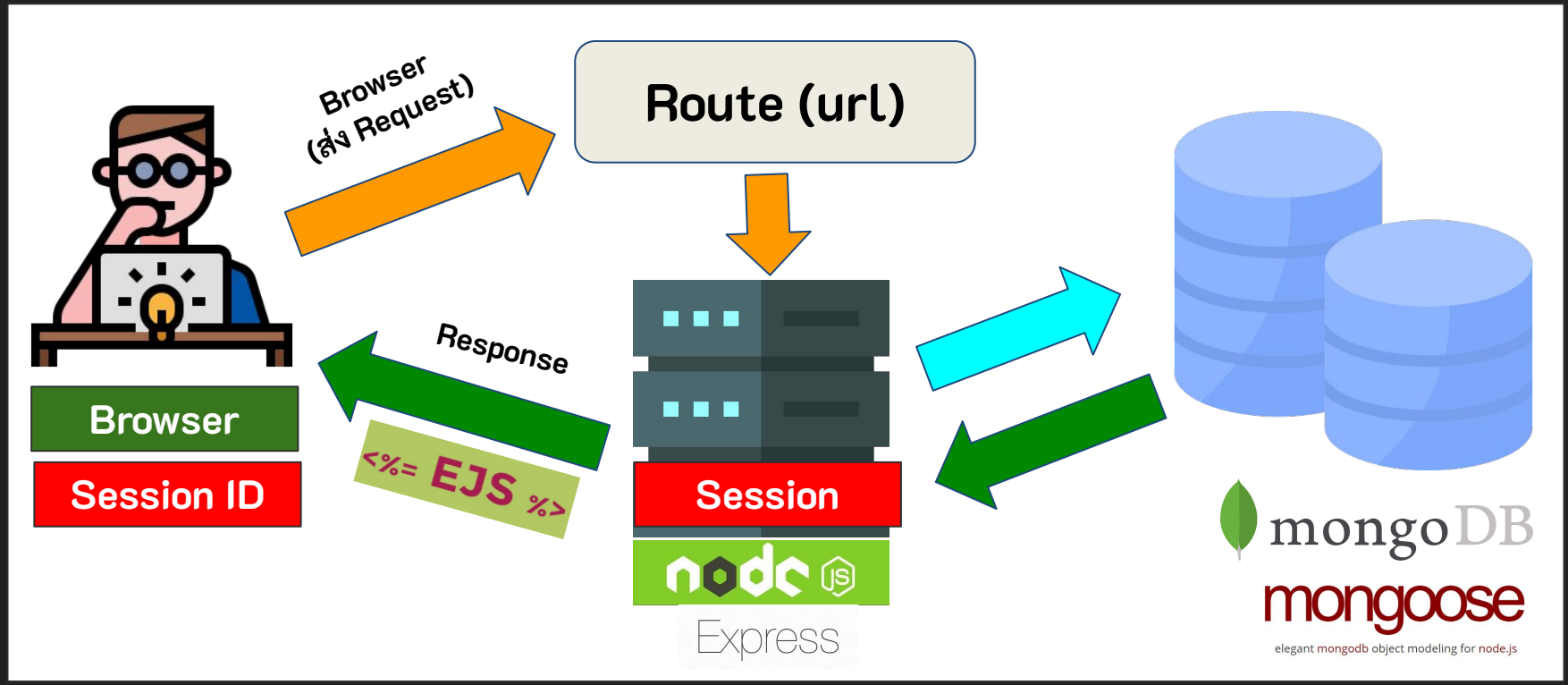
คุณสมบัติของ Session (เซสชั่น)

- ข้อมูล Session จะใช้งานได้แค่ชั่วคราวเท่านั้น คือเชื่อมต่อเมื่อเวลาเปิด Browser แต่เมื่อปิด Browser ไปข้อมูลก็จะถูกยกเลิกการเชื่อมต่อ

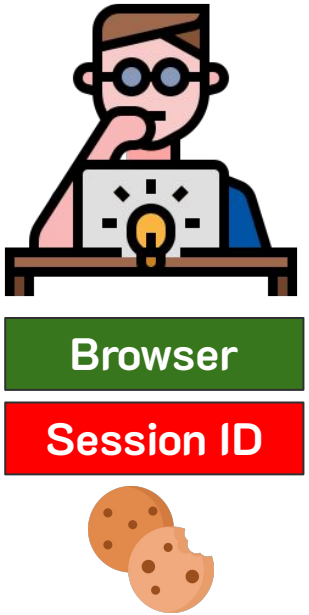
- ถ้าต้องการให้สภาพการเชื่อมต่อยังคงอยู่หรือการทำงานเหมือน Cookie จำเป็นต้องอาศัยการจัดเก็บ Session ID ไว้ที่เครื่องผู้ใช้และเก็บข้อมูล Session ไว้ที่ Server แทน



Session (เซสชั่น)



Session (เซสชั่น)



session id

จัดการสินค้า

seesion 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

})
```