

เริ่มต้นเขียนเว็บไซต์ ด้วย nodejs และ vuejs จะบันจับมือทำ

ผลงานการเขียนของ

นายชัยพล จันทร์

Mr.Chayapon Chandra

korn.chayapon@gmail.com

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

บทนำ

แรกเริ่มเดิมที่ ผมวางแผนที่จะเขียนหนังสือเกี่ยวกับพื้นฐานเรื่อง nodejs และ vuejs ก่อน จากนั้นจึงจะทำการเขียนหนังสือเกี่ยวกับ การทำเว็บไซต์ด้วย nodejs และ vuejs แต่รู้สึกว่ามันคงน่าเบื่อมาก ที่ต้องอ่านหนังสือถึงสามเล่ม เพื่อที่จะทำ เว็บไซต์ให้ได้หนึ่งเว็บไซต์ ถึงความจริงแล้วการทำเว็บไซต์ ขึ้นมาหนึ่งเว็บไซต์นั้น จะประกอบด้วยหลายสิ่งอย่างก็เถอะ ..

ในโลกอินเตอร์เนท มีการเขียนบทความการสอนเขียนโปรแกรม nodejs และ javascript พื้นฐานอยู่มากแล้ว และการทำงานจริงๆ เรา ก็ไม่จำเป็นต้องรู้ทุกอย่าง เกี่ยวกับสิ่งที่เราจะเรียนรู้ จนครบเสียก่อน แล้วค่อยลงมือทำ เพราะถ้าเป็นเช่นนั้น ในแต่ละปี เราคงทำงานได้ไม่กี่ชั้นแน่นอน ดังนั้นผมจึงคิดว่า จะเริ่มสอน ตาม workflow การทำงานจริง ของผมเลยน่าจะดีกว่า..

หนังสือเล่มนี้ ไม่ใช่หนังสือ สอนการทำเว็บไซต์ที่ดีและถูกต้องที่สุด และแน่นอนมัน ไม่ใช่หนังสือ ที่สอน nodejs และ vuejs ที่ระเอียดและถูกต้องที่สุด แต่เป็นหนังสือที่ จะสอนคุณทำเว็บไซต์ด้วย nodejs และ vuejs แบบเป็นขั้น เป็นตอน หรือ ภาษาปะ กิด เรียกว่า step by step จนคุณสามารถสร้างเว็บไซต์ ที่เป็นเว็บบล็อกส่วนตัวของ คุณได้

ผมแนะนำให้อ่านแบบสบายใจ yein ๆ และทำตามไปเรื่อย ๆ อันไหนที่ติดให้ถ(TM)า หรือค้างໃສ้กระดาษโน๊ตไว้ก่อน ข้ามไปทำตามเรื่อย ๆ จนสำเร็จหนึ่งเว็บไซต์ ไม่เช่นนั้น คุณจะเสียเวลาไปอย่างมาก และจะมาร้องอ้อในตอนท้าย

เมื่อทำได้หนึ่งเว็บไซต์แล้ว ให้นำความรู้ไปทำเว็บไซต์อื่น ๆ เพิ่มขึ้นให้สำเร็จอีก คุณจะสามารถสร้างงานของคุณได้อย่างชำนาญ ไม่ต้องกังวลหากคุณติดปัญหา หรือ error ตรงไหน ที่เกี่ยวข้องกับหัวข้อ ที่มีอยู่ในหนังสือเล่มนี้ (นอกเรื่องนิด หน่อยก็ได้นะครับ ^^) คุณสามารถสอบถามกับผมได้โดยตรงครับผ่าน “จับมือทำ nodejs และ vuejs” ใน Facebook ได้เลยครับ หรือ Inbox สอบถามได้ที่ <https://web.facebook.com/GoodDevThailand>

พื้นฐานด้านการเขียนโปรแกรม

พื้นฐานที่คุณควรมี ในการอ่าน และทำตามหนังสือเล่มนี้ นั้นคือ JavaScript พื้นฐานเพียงเล็กน้อยนั้นเพียงพอ ที่จะอ่านให้เข้าใจ และทำตามผมไป จนสามารถ ทำเว็บไซต์สำเร็จได้อย่างแน่นอน หากคุณ ไม่มีพื้นฐาน JavaScript เลย ก็สามารถ เริ่มศึกษา JavaScript จาก YouTube หรืออ่านจากคู่มือ <https://developer.mozilla.org/bm/docs/Web/JavaScript> ไปพร้อมกันได้เลย ครับ แต่ถึงคุณจะมีพื้นฐานด้านการเขียนโปรแกรม หรือ JavaScript ไม่มากนัก ผม จะพยายามแทรก และอธิบายพื้นฐานประกอบไปตลอดจนจบเล่ม

บทที่ 1 ทำความเข้าระบบ

เรากำลังจะเรียนทำอะไรกัน?

เรากำลังจะทำเว็บไซต์แบบ “เว็บบล็อก (Web Blog)” กันครับ โดยเราจะทำการเขียนส่วนของ Back End ด้วย nodejs และส่วนของ Front End เราจะใช้ vuejs กัน โดยในเว็บบล็อกของเราจะมีระบบ Member Login สำหรับ Admin เพื่อให้ผู้ใช้ที่เป็น Admin สามารถเข้าไปเขียน แก้ไข ลบ และควบคุมการเผยแพร่บล็อกของเรา รวมถึงมีระบบ User Login สำหรับให้ผู้อ่านเข้ามา Comment บล็อกของเราได้ด้วย

ในส่วนท้ายเราจะมาประยุกต์ใช้งาน โดยการทำส่วนของตักร้าสินค้าเพิ่มเข้าไปในเว็บบล็อกของเรา เพื่อขาย สินค้า online หรือเรียกว่า e-commerce แบบบ้าน ๆ กันครับ

ความสัมพันธ์ระหว่าง Back End, Front End, Back Office และ Front Office

ตอนนี้เรามาลองเขียนโปรแกรมใหม่ ๆ เรียกผิด เรียกถูก ฟังอย่าง เข้าใจยากอย่างประจำเลยครับ เราจะจำกันง่าย ๆ อย่างนี้ครับ ถ้าเราเปรียบเว็บบล็อกที่เราทำ เป็นสำนักพิมพ์ ส่วนของ Back End ก็คือ โรงพิมพ์ ของเรานั่นเอง โดย Front End ของเรา จะประกอบ 2 ส่วน คือ สำนักงานของโรงพิมพ์ ซึ่งเราจะเรียก Back Office และร้านขายหนังสือ ซึ่งเราจะเรียกว่า Front Office ของเรานั่นเอง

Back	End	=	Front	Office	=	โรงพิมพ์	
Front	End	=	Front	Office	=	Back	Office
Front						ร้านขายหนังสือ	
Back Office = สำนักงานของโรงพิมพ์							

Back Office หรือ สำนักงานของโรงพิมพ์ ก็คือส่วนของแอดมินของเราที่จะไปจัดการ บทความ หรือ บล็อกของเรา เช่น เขียน แก้ไข เพยแพร หรือ หยุดการเผยแพร่

Front Office หรือ ร้านขายหนังสือของเรา จะเป็นส่วนที่ผู้ใช้งานเข้ามาอ่านบทความ หรือบล็อกของเรา เพียงแต่ ร้านของเราบริการอ่านฟรี และคนที่เข้ามาอ่านบล็อกของเรา ถ้ากรอกข้อมูลให้เรา ก็จะเข้ามาเป็นสมาชิก และสามารถแนะนำติชม บทความ หรือ บล็อกของเราได้ด้วย

*อย่าลืมนะครับว่า Back Office (สำนักงานของโรงพิมพ์) และ Front Office (ร้านขายหนังสือ) ของเราจะอยู่ในส่วนของ Front End นะครับ

HTTP Method, URL, Request และ Response คืออะไร เกี่ยวข้องกันยังไง

เมื่อเราเปรียบ Front End ในส่วนของ Back Office เป็น สำนักงานของโรงพิมพ์ เวลาที่สำนักงานของโรงพิมพ์ จะสั่งงานโรงพิมพ์ หรือ Back End จะทำการส่งความต้องการ หรือ Request ไปหาที่โรงพิมพ์ (Back End) โดยระบุแพนก หรือ เราเรียกว่า URL และ คำสั่งหรือวิธีไปด้วย คำสั่งหรือวิธีนี้ล่ะ เราเรียกว่า HTTP Method โดย ความต้องการ ซึ่งเราเรียกว่า Request หรือตัวย่อ req จากสำนักงานที่ส่งไปที่โรงพิมพ์ ในการเขียนโปรแกรมของเรา หรือ HTTP Method ประกอบด้วย 4 อย่าง ตอนนี้รู้แล้วเนี่ยพอจะครับ

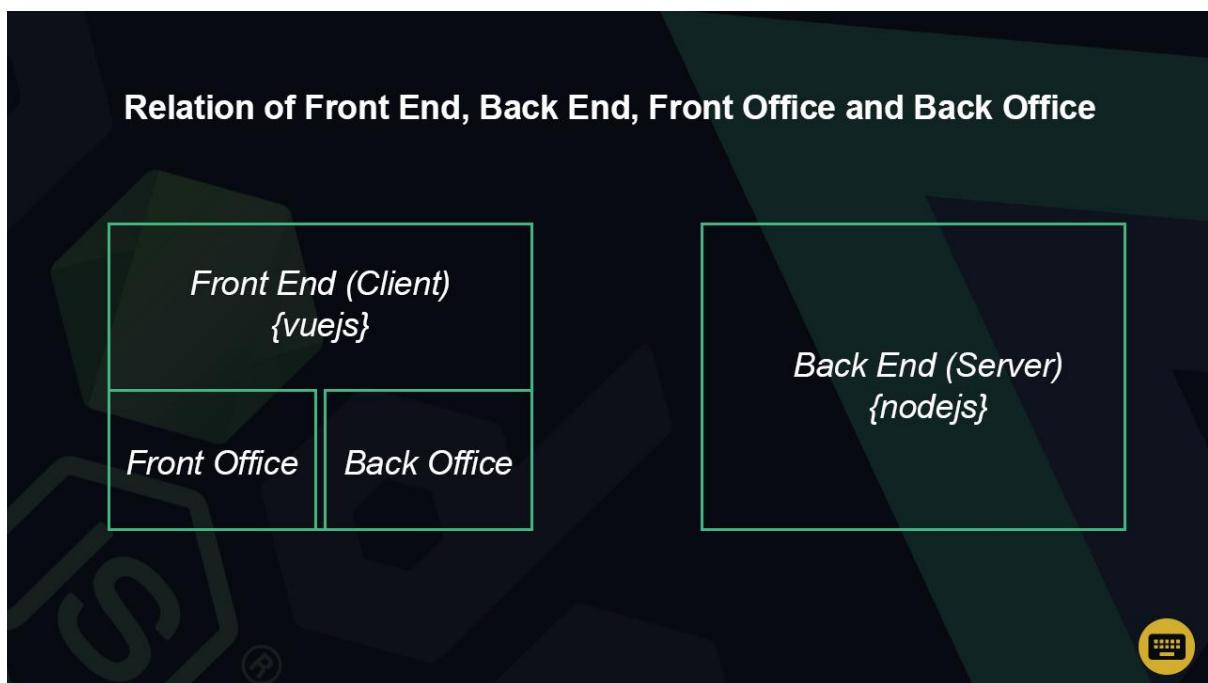
1. get คือ อ่าน หรือ เรียกหนังสือมาจากโรงพิมพ์
2. post คือ เขียน หรือ เอาหนังสือจากผู้แต่งไปโรงพิมพ์ ไปทำการพิมพ์
3. put คือ แก้ไข หรือ เขียนใหม่เดิม เอาไปแก้ใหม่นะ โรงพิมพ์
4. delete คือ ลบ พิมพ์ผิด เขียนไม่ดี เอาไปทำลาย

เมื่อโรงพิมพ์ (Back End) ได้รับความต้องการหรือการ Request (req) จากสำนักงานของโรงพิมพ์ (Front End / Back Office) และ จะทำการส่งไปที่แพนก (URL) ต่าง ๆ ตามที่ Back Office Request (req) มา หากมีแพนกหรือ URL นั้นมีอยู่ และสามารถทำได้ตามคำสั่ง ก็จะดำเนินงานให้ และส่งผลกลับมา หรือเรา

เรียกว่า Response (res) ที่สำนักงาน (Front End / Back Office) ถ้าไม่มีแผนกนั้น อปุ่ หรือ ไม่สามารถทำงานได้ตามคำสั่งที่ Request (req) มา ก็จะตอบกลับมาเป็น Error ตามกรณีนั้น ๆ รวมถึงตอบกลับ Error หากrongพิมพ์ทำงานผิดพลาดด้วย เช่นกัน หาก Error ต่าง ๆ ที่ส่งกลับมาได้ แก่ 404 Not Found เป็นต้น

การทำงานของ ร้านขายหนังสือ หรือ Front Office ของเรา จะทำงานร่วมกับ rongพิมพ์ (Back End) ของเราโดยตรง โดย เมื่อลูกค้าเลือกหนังสือแล้ว Front Office ของเรา Request (req) ไปที่rongพิมพ์ (Back End) แบบ get เพื่อเรียกหนังสือมา ให้ลูกค้าอ่าน แต่การทำงานบนสายเคเบิลนั้นเร็วมาก เรียกไปที่rongพิมพ์ (Back End) ปุ๊บ ถ้ามีหนังสืออยู่ rongพิมพ์จะส่งหนังสือกลับมา (Response) ลูกค้า หรือ ผู้อ่านเบล็อกของเรา ก็จะได้อ่านทันที ถ้าไม่มีหนังสือ หรือทำงานผิดพลาด จะตอบกลับมา (Response) เป็น Error ตามกรณีนั้น ๆ ดังที่ได้กล่าวไปแล้ว

โดยระบบจัดการคำสั่ง (Request : req) และการตอบกลับ (Response : res) ที่ วิ่งไปตามแผนกต่าง ๆ จาก Front End (Back Office / สำนักงานrongพิมพ์ + Front Office / ร้านขายหนังสือ) ไปที่ Back End (rongพิมพ์) ก็คือ RESTful API (อ่านว่า “เรสฟลู” นะครับ) ที่เราจะเขียนขึ้นเอง โดย เราจะใช้ nodejs ทำส่วนของ Back End และใช้ vuejs ทำส่วนของ Front End กันครับ



ส่วนประกอบฝั่ง Server และ ฝั่ง Client

ในบทนี้ เราไม่ต้องไปกังวลกับ Keywords กันมากมายนะครับ อันไหนยังไม่รู้จัก ไม่คุ้น ก็จำๆ มันไปก่อนว่า เคยกล่าวถึง คือ ๆ เรียนรู้ไปเรื่อย ๆ จะรู้จักมากขึ้น จะเข้าใจได้เองครับ

จากที่กล่าวมาพอเดา กันออกมั่ยครับ ว่า ส่วนไหนของเรานาง คือ ส่วนของฝั่ง Server และ ฝั่ง Client ระบบของเรา ฝั่ง Server คือ โง่พิมพ์ (Back End) และ ฝั่ง Client คือ (Front End) ของเราทั้งหมดนั่นเอง

ส่วนประกอบในฝั่ง **Server (Server Component /Nodejs - RESTFul API)** มีดังนี้ครับ

- Routes
- Controller
- Model (Database)
- Permission

Server เราทำด้วย nodejs แบบ RESTFul API

Routes เอาไว้ทำ URL Route หรือ ตำแหน่งสำหรับส่งข้อมูลไปแพนกต่าง ๆ ใน โง่พิมพ์ของเรานั่นเอง

Controller คือแพนกต่าง ๆ ของโง่พิมพ์ของเรา ตามที่กำหนดไว้ใน URL โดยทำงานร่วมกับ Model

Model เอาไว้จัดการฐานข้อมูลที่เราใช้งาน

Permission เอาไว้จัดการการเข้าถึงข้อมูลบน Server ของเรา ว่า ใครเป็นผู้จัดการล้านักพิมพ์ ใครเป็นผู้อ่าน

ส่วนประกอบในฝั่ง Server (Client Component / Vuejs) มีดังนี้ครับ

- Router
- Vue Component
- Assets
- Services
- Permission

Router เอาไว้จัดการ url ในส่วนของ Front end ว่าจะไปเลือกใช้ Vue Component ได ๆ

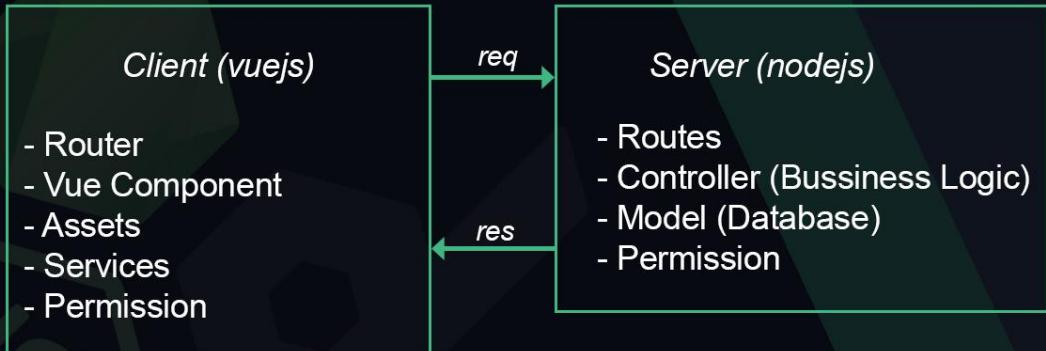
Vue Component ประกอบด้วย HTML, CSS/SCSS และ javascript เป็นตัวจัดการแสดงผล ตามที่กำหนด โดยทำงานร่วมกับ Services เพื่อวันส่งข้อมูล กับ Server และดึง Assets มาใช้งานร่วมด้วย

Assets เป็นพวก รูปภาพต่าง ๆ หรือ อะไรที่เป็น static file ไว้โหลดใช้งานที่หน้าบ้าน

Services เป็นการใช้บริการในส่วน Front End สำหรับให้ Vue Component เรียกใช้ เพื่อ ติดต่อ รับส่งข้อมูลกับ Server

Permission เป็นตัวจัดการสิทธิ์ ในการเข้าถึง Vue Component ต่างๆ ของเรา ในส่วน Front End

Client and Server Components



บทที่ 2 ความรู้พื้นฐาน

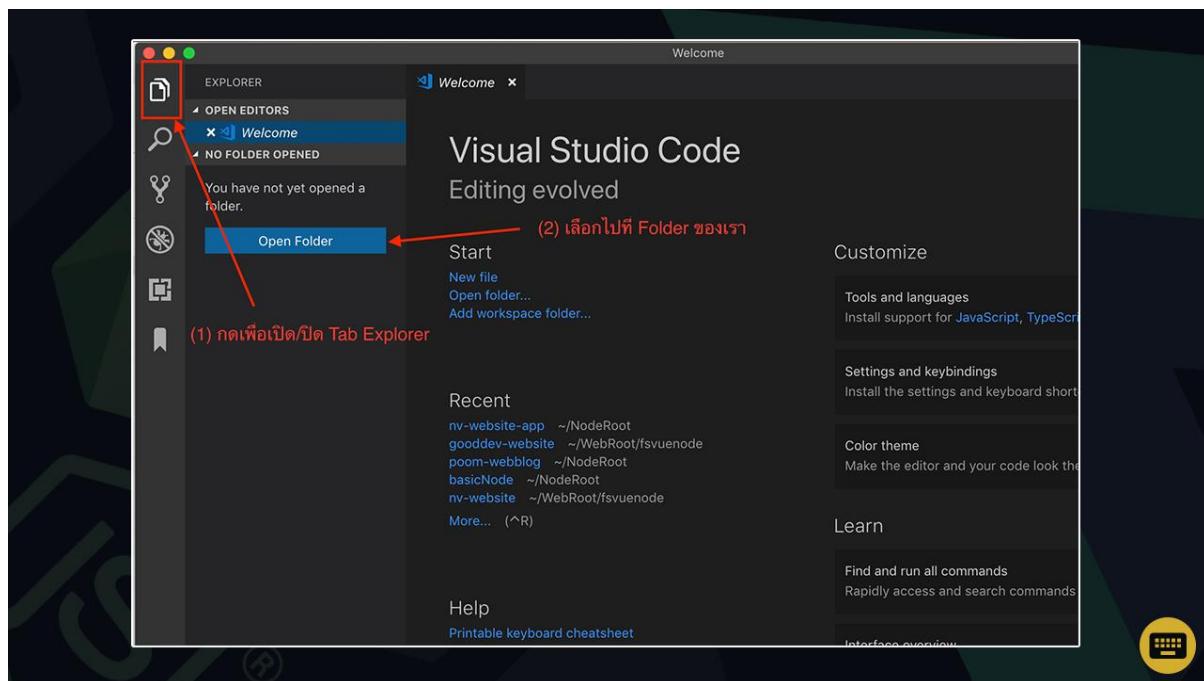
เครื่องมือในการเขียนโปรแกรม

เครื่องมือในการเขียนโปรแกรมทั้งหมดของหนังสือเล่มนี้ คือ Visual Studio Code หรือ เรียกสั้น ๆ ว่า VS Code เข้าไป download ได้ฟรี มีรองรับทั้งระบบปฏิบัติการ macOS และ Windows ทำการ download ได้ที่นี่ครับ <https://code.visualstudio.com/>

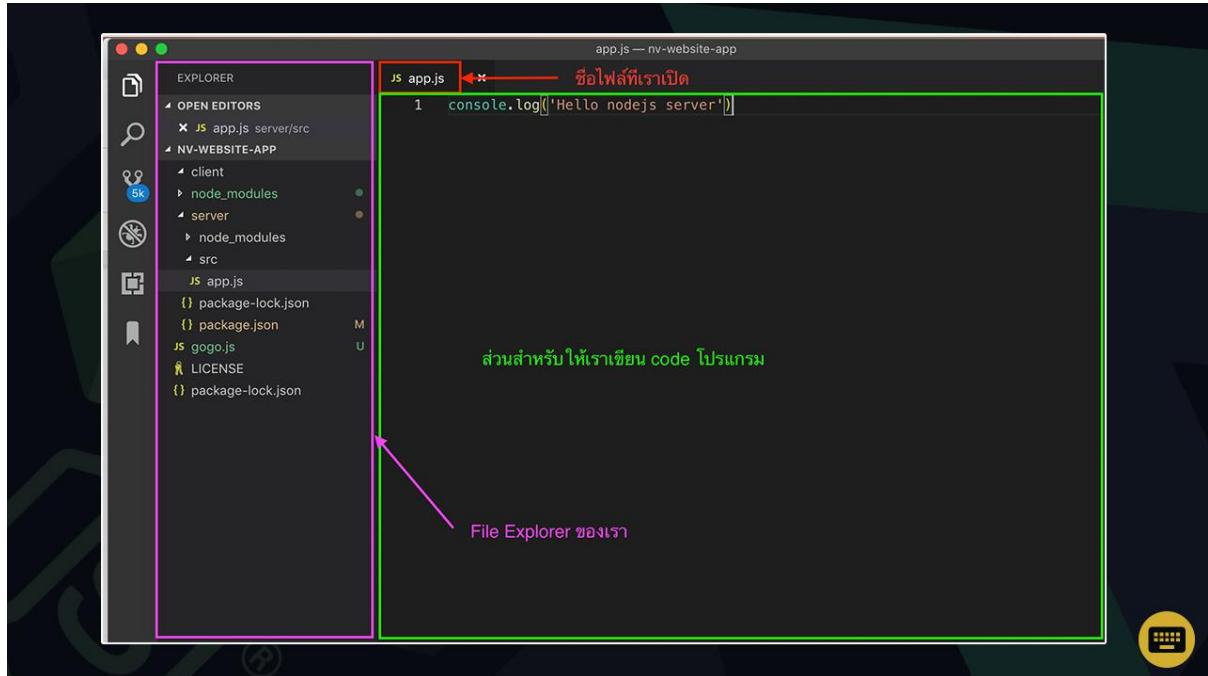
วิธีติดตั้งผมไม่ได้สอนนะครับ น่าจะทำเป็นกันเกือบทุกคนอยู่แล้ว แต่ถ้าใครไม่เป็นจริง ๆ ตั้งกระทู้ถามได้เลยครับ

การใช้งาน VS Code เป็นต้น (สำหรับมือใหม่มาก)

เมื่อเปิดโปรแกรม VS Code ขึ้นมา ให้เปิด File Explorer Tab ดังภาพ (1) โดยกดที่ปุ่มดังภาพ จะเป็นการเปิด/ปิด File Explorer ของเรา



จากนั้นทำการกดปุ่ม Open Folder เพื่อเปิดโปรเจคของเรามาทำการเขียนโปรแกรม ดังภาพ (2)



ในตอนนี้เรายังไม่ได้เขียนโปรแกรมอะไรกัน เรา ก็ปิดไปก่อนได้ครับ จะเขียนผมจะบอกอีกทีครับ ไม่ต้องกังวล

nodejs เปื้องต้น

ในส่วนนี้เราจะพูดถึงการเขียนโปรแกรมด้วย nodejs เปื้องต้น แต่ไม่ใช้การเขียนโปรแกรม javascript เปื้องต้น หากคุณไม่มีพื้นฐาน สามารถศึกษาได้จาก Youtube หรือ อ่านคู่มือการใช้งาน JavaScript ได้ที่นี่ครับ <https://developer.mozilla.org/bm/docs/Web/JavaScript>

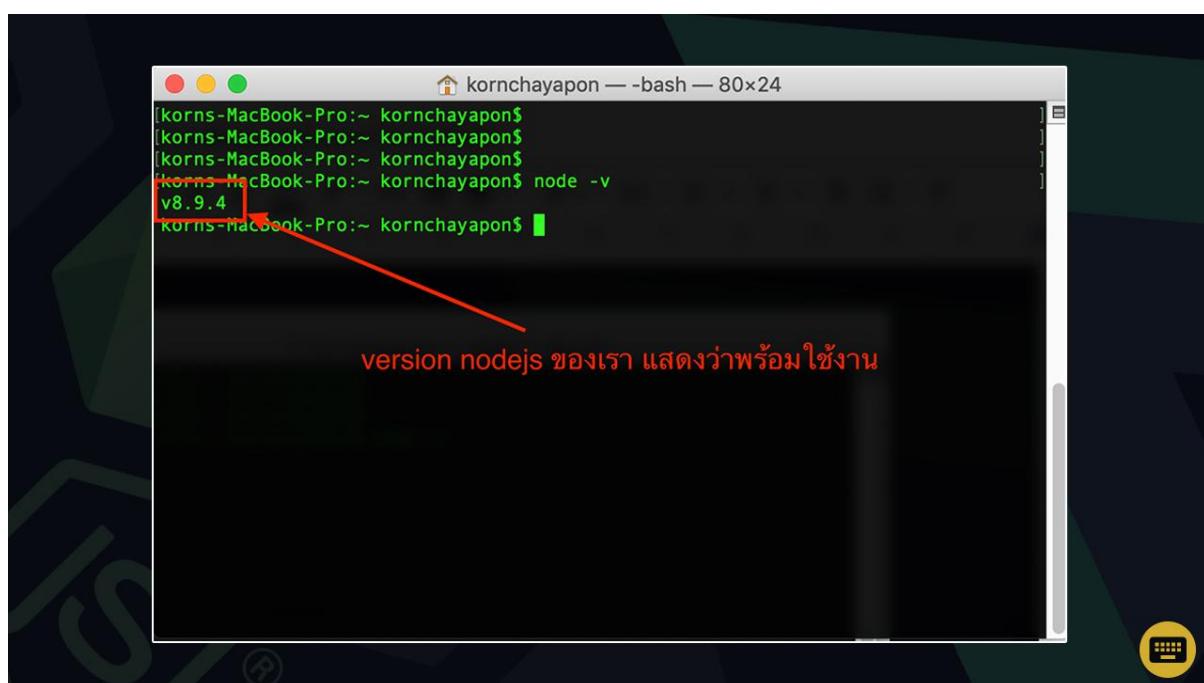
คุณไม่จำเป็นต้องเป็นเทพ javascript มาเลยเพื่อจะอ่านหนังสือเล่มนี้ให้เข้าใจ เพียงพื้นฐานเล็กน้อย ก็ยังได้ ตัวแปร, condition if else, for loop และ function ก็สามารถทำความเข้าใจ และเขียนโปรแกรมตามหนังสือเล่มนี้ได้ คุณเพียงทำความเข้าใจ และทำตามไปได้เรื่อย ๆ จะสามารถสร้างเว็บบล็อกของเราได้

nodejs คือ อะไร

nodejs คือโปรแกรม โปรแกรมนึง ที่ทำให้ javascript สามารถอุปกรณ์ และทำงานนอก Web Browser ได้ หรือพูดง่าย ๆ คือ ทำให้เราสามารถเขียนโปรแกรม javascript ของเรา ได้เหมือน ภาษาซี หรือ python แต่ใช้ Syntax ของ javascript แทนจะแสดงผลบน Web Browser เท่านั้น มันก็ไปแสดงผลผ่าน Console Window จะดี ๆ หรือมี ปุ่ม นั่นนี่ให้ Click ได้แทน และความสามารถที่โดดเด่น ของมัน ก็คือการรับและส่งข้อมูลบน HTTP ได้ด้วยนั่นเอง ซึ่งเมื่อนำเข้ามายัง nodejs Offline และ Online เข้าด้วยกันเลยครับ

โครงสร้างโฟล์เดอร์ และเตรียม nodejs ของเราให้พร้อม

สร้างโฟล์เดอร์ nodeRoot ขึ้นมาจะเอาไว้ให้เก็บตามส่วนต่างๆ เช่น d:/nodeBasic โดยโฟล์เดอร์นี้เราจะเอาไว้เก็บโปรแกรม nodejs ที่เราเขียนขึ้นนั่นเอง จะเขียน โปรแกรม และรัน nodejs เรายังต้องมี โปรแกรม nodejs ในเครื่องของเรามีการติดตั้งอยู่แล้วทำการตรวจสอบก่อนว่า เครื่องเรามี nodejs อยู่หรือไม่ โดยทำการพิมพ์คำสั่งดังนี้

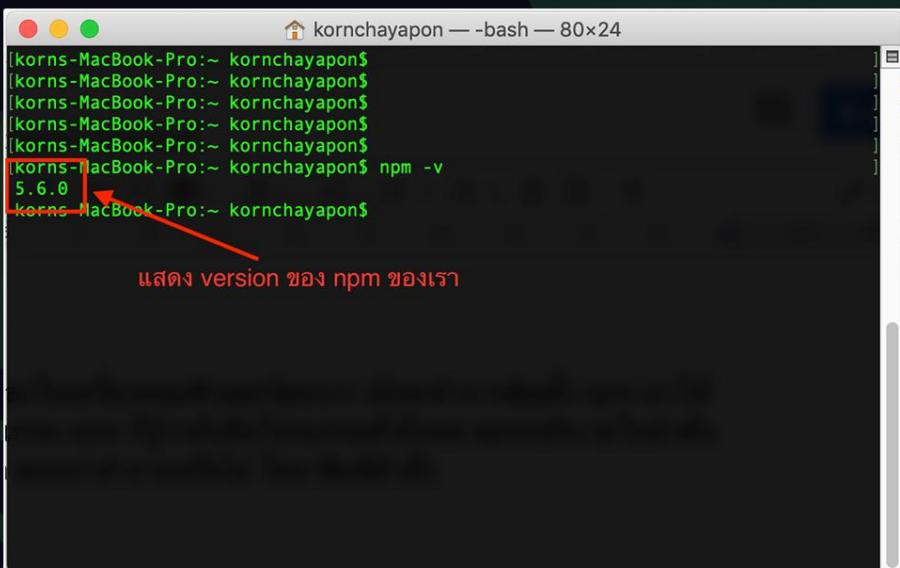


ถ้าไม่แสดง version ออกมาดังภาพ แสดงว่าไม่มีโปรแกรมอยู่ให้ทำการติดตั้ง nodejs ก่อน

โดย เข้าไปที่ <https://nodejs.org/en/> จากนั้นเลือก download และติดตั้งตามระบบปฏิบัติการที่ใช้อยู่

ทำการพิมพ์คำสั่งแสดง version อีกครั้ง ถ้าผลการรันโปรแกรมแสดง version ออกมาดังภาพแสดงว่า เครื่องเรารอุ่นสำหรับการรัน nodejs แล้ว ถ้า error อื่น ๆ สอนความหลังไม่คุ้มค่าผมได้ครับ

เมื่อเราทำการติดตั้งโปรแกรม nodejs ลงในเครื่องคอมพิวเตอร์ของเรา มันจะทำการติดตั้ง npm มาให้เราด้วย ถึงตอนนี้ถ้าใครยังไม่รู้จักโปรแกรม npm ก็รู้ว่ามันคือโปรแกรมตัวเน็งพอ ผมจะอธิบายในลำดับต่อไป แต่ตอนนี้มาทดสอบก่อนว่า npm ของเราทำงานหรือไม่ โดย พิมพ์คำสั่ง

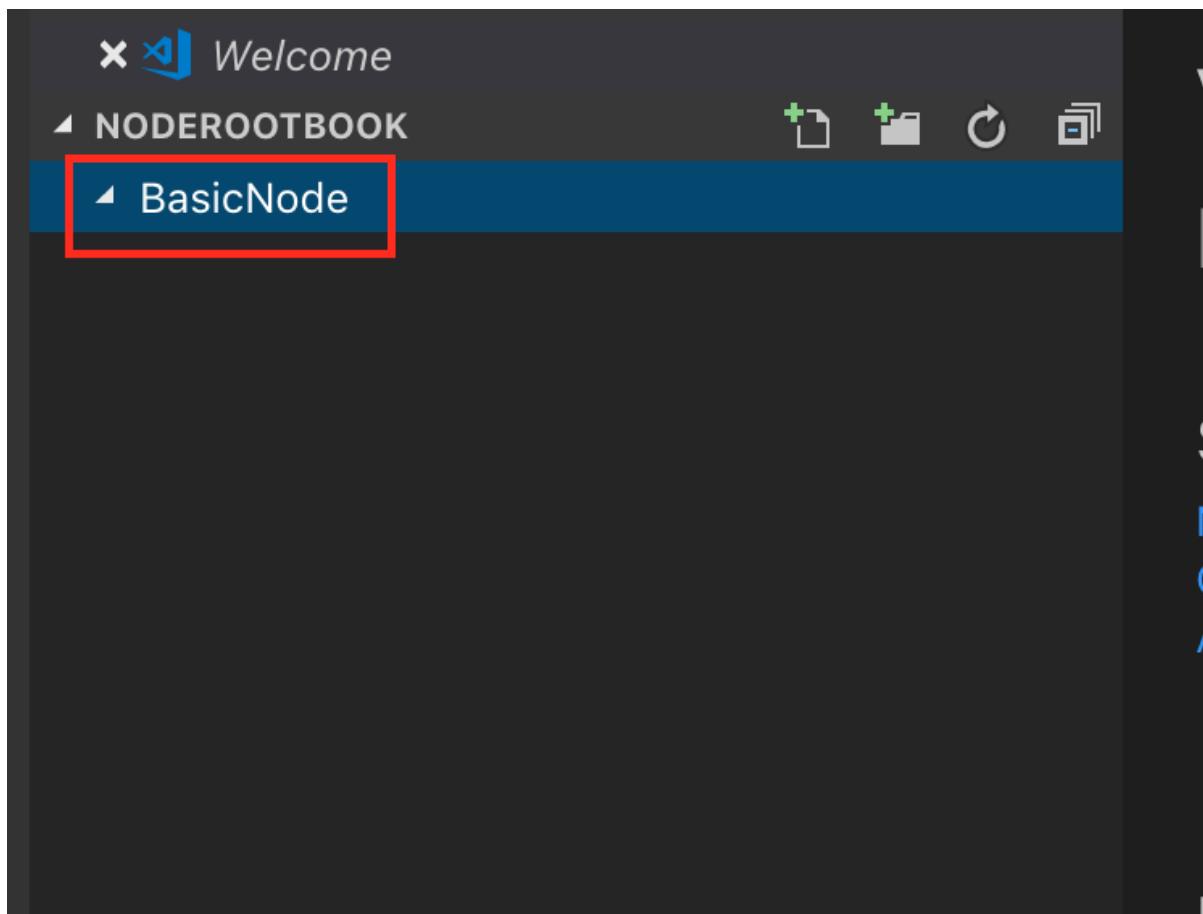


```
[korns-MacBook-Pro:~ kornchayapon$ npm -v 5.6.0 korns-MacBook-Pro:~ kornchayapon$
```

แสดง version ของ npm ของเรา

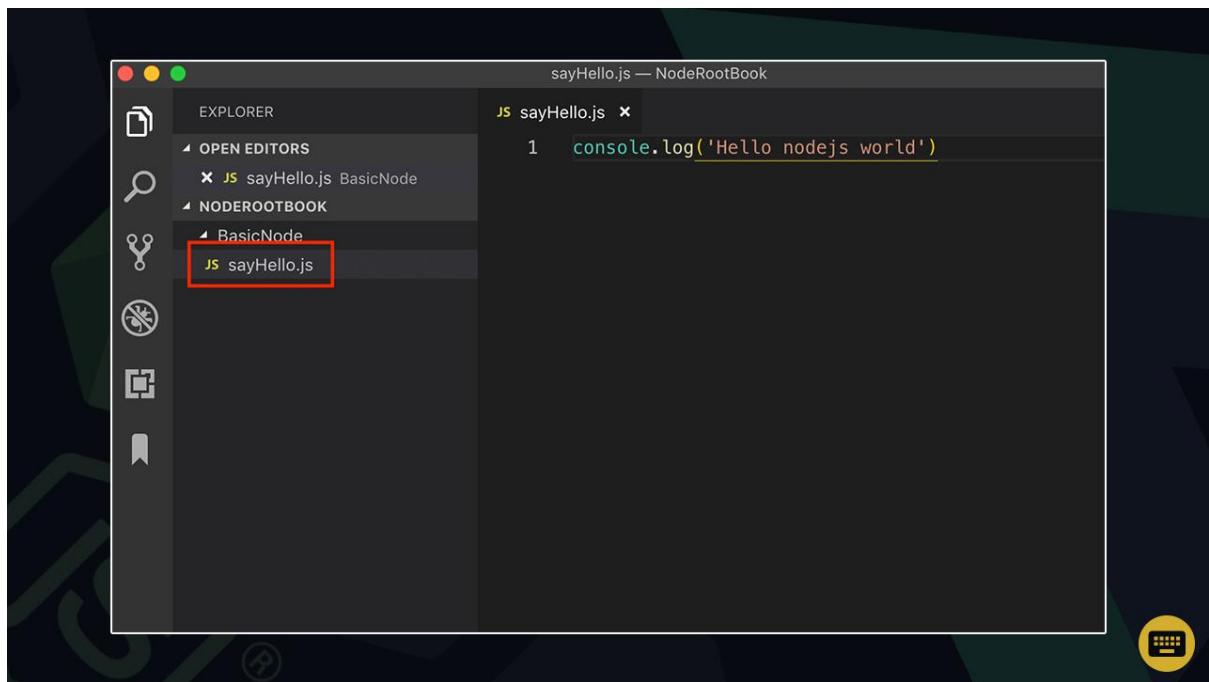
เริ่มต้นเขียน nodejs

เปิดโปรแกรม VS Code ขึ้นมา เขียน code ดังนี้ ผมจะทำการสอนสองส่วนนะครับ สำหรับพื้นฐาน ส่วนแรก คือ พื้นฐาน nodejs และ json จากนั้นเราจะไปเริ่มทำเว็บ บล็อกของเรากัน ผมเลยสร้าง BasicNode ไว้เก็บ การเขียนโปรแกรมพื้นฐานของ เราก็ฟอล์เดอร์นึงแยกกอกมา

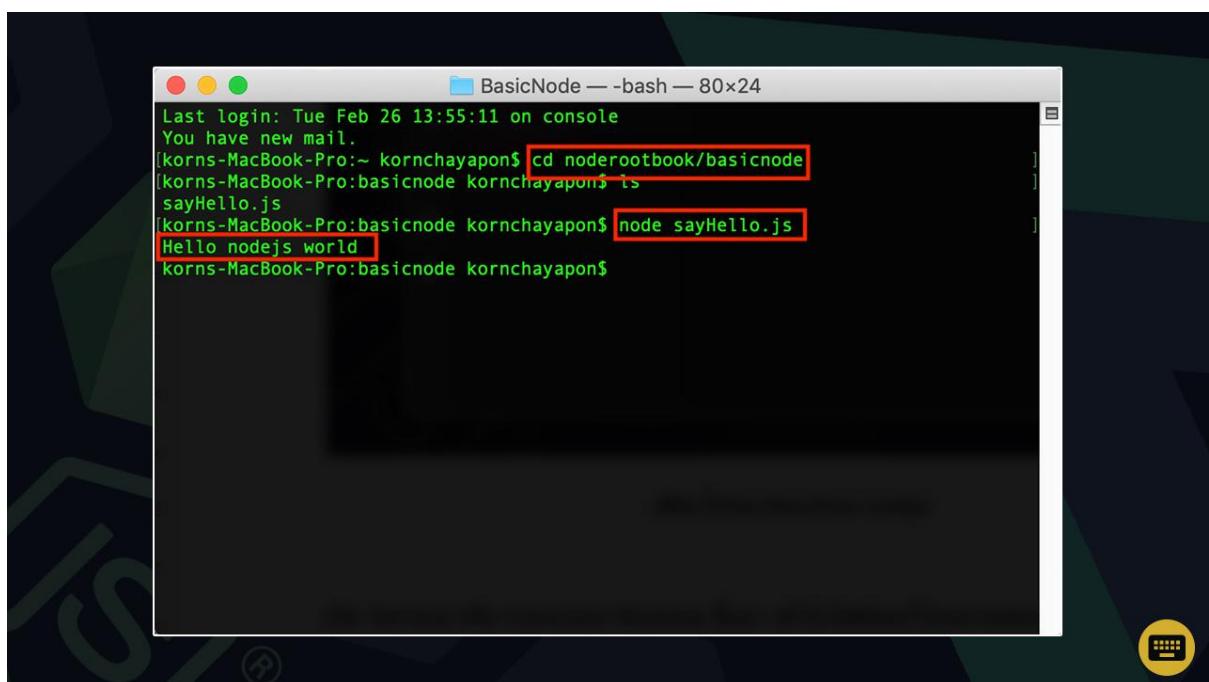


สังเกตว่าเรายังไม่ได้สร้างโฟล์เดอร์ใด ๆ เกี่ยวกับการเขียนเว็บบล็อกของเรานะ ครับ เพราะเราจะไปสร้างบน github.com และทำการ clone มา ซึ่งผมสอนในบท ต่อ ๆ ไปครับ ในบทนี้เราจะมาเล่นกับ nodejs เป็นต้นกันก่อนครับ

ทำการสร้างไฟล์ใหม่ขึ้นมา (New File) จากนั้น เรา ก็ทำตามธรรมเนียมการเขียน โปรแกรมครับ โดยการเขียนโปรแกรม Hello world ด้วย nodejs นั่นเองครับ ทำการพิมพ์ code โปรแกรมตามผมเลย ทำการบันทึกเป็น sayHello.js ครับ



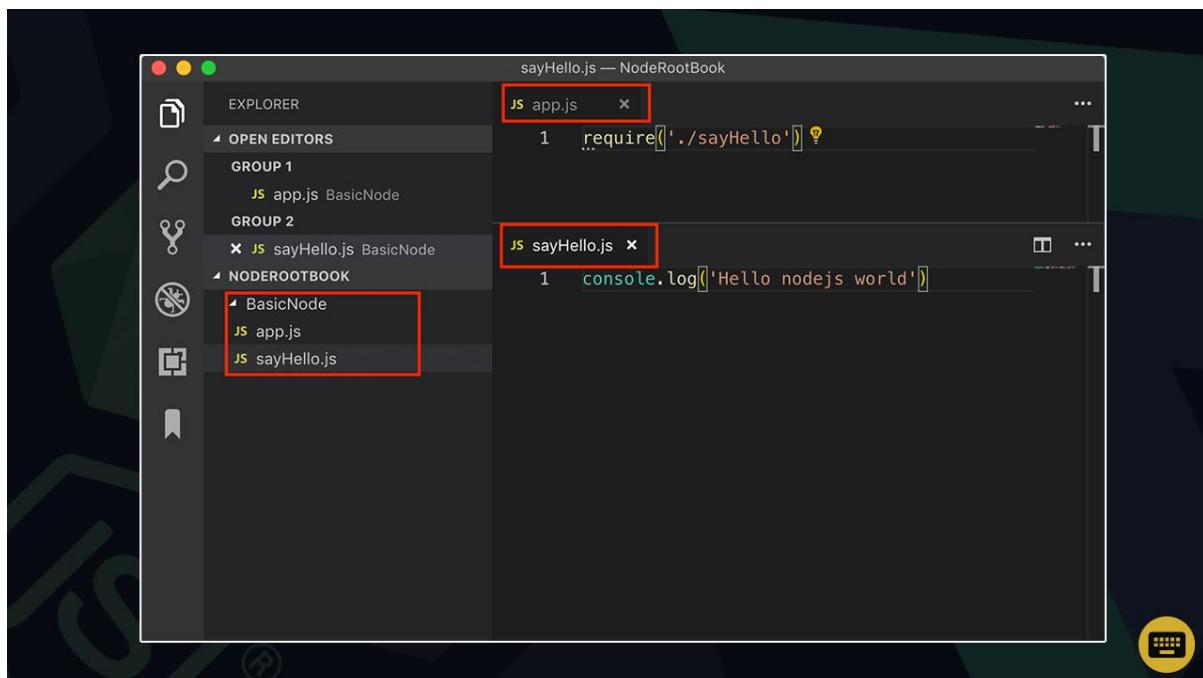
เปิด Terminal หรือ Command Windows ขึ้นมา เข้าไปโฟล์เดอร์โปรแกรมของเรา (NodeRootBook/BasicNode) ด้วยคำสั่ง cd NodeRootBook/BasicNode ทำการรันโปรแกรม nodejs ของเรา โดย พิมพ์ node sayHello.js จะได้ผลลัพธ์ดังภาพ



เริ่มต้นใช้งาน require

จากตัวอย่างที่ผ่านมาผมได้ทำการสร้าง sayHello.js เพื่อทำการส่งข้อความออกมายัง Console Window ว่า 'Hello nodejs world' และทำการรันโปรแกรมที่ไฟล์ sayHello.js มาถึงตอนนี้ผมต้องการจะรันโปรแกรมที่อยู่ไฟล์ sayHello.js โดยทำการเรียกใช้งานผ่านโปรแกรมใน app.js แทน

ทำการสร้างไฟล์ใหม่ ชื่อ app.js อยู่ folder เดียวกันเลยนะครับ และทำการเขียน code โปรแกรมดังนี้

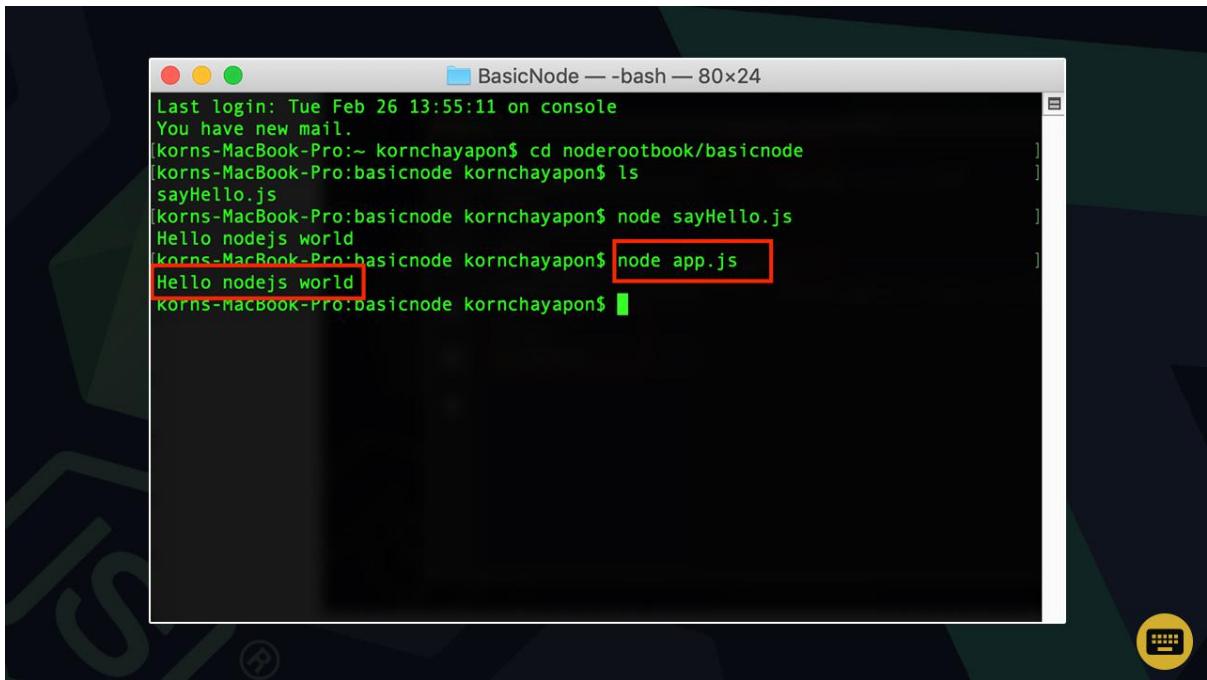


```
sayHello.js — NodeRootBook
EXPLORER
OPEN EDITORS
GROUP 1
JS app.js BasicNode
GROUP 2
JS sayHello.js BasicNode
NODEROOTBOOK
BasicNode
JS app.js
JS sayHello.js
```

```
app.js
1 require('./sayHello') ?
```

```
sayHello.js
1 console.log('Hello nodejs world')
```

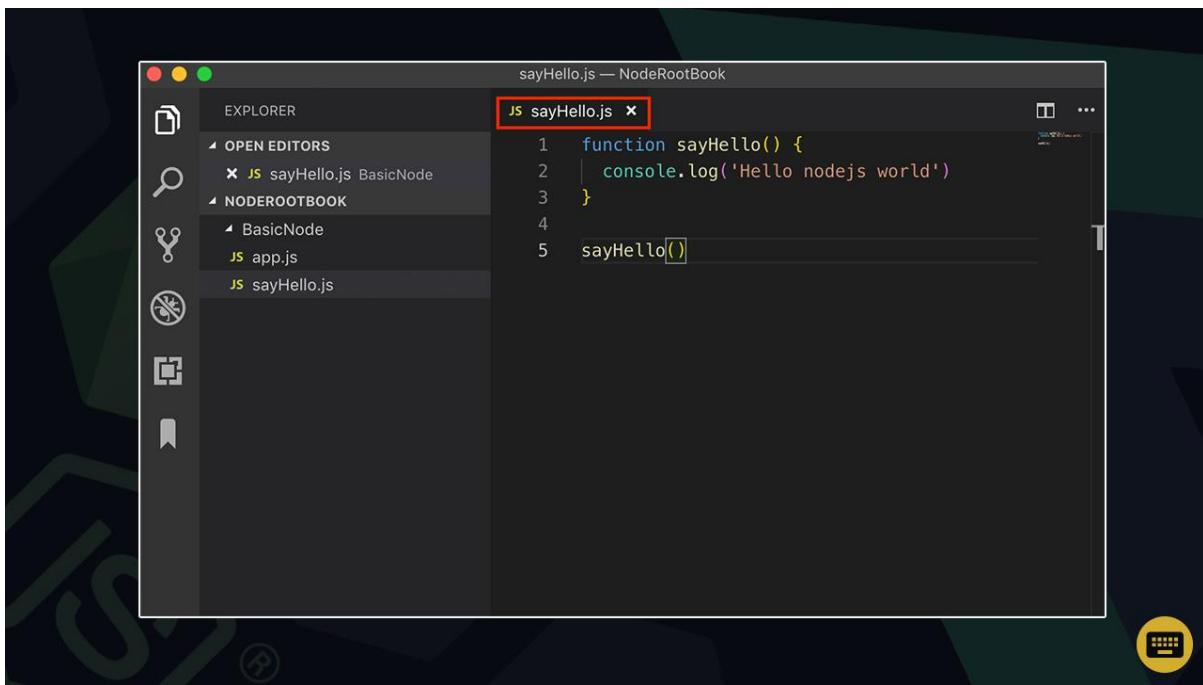
เมื่อเราทำการรันโปรแกรมด้วยคำสั่ง node app.js ในขั้นตอนนี้จะสังเกตว่า เราทำการรันโปรแกรมที่ app.js แทน sayHello.js แต่ด้วยคำสั่ง require ทำให้ app.js ไปทำการดึงโปรแกรมใน sayHello.js มาทำการรันด้วย โดยผลการรันโปรแกรมยังคงเหมือนกับ การรันโปรแกรมที่ sayHello.js เมื่อเดิม แต่การทำงานลักษณะนี้เป็นการดึงโปรแกรมจากไฟล์อื่นเข้ามาใน โปรแกรมที่เรา.rันด้วย require นั่นเอง



การใช้งาน Function บน nodejs เป็นต้น

ในส่วนนี้เราจะมาพูดถึงการใช้งาน function กัน เมื่อโปรแกรมคุณรัน เช่น โปรแกรมที่เราเขียนใน sayHello.js โปรแกรมจะถูกสั่งให้ทำงานตามลำดับ จากบนลงล่าง เมื่อมันเจอคำสั่ง console.log ก็จะทำงานโดยทันที

เราจะมาลองแก้ไข code โปรแกรม sayHello.js ของเราให้เป็นการเขียนแบบ function กันดูนะครับ



จาก code ด้านบน console.log ของผมจะถูกเก็บไว้ใน function ที่ชื่อว่า sayHello และจะถูกเรียกใช้ ที่ sayHello() เมื่อทำการรันโปรแกรมที่ sayHello.js ผลลัพธ์จะเป็นเหมือนกับครั้งที่ผ่านมา แต่จะเป็นการเขียนโปรแกรมแบบ function

ข้อดีของการเขียนโปรแกรมเป็นแบบ function ทำให้เราเรียกใช้งาน function เหล่านั้น ช้า ๆ ได้ โดยไม่ต้องเขียน code ใหม่ และแยกส่วนการทำงานต่าง ๆ ของแต่ละ function ออกเป็นระบบ ทั้งในไฟล์โปรแกรมเดียวกัน และหลาย ๆ ไฟล์ใช้งานร่วมกัน

การเขียนโปรแกรมบนโลกของความเป็นจริง ในหนึ่งโปรแกรมนั้นประกอบด้วย หลาย function และหลายโมดูลมาก การจัดการระบบไฟล์ที่ดีจะทำให้เราจบงาน แก้ไข และปรับปรุงได้ง่ายในอนาคต

เรียกใช้งานฟังก์ชันจากไฟล์โปรแกรมอื่นด้วย **export** **function**

ในส่วนนี้ ผู้จะทำการเรียกใช้งาน function ที่ผู้เขียนใน sayHello.js ผ่าน app.js โดยผู้จะทำการแก้ไข code ใน sayHello.js และ app.js ดังนี้ เพื่อให้เราสามารถไปเรียก function ผ่าน app.js ได้นั่นเอง

The screenshot shows the VS Code interface with two open files:

- sayHello.js** (top editor):

```
1 function sayHello() {  
2   console.log('Hello nodejs world')  
3 }  
4  
5 module.exports = sayHello
```
- app.js** (bottom editor):

```
1 let sayHello = require('./sayHello')  
2 sayHello()
```

Both code snippets are highlighted with red boxes.

หลังจากเราสร้าง function sayHello จาก sayHello.js ออกมายังโกลบาร์จัก โดยการ module.exports ผู้จะต้องไปดึงเข้าที่ app.js โดยการ require และอ้างถึง function ผ่านตัวแปรสำหรับอ้างอิง function โดยผู้ตั้งชื่อว่า say ซึ่งปกติเราจะใช้ชื่อเดียวกับ module ต้นทาง ที่เราทำการ exports มา แต่ในที่นี้ ผู้อ้างถึงผ่านตัวแปรชื่อว่า say เพื่อแสดงให้เห็นว่า เราจะใช้ชื่ออะไรแทนการอ้างถึง function ที่เรา require มารวมกับโปรแกรมเราก็ได้ ไม่ได้มีข้อกำหนดไว้ ซึ่งโดยปกติเพื่อป้องกันการลับสน เรามักใช้ชื่อตัวแปรที่เราอ้างถึง function นั้น ๆ ตามชื่อ function ที่เรา exports มาครับ

เมื่อเราทำการรันโปรแกรม app.js ของเรา ผลการรันโปรแกรมจะเป็นเหมือนเดิม กับขั้นตอนที่ผ่านมาไม่มีอะไรเปลี่ยนแปลง คือ แสดงข้อความ ‘Hello nodejs world’ ออกมาหนึ่ง行

แต่เมื่อเราสังเกตุ code ของโปรแกรมตี ๆ แล้ว เราจะเห็นว่า function sayHello() ของเรานั้น ถูกเรียกใช้ ภายนอกไฟล์ sayHello.js โดยถูกเรียกใช้ผ่าน say() ในไฟล์โปรแกรม app.js นั้นเอง การเขียนโปรแกรมลักษณะนี้เป็น พื้นฐานในการดึง code โปรแกรมในแต่ละไฟล์ที่เราแยกกันไว้ มาใช้งานร่วมกันนั้นเอง โดยการเขียนโปรแกรมในลักษณะนี้ จะกำหนดได้ว่า เราจะทำการรันโปรแกรมที่เขียนเข้ามาในไฟล์ไหน และตอนไหนนั้นเอง

การอ้างถึง function แบบหลายไฟล์ด้วย index.js

เรามาลองสร้างไฟล์และโฟล์เดอร์พร้อมทั้งเขียน code โปรแกรมตามผูกันก่อน แล้วผมจะมาอธิบายว่า มันทำงานยังไงกันนะครับ

อย่างแรกเลยผมจะทำการสร้าง ผมจะทำการสร้างโฟล์เดอร์ ขึ้นมาซึ่งชื่อว่า say จากนั้นผมสร้างไฟล์ขึ้นมาทั้งหมด 3 ไฟล์ คือ index.js, english.js และ thai.js ในโฟล์เดอร์ say ของผม และทำการสร้าง app2.js เพื่อทำเป็น “จุดเริ่มต้นการรันโปรแกรมทั้งหมด (Entry Point)” ของผม

จากนั้นผมทำการเขียน code โปรแกรมทั้งหมดดังนี้

```

thai.js — NodeRootBook
JS index.js x
1 let english = require('../english')
2 let thai = require('../thai')
3
4 module.exports = {
5   english: english,
6   thai: thai
7 }
8

JS thai.js x
1 let say = function () {
2   console.log("Sawadee Krub")
3 }
4
5 module.exports = say

JS english.js x
1 let say = function () {
2   console.log("Hello")
3 }
4 module.exports = say

JS app2.js x
1 let say = require('./say')
2 say.english()
3 say.thai()

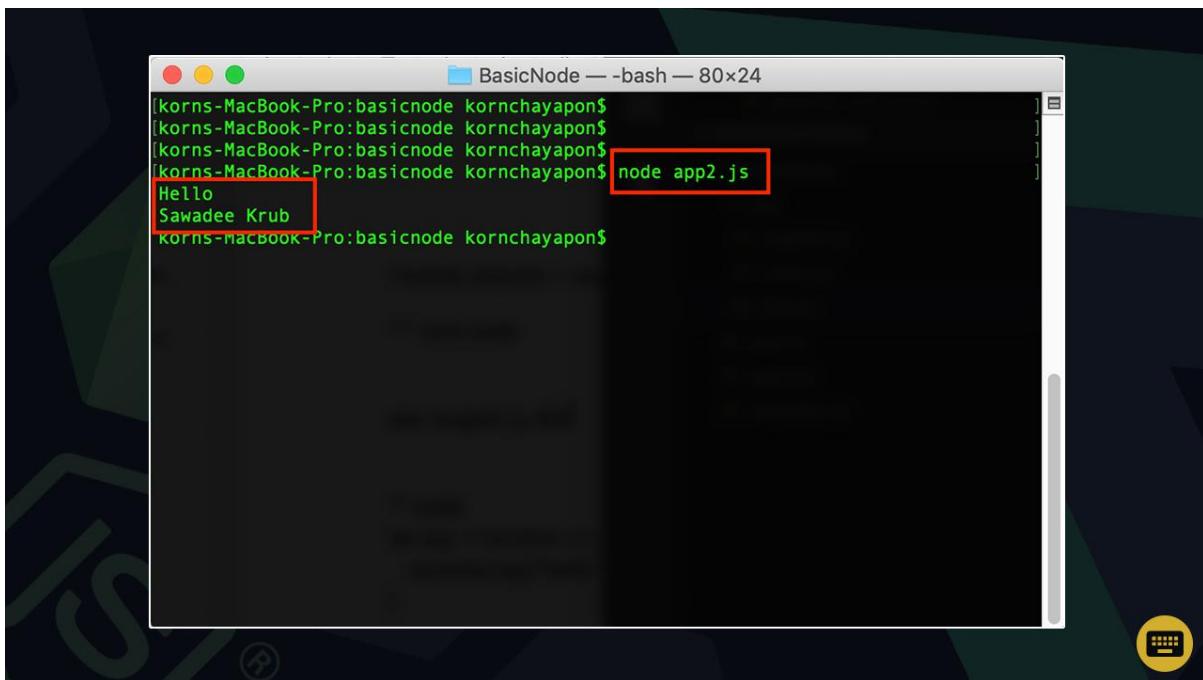
```

p15 การเรียกใช้งาน *function* แบบดึงมาร่วม helyay ไฟล์

การทำงานของการเขียนโปรแกรมที่เราเขียนขึ้น มีการทำงานดังนี้

app2.js จะ ดึงไฟล์ทั้งหมดเข้ามาโดยการอ้างถึง ชื่อโฟล์เดอร์ ./say ของเรา
จากนั้นจะทำงานที่ index.js เพื่อดึงไฟล์ที่เกี่ยวข้องมาใช้งาน แต่เราจะเห็นกรอบ
เขียว ๆ คือ ชื่อโฟล์เดอร์ กับ ชื่อฟังก์ชัน ทั้งใน thai.js และ english.js รวมถึงการ
export module ก็จะต้องเหมือนกันด้วย และทำการเลือกฟังก์ชันที่จะใช้งานตามที่
index.js exports ออกมานะ

จากนั้นเราทำการรันโปรแกรมที่ app2.js ด้วยคำสั่ง nodejs app2.js จะได้ผลการ
ทำงานของโปรแกรมดังนี้ครับ



```
korns-MacBook-Pro:basicnode kornchayapon$  
korns-MacBook-Pro:basicnode kornchayapon$  
korns-MacBook-Pro:basicnode kornchayapon$  
korns-MacBook-Pro:basicnode kornchayapon$ node app2.js  
Hello  
Sawadee Krub  
korns-MacBook-Pro:basicnode kornchayapon$
```

p16 ผลการรันโปรแกรม จากการเรียกใช้งาน *function* แบบตีงมาร่วมหลายไฟล์

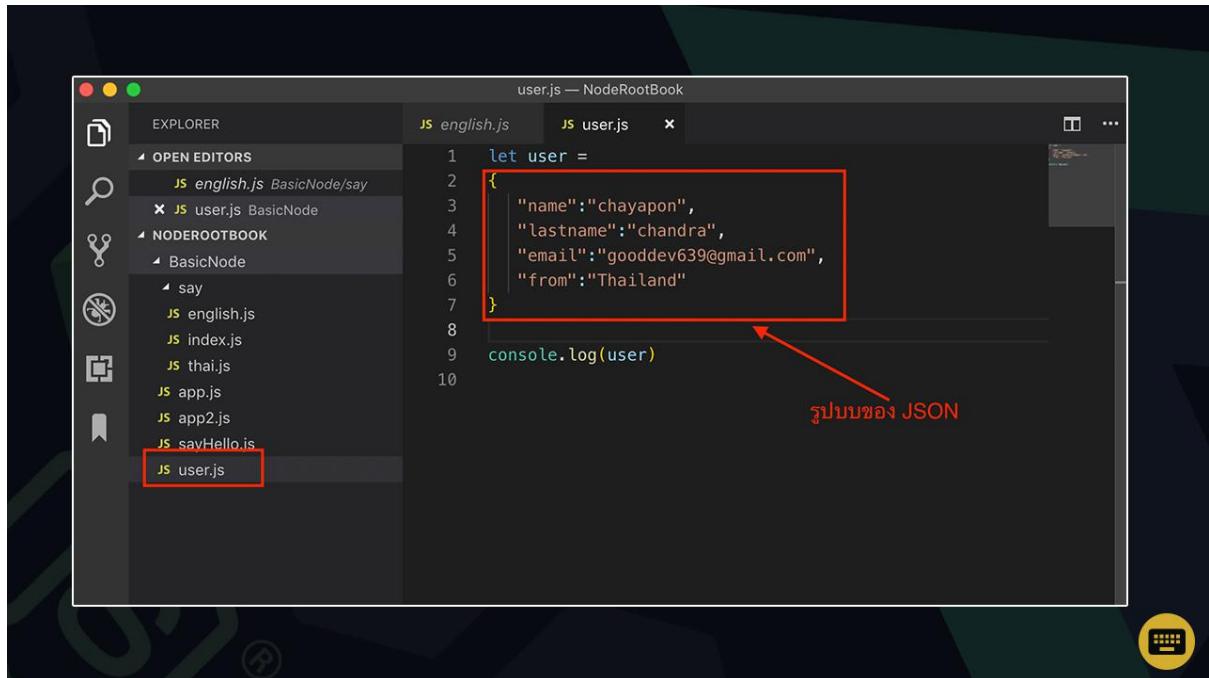
จริง ๆ แล้วการเขียนโปรแกรมด้วย node นั้นมีเนื้อหาเยอะมาก แต่ผมได้นำมาในสิ่งที่จำเป็นต่องานของเรามาสอน เพื่อให้เราสามารถสร้างงานได้ เมื่อเรามีสิ่งที่ต้องการสร้าง เราจะสามารถสร้างได้โดยไม่ต้องเขียนโค้ดเอง แต่ถ้าหากเราต้องการสร้างงานที่ซับซ้อน เช่น เว็บไซต์ หรือแอปพลิเคชัน ที่ต้องมีการเชื่อมต่อฐานข้อมูล หรือต้องมีการรับส่งข้อมูลจากผู้ใช้ อาจจะต้องใช้ภาษาอื่นๆ เช่น JavaScript หรือ Python แทน

ผมขอจบทริ�การใช้งาน node.js เนื่องต้นไว้เพียงเท่านี้ก่อนนะครับ ณ ตอนนี้ถ้าเข้าใจในสิ่งที่ผมสอนมา คุณก็พร้อมที่จะเริ่มเรียนการเขียนเว็บไซต์ ในส่วนของ server ด้วย node.js แล้วครับ

ทำความรู้จัก JSON

JSON ย่อมาจาก JavaScript Object Notation สำหรับพนมมันคือ รูปแบบการจัดเก็บข้อมูลของ JavaScript จัดเก็บในลักษณะ JSON Object จำให้ชื่นใจเลย นะครับ ส่วนใหญ่ ถ้าอ่านค่าอุปกรณ์แล้วมันบอกว่าเป็น object ให้เดาว่ามันเป็น JSON ไว้ก่อนครับ โดยการเก็บข้อมูลในรูปแบบของ JSON นั้นสามารถเก็บได้ทั้งแบบ เก็บลงตัวแปร (เก็บในหน่วยความจำ) และ เป็นไฟล์ .json (บันทึกเป็นไฟล์ข้อมูล)

เรามาลองเล่นกับ JSON กันดูครับ โดยผมทำการสร้างไฟล์ user.js ขึ้นมา แล้วทำการเขียน code โปรแกรมดังนี้ครับ



The screenshot shows a dark-themed instance of Visual Studio Code. In the center-right is the code editor window titled "user.js — NodeRootBook". It contains the following JavaScript code:

```
1 let user =
2 {
3     "name": "chayapon",
4     "lastname": "chandra",
5     "email": "gooddev639@gmail.com",
6     "from": "Thailand"
7 }
8 console.log(user)
9
10
```

A red rectangular box highlights the JSON object definition from line 2 to line 7. A red arrow points from the text "รูปแบบของ JSON" (The format of JSON) to the start of this highlighted block.

ในการเขียนโปรแกรมครั้งนี้ผมเพียงแสดงให้เห็น ถึงรูปแบบของเก็บข้อมูลแบบ JSON ครับ เมื่อทำการรันโปรแกรม user.js ด้วยคำสั่ง node user.js จะได้ผลลัพธ์ดังนี้

A screenshot of a macOS terminal window titled "BasicNode — bash — 80x24". The terminal shows the following command-line session:

```
[korns-MacBook-Pro:basicnode kornchayapon$ node app2.js
Hello
Sawadee Krub
[korns-MacBook-Pro:basicnode kornchayapon$ ls
app.js      app2.js  say          sayHello.js  user.js
You have new mail in /var/mail/kornchayapon
[korns-MacBook-Pro:basicnode kornchayapon$ node user.js
{ name: 'chayapon',
  lastname: 'chandra',
  email: 'gooddev639@gmail.com',
  from: 'Thailand'
}
[korns-MACBOOK-Pro:basicnode Kornchayapon$ ]
```

The output of the "user.js" script is highlighted with a red box.

จะสังเกตว่า โปรแกรมที่เราเขียนจะ print out json ทั้งหมดออกมามาเลย เมื่อ json เป็น string ตัวนึง แต่เดียวก่อนครับ ถ้าผมแก้ code เป็นดังนี้ล่ะ

A screenshot of the VS Code interface showing the "user.js" file in the "EXPLORER" sidebar. The code in the editor is:

```
let user =
{
  "name": "chayapon",
  "lastname": "chandra",
  "email": "gooddev639@gmail.com",
  "from": "Thailand"
}
console.log('Name: ' + user.name + ' ' + user.lastname)
```

The line "console.log('Name: ' + user.name + ' ' + user.lastname)" is highlighted with a red box.

การอ้างถึงตัวแปรลักษณะนี้ ก็คือ การถึงค่า หรือ value ภายใน JSON Object ของเรานั้นเองเมื่อทำการรันโปรแกรม user.js ของผม จะได้ผลลัพธ์ดังนี้ครับ

```

BasicNode — bash — 80x24
[korns-MacBook-Pro:basicnode kornchayapon$ node app2.js
Hello
Sawadee Krub
[korns-MacBook-Pro:basicnode kornchayapon$ ls
app.js      app2.js    say          sayHello.js   user.js
You have new mail in /var/mail/kornchayapon
[korns-MacBook-Pro:basicnode kornchayapon$ node user.js
{ name: 'chayapon',
  lastname: 'chandra',
  email: 'gooddev639@gmail.com',
  from: 'Thailand'}
[korns-MacBook-Pro:basicnode kornchayapon$ node user.js
Name: chayapon chandra
You have new mail in /var/mail/kornchayapon
korns-MacBook-Pro:basicnode kornchayapon$ ]

```

เมื่อเราทำการอ่านค่าจาก JSON ได้แล้ว ตอนนี้ผมจะมาลองเปลี่ยนแปลงค่าใน JSON ของเราดูนะครับ ว่าสามารถทำได้มั้ย โดยผมทำการแก้ไข code ใน user.js ดังนี้ครับ

```

user.js — NodeRootBook
EXPLORER
OPEN EDITORS
JS user.js BasicNode
NODEROOTBOOK
BasicNode
say
english.js
index.js
thai.js
app.js
app2.js
sayHello.js
user.js
1 let user =
2 {
3   "name":"chayapon",
4   "lastname":"chandra",
5   "email":"gooddev639@gmail.com",
6   "from":"Thailand"
7 }
8
9 console.log('Name: ' + user.name + ' ' + user.lastname)
10
11 user.name = "Steve"
12 user.lastname = "Job"
13
14 console.log('New Data - Name: ' + user.name + ' ' + user.lastname)
15
16

```

บรรทัดแรกจะเป็นผลการรันโปรแกรมจากค่าที่เรากำหนด

บรรทัดที่สองจะค่าที่เราเปลี่ยนแปลงค่าภายใน json ของเรา เห็นมั้ยครับ ใช้งานง่ายมากเลยครับ และทำการรันโปรแกรมของเรา จะได้ผลลัพธ์ดังนี้ครับ

```
[korns-MacBook-Pro:basicnode kornchayapon$ [korns-MacBook-Pro:basicnode kornchayapon$ [korns-MacBook-Pro:basicnode kornchayapon$ [korns-MacBook-Pro:basicnode kornchayapon$ node app2.js
Hello
Sawadee Krub
[korns-MacBook-Pro:basicnode kornchayapon$ ls
app.js      app2.js    say          sayHello.js   user.js
You have new mail in /var/mail/kornchayapon
[korns-MacBook-Pro:basicnode kornchayapon$ node user.js
{
  name: 'chayapon',
  lastname: 'chandra',
  email: 'gooddev639@gmail.com',
  from: 'Thailand'
}
[korns-MacBook-Pro:basicnode kornchayapon$ node user.js
Name: chayapon chandra
You have new mail in /var/mail/kornchayapon
[korns-MacBook-Pro:basicnode kornchayapon$ node user.js
Name: chayapon chandra
New Data - Name: Steve Job
[korns-MacBook-Pro:basicnode kornchayapon$ ]
```

การใช้งาน JSON ในรูปแบบของ Array

ในการใช้งานจริงนั้น เวลาเราเก็บข้อมูล เช่น ข้อมูลผู้ใช้งานนั้น บ่อยครั้งที่มีผู้ใช้งานที่เราต้องการเก็บข้อมูลหลายคน หรือเราเรียกว่า หลายแคล หรือ หลาย rows

เราสามารถใช้งาน json แบบ array ได้นี้ครับ

ผมจะสร้างไฟล์ users.js ขึ้นมานะครับ และการเขียน code ดังนี้ครับ

The screenshot shows the VS Code interface with the 'users.js' file open in the editor. The code defines two objects: 'user[0]' and 'user[1]', each with properties like name, lastname, email, and from. It then logs the names of both users to the console.

```
let user = [
  {
    "name": "chayapon",
    "lastname": "chandra",
    "email": "gooddev639@gmail.com",
    "from": "Thailand"
  },
  {
    "name": "Job",
    "lastname": "Steve",
    "email": "steve.job@gmail.com",
    "from": "USA"
  }
]

console.log('Name: ' + user[0].name + ' ' + user[0].lastname)
console.log('Name: ' + user[1].name + ' ' + user[1].lastname)
```

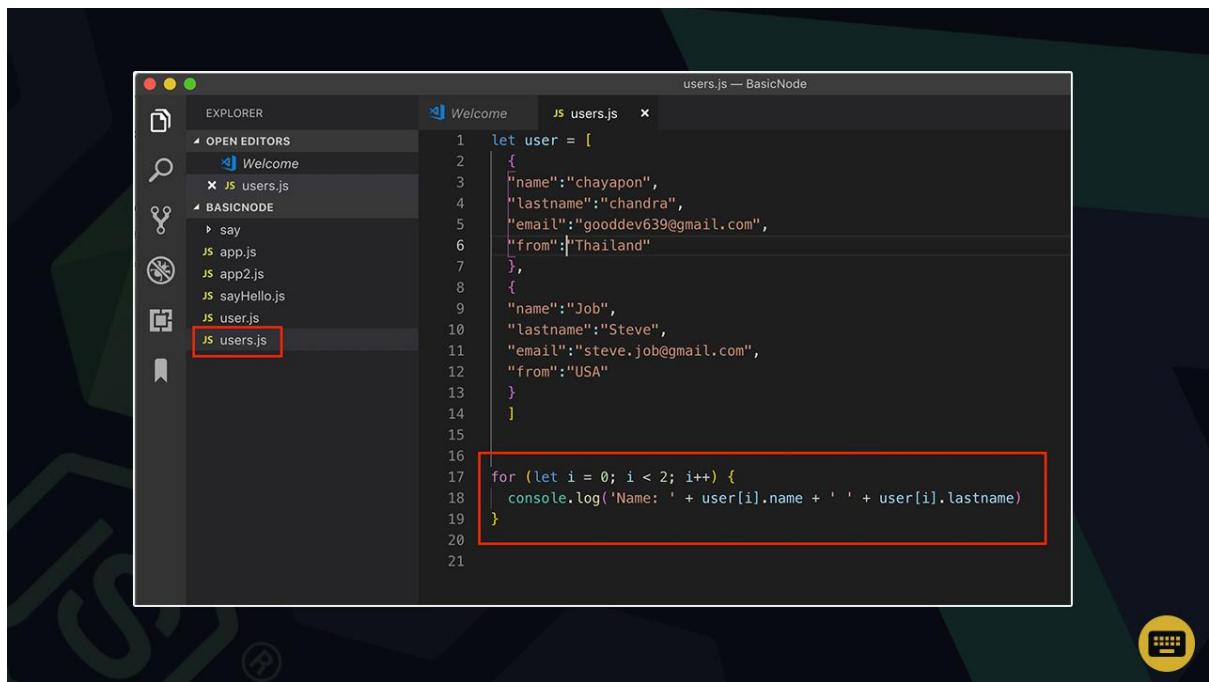
เมื่อผู้ใช้ทำการรันโปรแกรม จะได้ผลลัพธ์ดังนี้ครับ

The screenshot shows a terminal window with the command 'node users.js' run. The output displays the names of the two users defined in the code.

```
cd ..
cd ..
ls
node users.js
Name: chayapon chandra
Name: Job Steve
```

เข้าสังเกตุของ code ชุดนี้ เราจะทำการกำหนดค่าให้กับ object สองซัด โดยแยกจากเครื่องหมาย ',' และใส่ [] มาครอบไว้ ซึ่งคือ array นั่นเอง

ลองประยุกใช้ loop for ในการแสดงผลลัพธ์ครับ ผู้ลองปรับ code อีกนิดหน่อยดังนี้ครับ

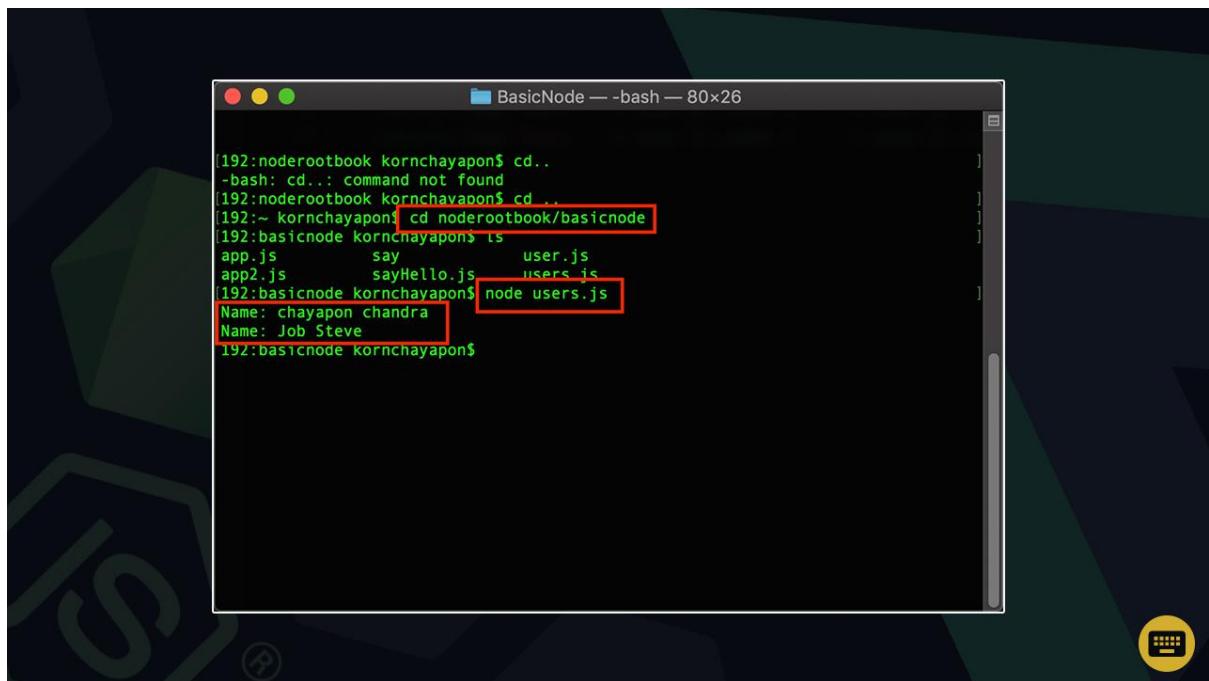


```
let user = [
  {
    "name": "chayapon",
    "lastname": "chandra",
    "email": "gooddev639@gmail.com",
    "from": "Thailand"
  },
  {
    "name": "Job",
    "lastname": "Steve",
    "email": "steve.job@gmail.com",
    "from": "USA"
  }
]

for (let i = 0; i < 2; i++) {
  console.log('Name: ' + user[i].name + ' ' + user[i].lastname)
}
```

เมื่อทำการรันจะได้ผลลัพธ์ เหมือนเดิม คือ

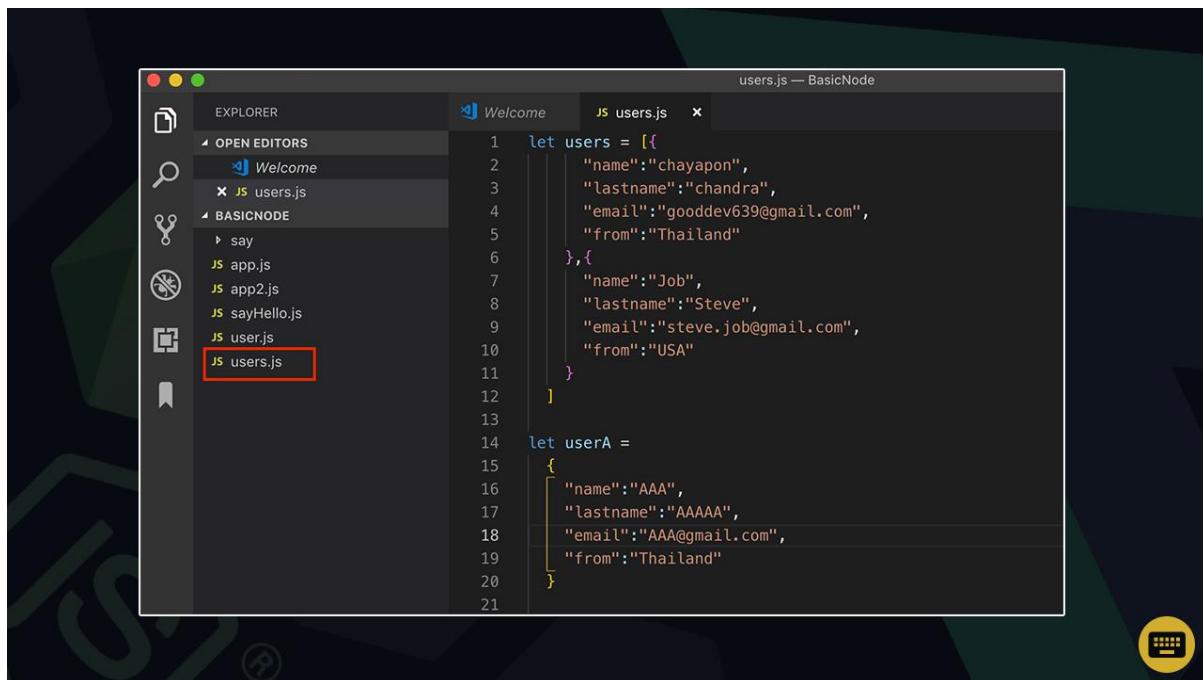
แต่การใช้ loop for มาช่วยนี้ ทำให้ง่ายเวลาข้อมูลเยอะ ๆ ไม่ต้องเขียน code เพิ่มไปตามจำนวนข้อมูลที่เพิ่มขึ้นนั่นเอง



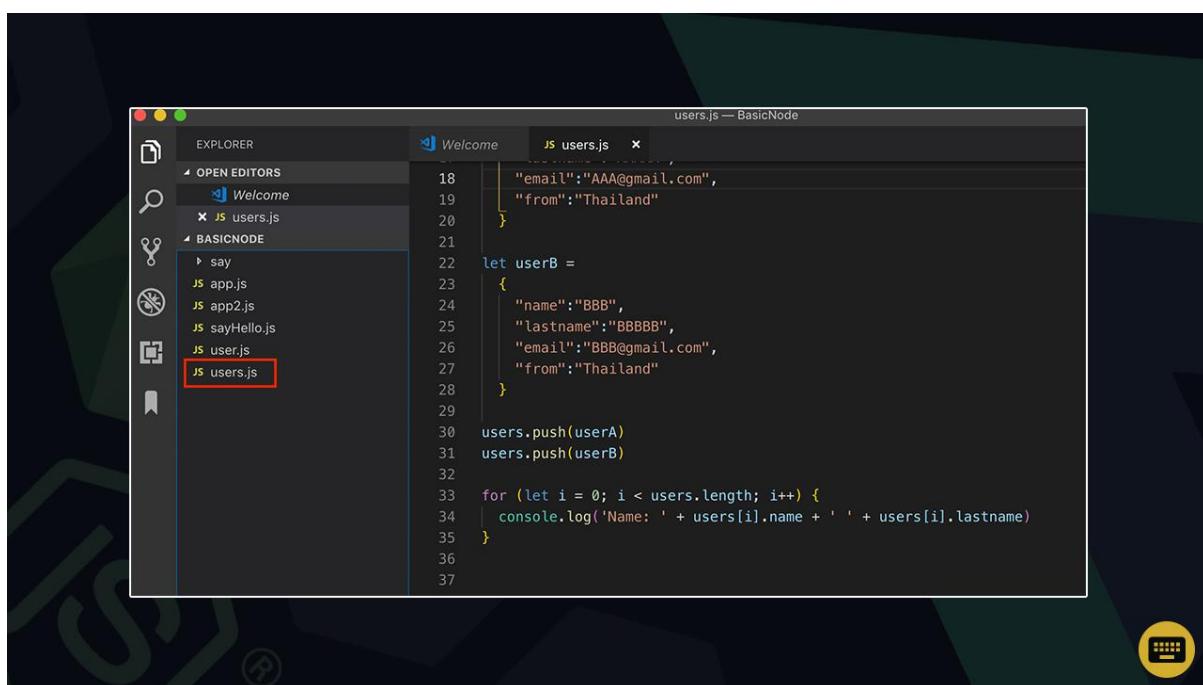
```
192:noderootbook kornchayapon$ cd..
-bash: cd..: command not found
192:noderootbook kornchayapon$ cd ..
192:~ kornchayapon$ cd noderootbook/basicnode
192:basicnode kornchayapon$ ls
app.js      say          user.js
app2.js     sayHello.js  users.js
192:basicnode kornchayapon$ node users.js
Name: chayapon chandra
Name: Job Steve
192:basicnode kornchayapon$
```

การเพิ่มข้อมูลใน json array ด้วย push array

ในตัวอย่างนี้ ผู้จะทำการแก้ไข code เพื่อทำการเพิ่มข้อมูลเข้าไปใน json array ของเราด้วย คำสั่ง push และครับ ผู้ทำการแก้ไข users.js ดังนี้ครับ

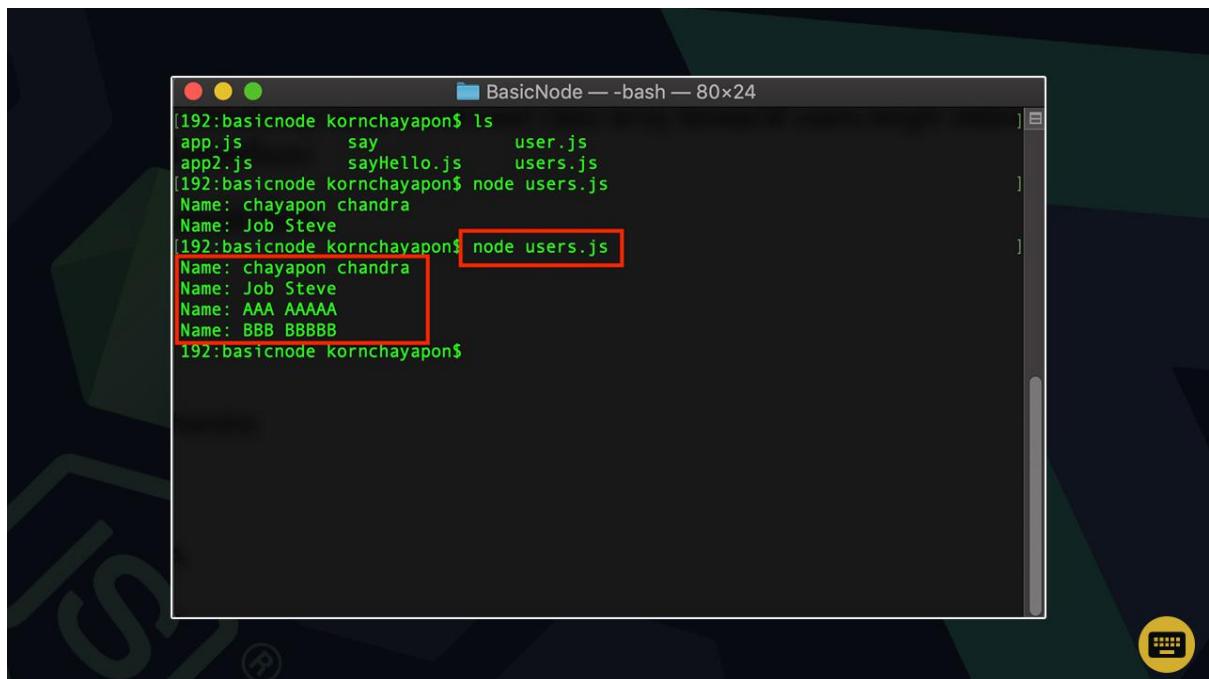


```
1 let users = [
2   {
3     "name": "chayapon",
4     "lastname": "chandra",
5     "email": "gooddev639@gmail.com",
6     "from": "Thailand"
7   },
8   {
9     "name": "Job",
10    "lastname": "Steve",
11    "email": "steve.job@gmail.com",
12    "from": "USA"
13  }
14
15 let userA =
16 {
17   "name": "AAA",
18   "lastname": "AAAAA",
19   "email": "AAA@gmail.com",
20   "from": "Thailand"
21 }
```



```
18   "email": "AAA@gmail.com",
19   "from": "Thailand"
20 }
21
22 let userB =
23 {
24   "name": "BBB",
25   "lastname": "BBBBB",
26   "email": "BBB@gmail.com",
27   "from": "Thailand"
28 }
29
30 users.push(userA)
31 users.push(userB)
32
33 for (let i = 0; i < users.length; i++) {
34   console.log('Name: ' + users[i].name + ' ' + users[i].lastname)
35 }
36
37
```

ในตัวอย่างนี้ผมได้ทำการ เพิ่ม user AAA และ BBB เข้าไปใน users ของผมผ่าน การ push แล้วการแสดงผลผ่าน loop for โดยสังเกตว่าผมอ่านค่าความยาวของ array ของผมได้ users.length เมื่อตอน การใช้งาน array ทั่วไปเลยนั้นเอง เมื่อทำการรันจะได้ผลลัพธ์ดังนี้ครับ



The screenshot shows a terminal window titled "BasicNode — bash — 80x24". The command "ls" is run, showing files: app.js, say, user.js, app2.js, sayHello.js, and users.js. Then, "node users.js" is run, outputting the following names:

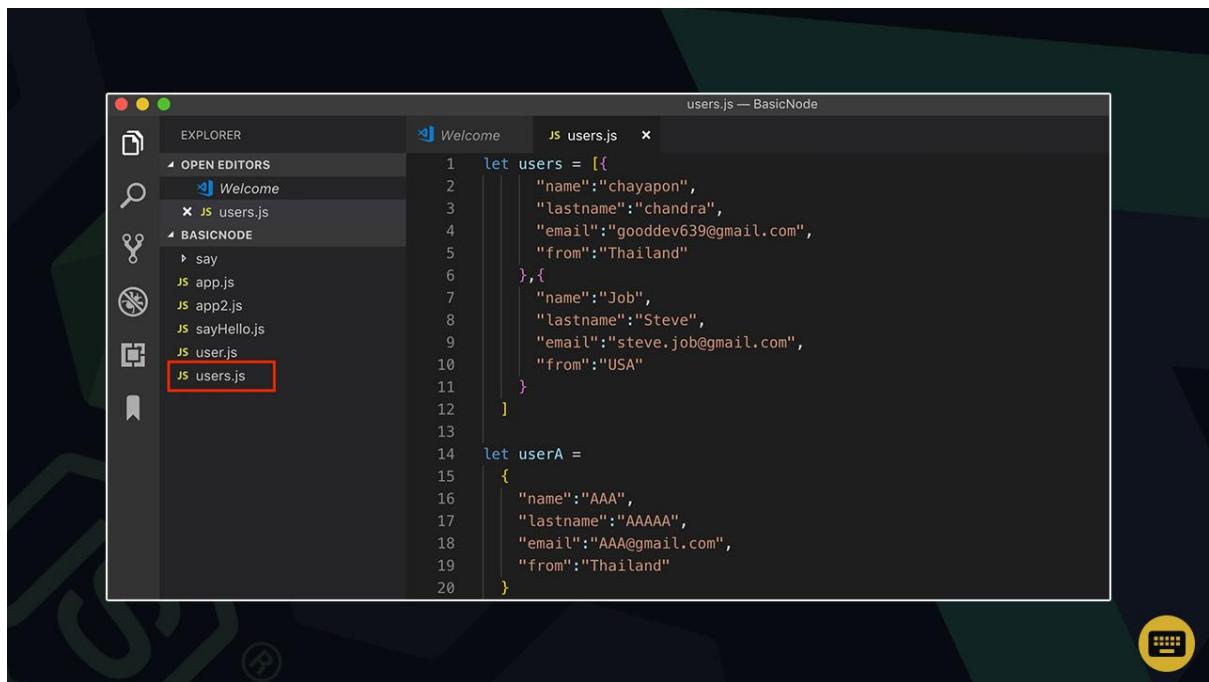
```
(192:basicnode kornchayapon$ ls
app.js      say          user.js
app2.js     sayHello.js   users.js
192:basicnode kornchayapon$ node users.js
Name: chayapon chandra
Name: Job Steve
192:basicnode kornchayapon$ node users.js
Name: chayapon chandra
Name: Job Steve
Name: AAA AAAAAA
Name: BBB BBBBBB
192:basicnode kornchayapon$
```

The output from the second "node users.js" command is highlighted with a red box.

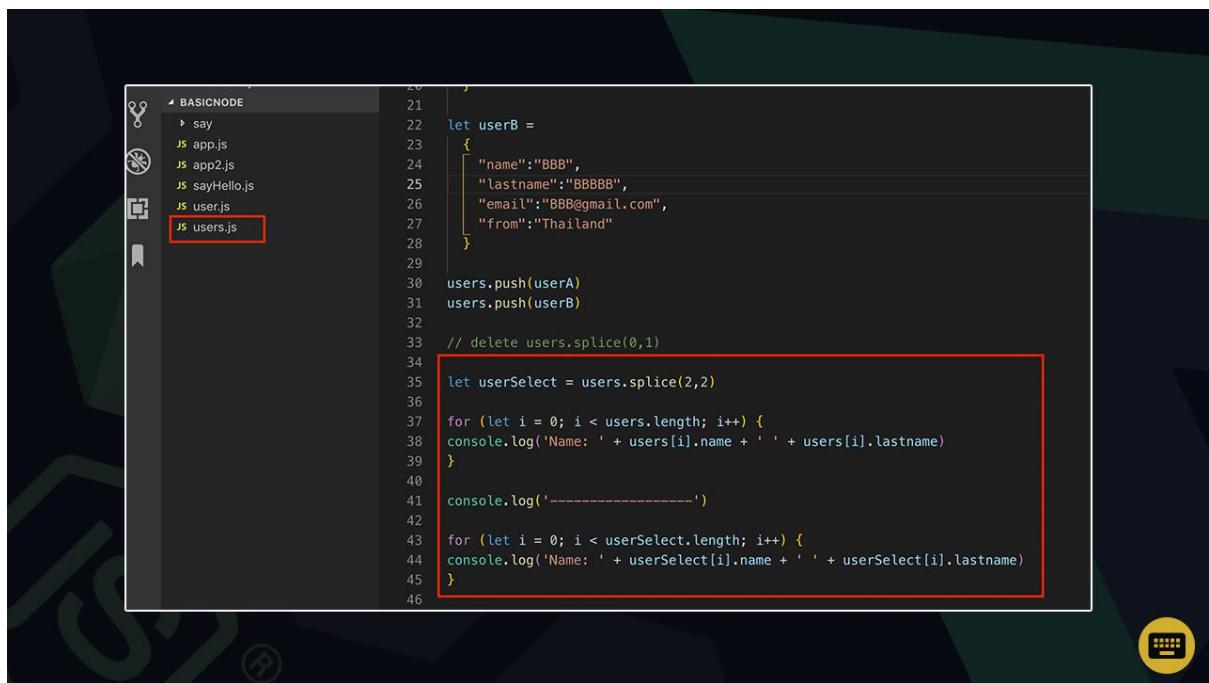
การเขียนโปรแกรมเพื่อเลือกช่วง Array ใน JSON array มาใช้งาน (**splice**)

คำสั่งที่นำเสนอจะอีกคำสั่งนึง นั่นคือ slice คือการตัดอะเรย์ หรือเลือก array ใช้ช่วงที่เราต้องการแสดงผล หรือ ลบช่วง array ที่เราเลือกออกจาก json ครับ

โดยผมจะทำการ เขียน code เพิ่มเติมดังนี้ครับ

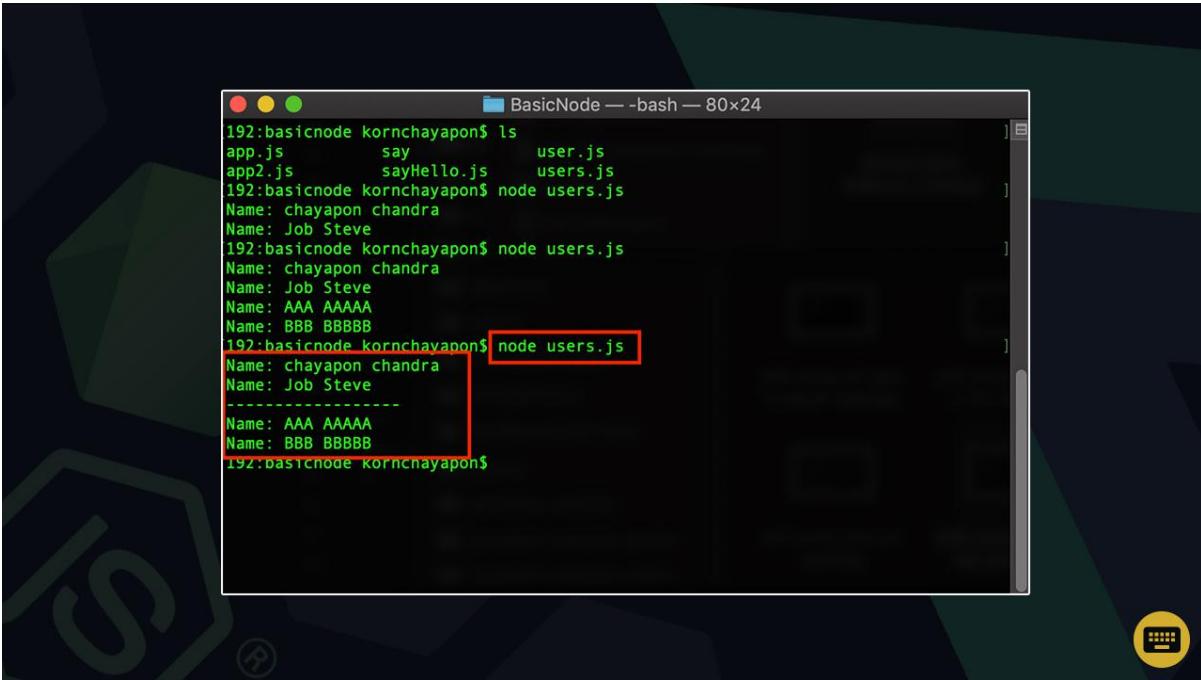


```
1 let users = [
2   "name":"chayapon",
3   "lastname":"chandra",
4   "email": "gooddev639@gmail.com",
5   "from": "Thailand"
6 },
7   "name": "Job",
8   "lastname": "Steve",
9   "email": "steve.job@gmail.com",
10  "from": "USA"
11 ]
12
13
14 let userA =
15 {
16   "name": "AAA",
17   "lastname": "AAAAA",
18   "email": "AAA@gmail.com",
19   "from": "Thailand"
20 }
```



```
21 }
22 let userB =
23 {
24   "name": "BBB",
25   "lastname": "BBBBB",
26   "email": "BBB@gmail.com",
27   "from": "Thailand"
28 }
29
30 users.push(userA)
31 users.push(userB)
32
33 // delete users.splice(0,1)
34
35 let userSelect = users.splice(2,2)
36
37 for (let i = 0; i < users.length; i++) {
38   console.log('Name: ' + users[i].name + ' ' + users[i].lastname)
39 }
40
41 console.log('-----')
42
43 for (let i = 0; i < userSelect.length; i++) {
44   console.log('Name: ' + userSelect[i].name + ' ' + userSelect[i].lastname)
45 }
```

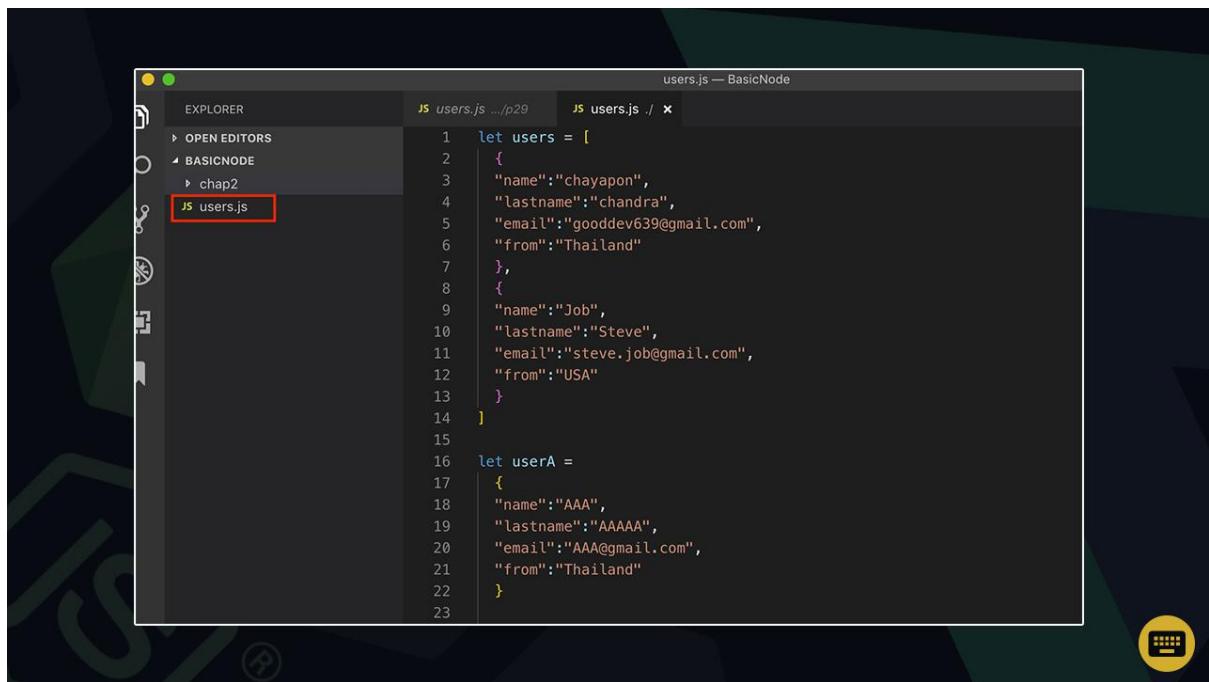
เมื่อทำการรันจะได้ผลลัพธ์ ดังนี้ คือ



The screenshot shows a terminal window titled "BasicNode — -bash — 80x24". The command "ls" is run, listing files: app.js, say, user.js, app2.js, sayHello.js, users.js. Then, "node users.js" is run twice. The output shows the contents of the "users.js" file, which contains an array of objects representing users. The first run shows the full array, and the second run shows a portion of it, indicated by a red box around the first two entries. The array structure is as follows:

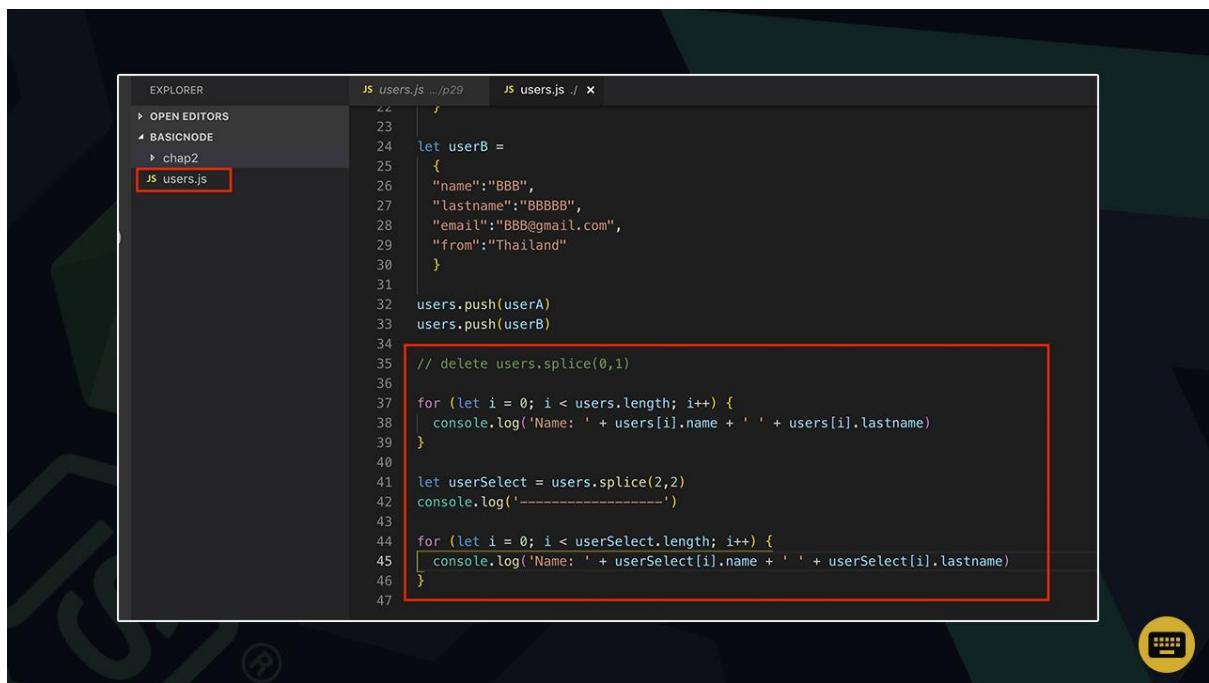
```
192:basicnode kornchayapon$ ls
app.js      say          user.js
app2.js     sayHello.js   users.js
192:basicnode kornchayapon$ node users.js
Name: chayapon chandra
Name: Job Steve
192:basicnode kornchayapon$ node users.js
Name: chayapon chandra
Name: Job Steve
Name: AAA AAAAA
Name: BBB BBBBB
192:basicnode kornchayapon$ node users.js
Name: chayapon chandra
Name: Job Steve
-----
Name: AAA AAAAA
Name: BBB BBBBB
192:basicnode kornchayapon$
```

จาก code ส่วนนี้ผมทำการตัด array users ออกมานแล้วเอาไปไว้ที่ userSelect โดย เริ่มจากตำแหน่งที่ 2 ไป 2 ແກ (เริ่ม, จำนวน) นั่นเอง โดยเราต้องจำไว้ เมื่อ ถูกตัดไปแล้ว json ตัวที่ถูกตัดจะหายออกไปเลย ไม่ได้คงอยู่เหมือนเดิมนะครับ ถ้า ผิดพลาดในการแสดงและการตัดใหม่เป็น



```
let users = [
  {
    "name": "chayapon",
    "lastname": "chandra",
    "email": "gooddev639@gmail.com",
    "from": "Thailand"
  },
  {
    "name": "Job",
    "lastname": "Steve",
    "email": "steve.job@gmail.com",
    "from": "USA"
  }
]

let userA = {
  "name": "AAA",
  "lastname": "AAAAA",
  "email": "AAA@gmail.com",
  "from": "Thailand"
}
```



```
let userB = {
  "name": "BBB",
  "lastname": "BBBBB",
  "email": "BBB@gmail.com",
  "from": "Thailand"
}

users.push(userA)
users.push(userB)

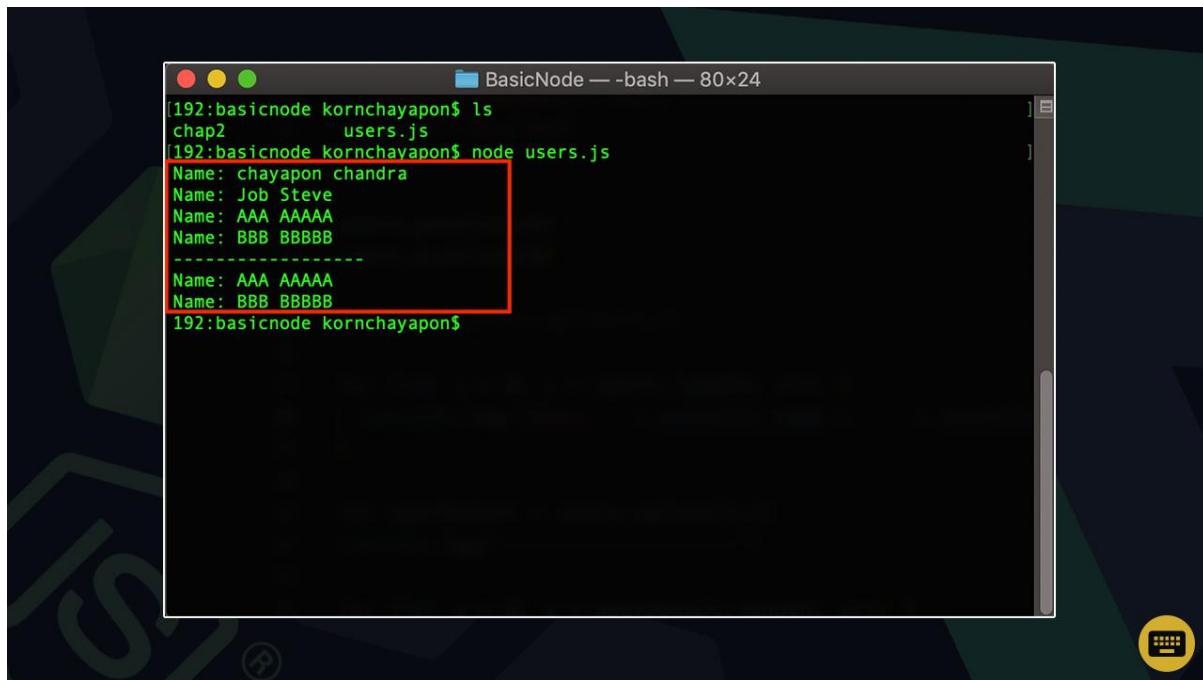
// delete users.splice(0,1)

for (let i = 0; i < users.length; i++) {
  console.log('Name: ' + users[i].name + ' ' + users[i].lastname)
}

let userSelect = users.splice(2,2)
console.log('-----')

for (let i = 0; i < userSelect.length; i++) {
  console.log('Name: ' + userSelect[i].name + ' ' + userSelect[i].lastname)
}
```

จะได้ผลลัพธ์เป็น



```
[192:basicnode kornchayapon$ ls
chap2      users.js
[192:basicnode kornchayapon$ node users.js
Name: chayapon chandra
Name: Job Steve
Name: AAA AAAAA
Name: BBB BBBBB
-----
Name: AAA AAAAA
Name: BBB BBBBB
192:basicnode kornchayapon$
```

เพราะว่าแสดงผลก่อนถูกตัด array นั่นเอง

การใช้การเลือกช่วงข้อมูลแบบ slice

การคำสั่ง slice จะเหมือนกับการทำงานด้วยคำสั่ง splice ทุกประการเพียงแต่ การใช้คำสั่ง slice ข้อมูลต้นแบบจะไม่หายไป เมื่อถูกเลือกช่วงข้อมูลมาแล้วนั่นเอง

การลบข้อมูลใน JSON Array

การลบ array ใน json ของเรานั้น ทำได้โดยการ ใช้คำสั่ง delete ร่วมกับคำสั่ง splice โดยทำการเขียน code ได้ดังนี้

The screenshot shows the VS Code interface with the 'users.js' file open in the editor. The code defines two arrays: 'users' containing two objects with names, lastnames, emails, and from locations, and 'userA' containing a single object with similar details. A red box highlights the file name 'users.js' in the Explorer sidebar.

```
1 let users = [
2   {
3     "name": "chayapon",
4     "lastname": "chandra",
5     "email": "gooddev639@gmail.com",
6     "from": "Thailand"
7   },
8   {
9     "name": "Job",
10    "lastname": "Steve",
11    "email": "steve.job@gmail.com",
12    "from": "USA"
13  }
14]
15
16 let userA =
17 {
18   "name": "AAA",
19   "lastname": "AAAAA",
20   "email": "AAA@gmail.com",
21   "from": "Thailand"
22 }
```

The screenshot shows the VS Code interface with the 'users.js' file open in the editor. The code adds two users ('userA' and 'userB') to the 'users' array, logs their names to the console, and then uses the 'splice' method to remove the first user from the array. A red box highlights the 'delete users.splice(0,1)' line.

```
23
24 let userB =
25 {
26   "name": "BBB",
27   "lastname": "BBBBB",
28   "email": "BBB@gmail.com",
29   "from": "Thailand"
30 }
31
32 users.push(userA)
33 users.push(userB)
34
35 for (let i = 0; i < users.length; i++) {
36   console.log('Name: ' + users[i].name + ' ' + users[i].lastname)
37 }
38
39 delete users.splice(0,1)
40
41 console.log('\n*** After Delete\n')
42
43 for (let i = 0; i < users.length; i++) {
44   console.log('Name: ' + users[i].name + ' ' + users[i].lastname)
45 }
```

ພວເຮັນ code ຈະໄດ້ຜລລັບທີ່ດັ່ງນີ້

```
192:basicnode kornchayapon$ node users.js
Name: chayapon chandra
Name: Job Steve
Name: AAA AAAAA
Name: BBB BBBBB

*** After Delete

Name: Job Steve
Name: AAA AAAAA
Name: BBB BBBBB
192:basicnode kornchayapon$
```

สังเกตุเห็นว่า ถ้าเราแก้ไขของเราว่าจะถูกลบออกไป เพราะเราทำการลบ index ที่ 0 และจำนวนการลบคือ 1 แล้วครับ

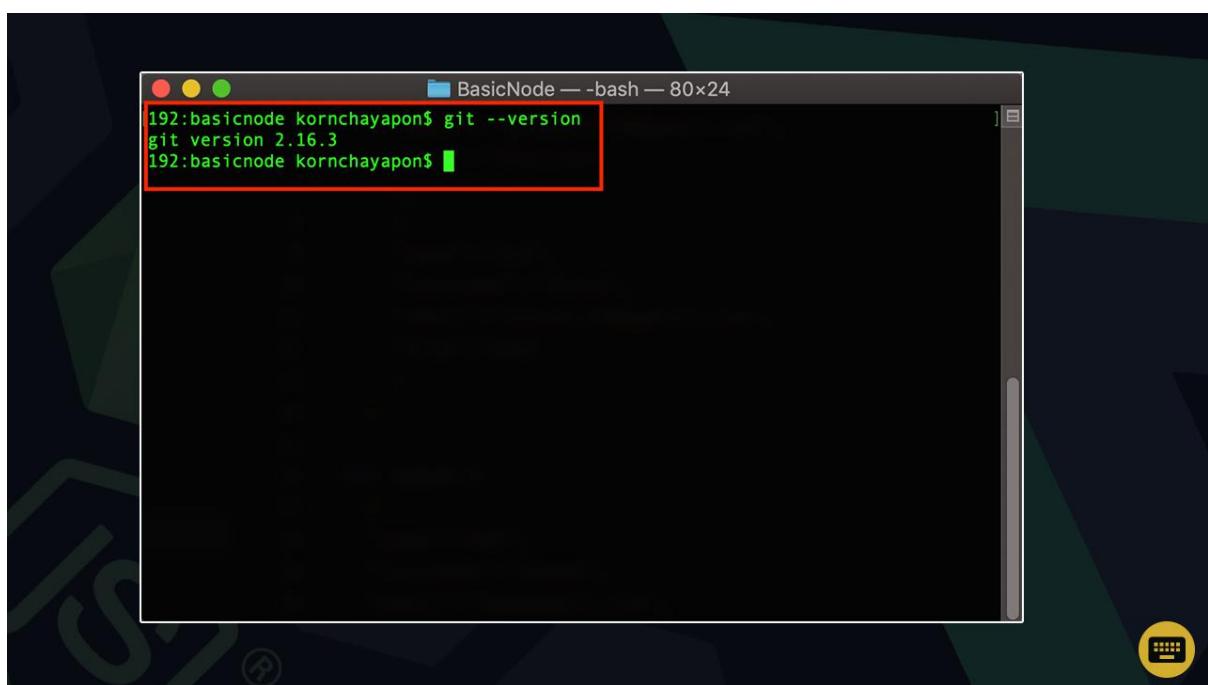
มาถึงตรงนี้เรามีความรู้เบื้องต้น เกี่ยวกับ nodejs และ json เพียงพอที่จะเริ่มเรียน การทำเว็บไซต์ แบบ full stack ด้วย nodejs และ vuejs แล้วครับ

บทที่ 3 ทำความรู้จักโปรแกรมจัดการเรื่องการ Back Up และ version กันคับ

ทำความรู้จัก repository และ git กันก่อนครับ

ก่อนเราจะเริ่มต้นกับ git เรามาทำความรู้จักกับ repository กันก่อน โดย repository คือ คลังหรือที่เก็บ code หรือ file ในโปรเจคของเรา โดยเก็บเอาไว้แบบแยก version เอาไว้ และสามารถ แสดงการเปรียบเทียบแต่ละ version กันได้ Repository เป็นเพียงรูปแบบ และพื้นที่จัดเก็บ เราจะจัดการมันได้ต้องมี git โดย git จะเป็นตัวจัดการ repository หรือ code ที่เราเก็บไว้นั่นเอง

ก่อนที่เราจะไปใช้งาน github.com เราต้องมี โปรแกรม git เลี้ยงก่อน โดยเราทำการตรวจสอบก่อนว่า ภายในเครื่องที่เราใช้งานนั้นได้ทำการติดตั้งไว้หรือยัง โดยทำการพิมพ์คำสั่ง



The screenshot shows a terminal window titled "BasicNode — bash — 80x24". The command "git --version" is entered and its output "git version 2.16.3" is displayed. The entire command line area is highlighted with a red rectangle.

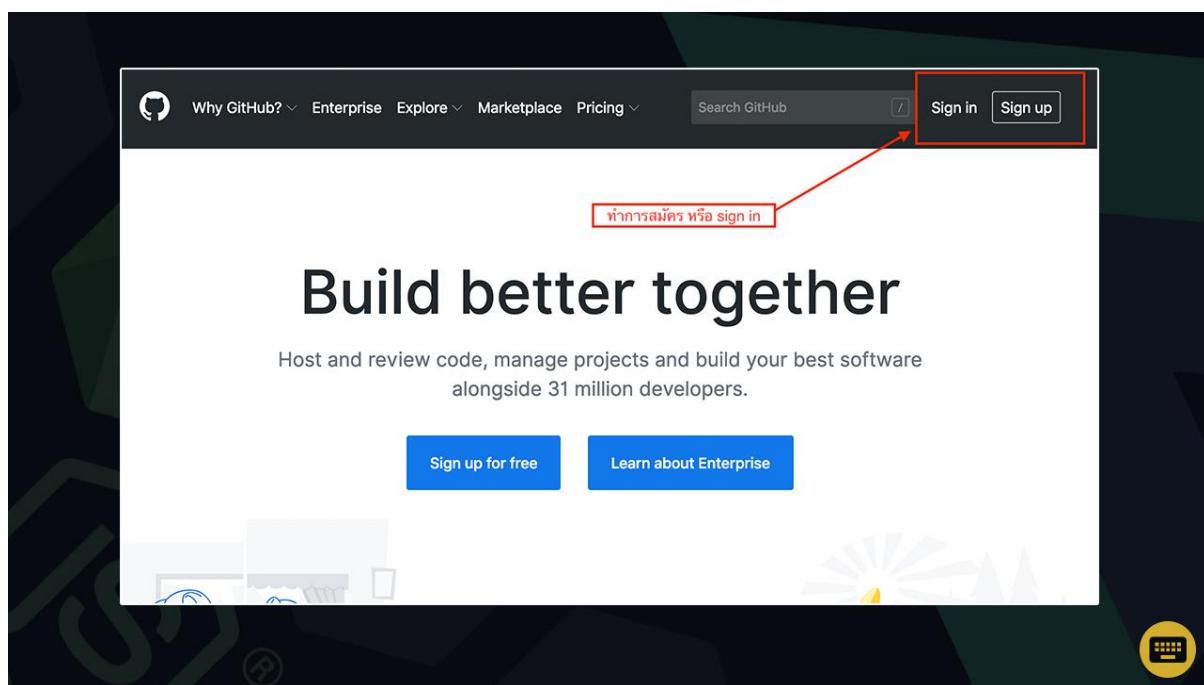
เพื่อทำการตรวจสอบ version โปรแกรม git ของเรา หากมีโปรแกรม git ในเครื่องเราแล้ว จะทำการแสดง version ของโปรแกรม git ของเราขึ้นมาดังรูป

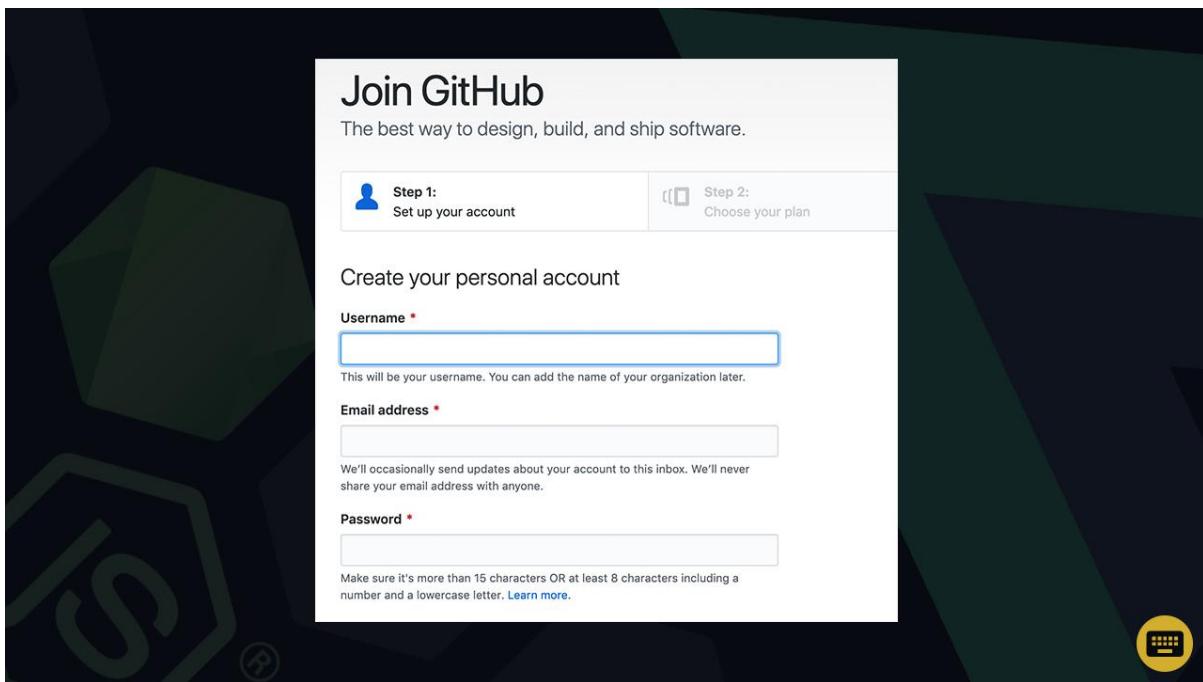
หากเราไม่มีโปรแกรม git ในเครื่องเรา สามารถทำการ download ได้จาก <https://git-scm.com/> จากนั้นทำการ download และติดตั้ง ตาม OS ที่เราใช้งาน ถ้าระบบปฏิบัติการของเราเป็น Windows OS ก็เลือก windows version หากเราใช้ macOS ก็เลือก version สำหรับ macOS ครับ ส่วนนี้ติดตั้งง่ายมาก ผ่านช่องทางด้านล่าง

สร้าง project บน github.com

เราเริ่มต้นในหัวข้อนี้จาก github.com สำหรับคนที่ยังไม่มีบัญชี github.com สามารถทำการสมัครกันได้ฟรีเลย โดยการสมัครนั้นก็แสนง่าย ซึ่งผู้ใช้ไม่อธิบายส่วนนี้นะครับ แต่ถึงอย่างไรหากติดขัด ปัญหาตรงไหนสามารถสอบถามในกลุ่ม Facebook ถามตอบของเราได้เลยครับ

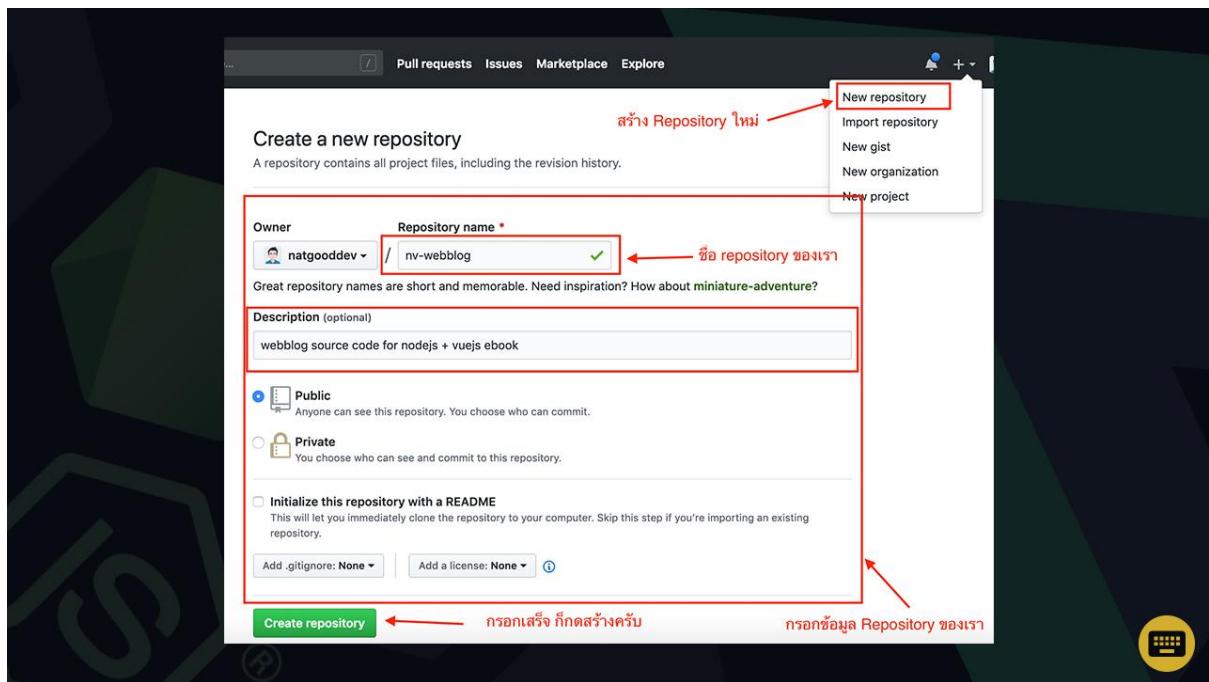
เมื่อทำการสมัครเสร็จแล้ว สิ่งแรกที่เราต้องทำ คือ การ login เข้าสู่ระบบ github.com ของเรา





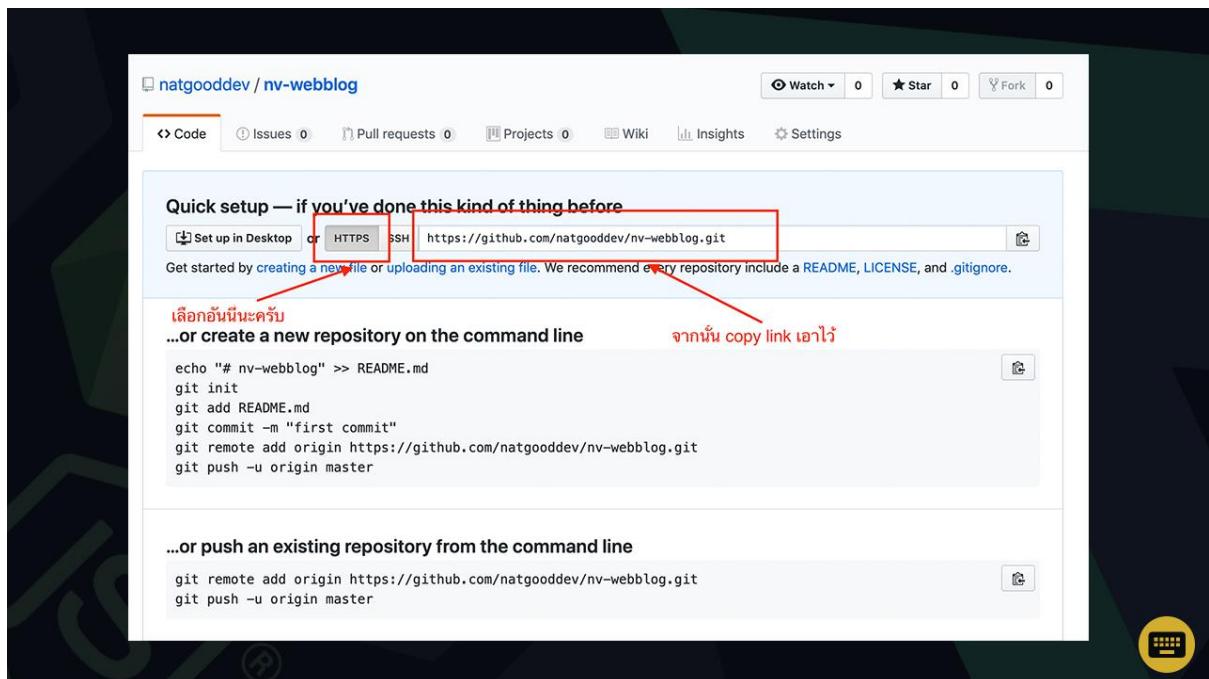
p36 ทำการสมัคร github.com

จากนั้นทำการสร้าง repository หรือ ที่เก็บ code และไฟล์ต่าง ๆ ของเรามา ผม ตั้งชื่อ nv-webblog ซึ่งมาจาก node vue web blog นั้นเอง หรือ จะใช้ชื่อโปรเจค อื่น ก็ไม่ว่ากันนะครับ เพียงแต่ต้องจำชื่อและรูปแบบให้ดีในการเขียน code เพราะ อาจจะผิดเรื่อง dependency หรือ path ต่างๆ ได้ นั่นคือเวลาผມใช้ nv-webblog ในโปรเจคของผม ท่านต้องเปลี่ยน dependency ตามด้วยเช่นกัน



ทำการ Clone โปรเจคมาที่เครื่องคอม ของเรา

เมื่อเราสร้าง Reposite หรือโปรเจคเสร็จของเราแล้ว เราจะทำการเลือกไปที่โปรเจคที่สร้างไว้ จากนั้นทำการ clone หรือ ทำการ copy โปรเจค (Repository) โดยทำการคัดลอก link git ของเราไว้ก่อน เดียวเราจะไป clone โปรเจคลงในเครื่องของเรา กันครับ



จากนั้น เปิด Terminal หากไอค์ mac OS หรือ Command Windows หากใช้ Windows OS ทำการเข้าไปที่ folder ที่เราจะใช้ เก็บโปรเจคของเรา โดยโฟล์เดอร์ หลักของ项目 เช่น d:/NodeRootBook จากนั้นทำการพิมพ์คำสั่งดังนี้

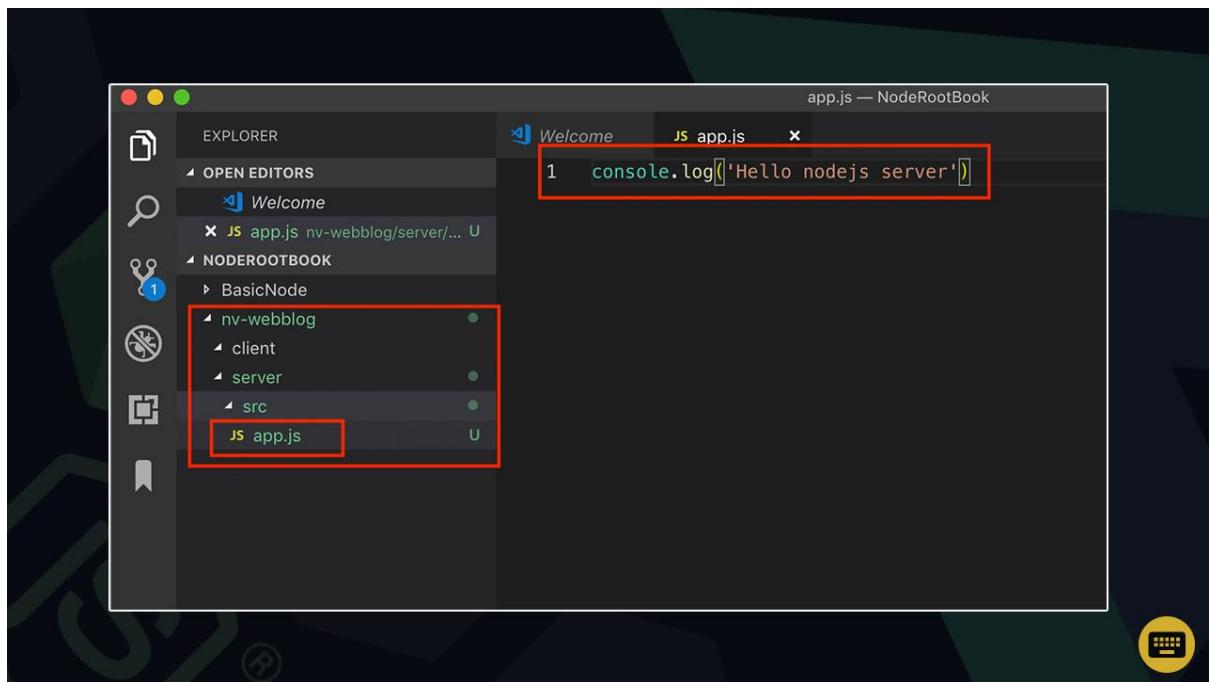
```
[192:basicnode kornchayapon$ git --version
git version 2.16.3
[192:basicnode kornchayapon$ ls
chap2      users.js
[192:basicnode kornchayapon$ cd ..
[192:noderootbook kornchayapon$ ls
BasicNode
[192:noderootbook kornchayapon$ git clone https://github.com/natgooddev/nv-webblog.git
Cloning into 'nv-webblog'...
Warning: You appear to have cloned an empty repository.
[192:noderootbook kornchayapon$ ls
BasicNode    nv-webblog
[192:noderootbook kornchayapon$ ]
```

จากนั้นกด Enter โปรแกรม git ของเราจะทำการ clone หรือคัดลอกโฟล์เดอร์ที่สร้างขึ้นไว้บน github.com มาไว้ที่เครื่องของเราโดยโฟล์เดอร์ที่ถูกสร้างขึ้นจากโปรแกรม git ในเครื่องของจะเป็นชื่อเดียวกับโปรเจคที่สร้างขึ้นไว้บน github.com

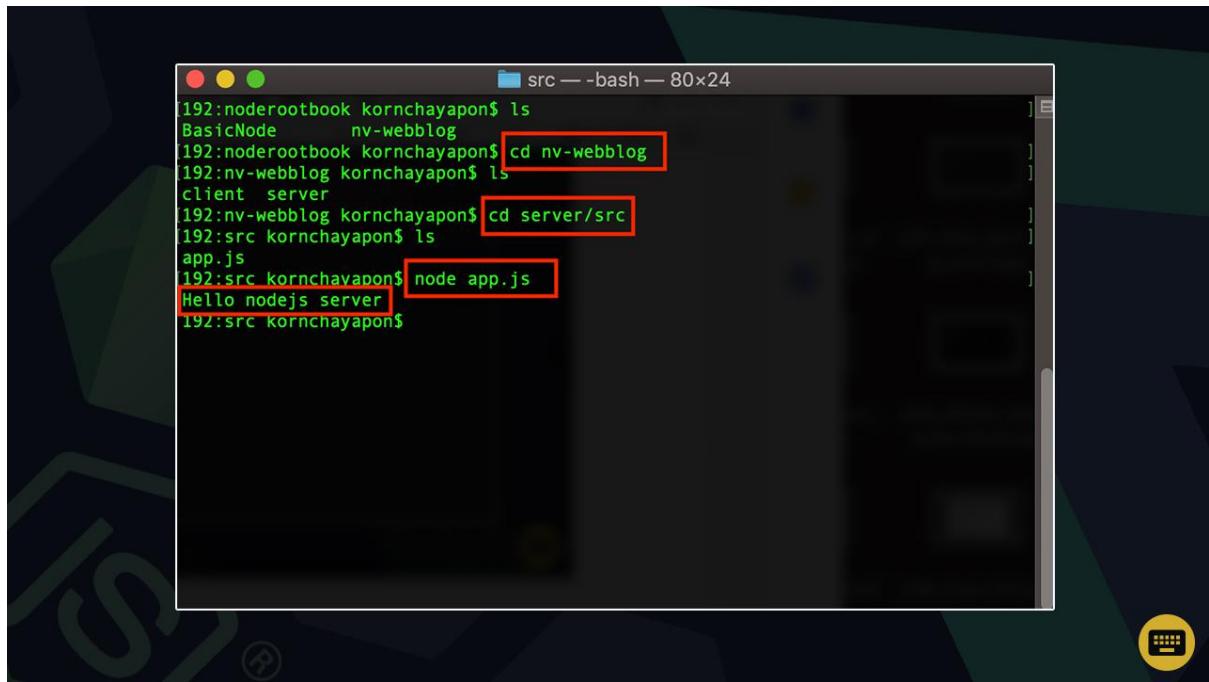
หลังจากการ clone โปรเจค มาไว้ในเครื่องของเราแล้ว เราจะทำการทดสอบเพิ่มไฟล์ในโฟล์เดอร์และ code โปรแกรมในเครื่องของเรา และทำการ upload ขึ้น github.com กัน

เปิด VS Code ของเราขึ้นมา จากนั้นทำการสร้างเราจะทำการเตรียมโครงสร้างโปรเจคของเรา โดยสร้างโฟล์เดอร์ client และ server ขึ้นมา และทำการสร้างโฟล์เดอร์ src ไว้ในโฟล์เดอร์ server เพื่อทำการเก็บ code โปรแกรมที่เราเขียนขึ้นนั่นเอง

ทำการสร้างไฟล์ app.js ขึ้น เพื่อทำการเขียน code โปรแกรม เปื้องต้นของเรา ทำการเขียน code โปรแกรมของเราดังนี้



ทำการรัน server/src/app.js ของเรา จะได้ผลลัพธ์ดังนี้



ทำการ upload โปรแกรมของเรา ขึ้น server github.com ด้วย commit และ push

เราเปิด Terminal ของเราขึ้นมา ปกติเวลาผมทำงานจะเปิดทิ้งไว้ตลอดนะครับ ไม่ต้องปิด ๆ เปิด ๆ ครับ

ไปที่ root folder ของเราที่ทำการเก็บ code project ของเรา สังเกตว่า root folder ของเรา ก็คือชื่อโปรเจคที่เราตั้งขึ้นบน github.com นั่นเอง จากนั้นทำการใส่คำสั่งดังภาพ เพื่อ upload version ของเราขึ้น github.com

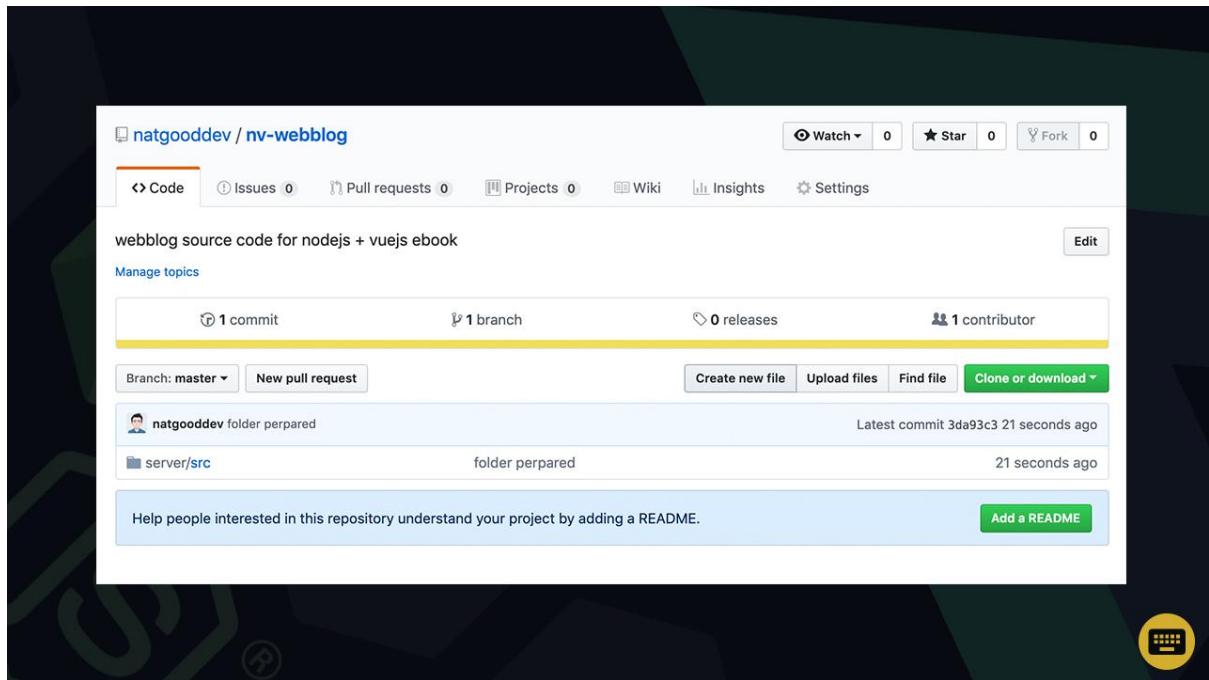
The terminal window shows the following command sequence:

```
[192:src kornchayapon$ cd ..  
[192:server kornchayapon$ cd ..  
[192:nv-webblog kornchayapon$ ls  
client_server  
[192:nv-webblog kornchayapon$ git add --all  
[192:nv-webblog kornchayapon$ git commit -m 'folder perpared'  
[master (root-commit) 3da93c3] folder perpared  
 1 file changed, 1 insertion(+)  
  create mode 100644 server/src/app.js  
[192:nv-webblog kornchayapon$ git push  
Counting objects: 5, done.  
Writing objects: 100% (5/5), 326 bytes | 326.00 KiB/s, done.  
Total 5 (delta 0), reused 0 (delta 0)  
To https://github.com/natgoodee/nv-webblog.git  
 * [new branch] master -> master  
192:nv-webblog kornchayapon$
```

Annotations in the screenshot:

- ต้องอยู่ใน folder นี้ (Must be in this folder)
- บอกว่าเอาทุกไฟล์เดียวจะ upload (Upload all files at once)
- กำหนดรายละเอียดของ version ที่เราจะเอาขึ้น (Specify the details of the version we will upload)
- ส่งขึ้นไปเลย (Send it up)

เข้าไปดูผลงานที่เราทำที่ github.com ตามโปรเจคของเรา คือ 'nv-webblog' จะเห็นการเปลี่ยนแปลง เอ๊ะ แต่ทำไมไม่มี folder client ของเราล่ะ เหตุผล คือ repository จะทำการเก็บโฟล์เดอร์เฉพาะโฟล์เดอร์ที่มีไฟล์อยู่ภายในนั้นเองครับ



*** หลังจากบันทึก เราจะไม่อธิบาย เกี่ยวกับ github หรือทำให้ดูแล้วนะครับ เพียงแต่ เตือนเรื่องการ upload ขึ้น github.com เท่านั้น ***

สรุปคำสั่ง git ที่เราจำเป็นต้องใช้ *** ทำ link เพื่อไปดูรายละเอียดคำสั่งต่าง ๆ

```
git clone {link-git-ของเรา}
git add --all
git commit -m 'รายละเอียดใน version นี้ เพิ่มอะไร แก้อะไร ใส่ไว้ครับ เป็น note ของ version
ของเรา'
git push เป็นคำสั่ง upload ขึ้น github.com
```

บทที่ 4 เริ่มต้นทำ Web Server ด้วย Express และ Body Parser กันเถอะ

ทำความรู้จัก nodemon

จากการเขียนโปรแกรมที่ผ่านมาเวลาที่เราจะทำการรันโปรแกรม nodejs ของเรา นั้นเราต้องทำการรันคำสั่ง node app.js ของเราทุกครั้งเวลาที่เรา แก้ไขโปรแกรม และทำการ save โปรแกรมที่เราเขียนขึ้น

จะมีหนทางหนึ่งในการแก้ไขปัญหาเหล่านั้น nodemon เป็นโปรแกรมที่คอย update update โปรแกรม หรือ auto restart nodejs ของเราทันทีเมื่อเราทำการกด save ในโปรแกรมของเรา

ทำการติดตั้ง nodemon ด้วยคำสั่ง ดังนี้

```
npm install -g nodemon
```

เมื่อเราทำการรันโปรแกรมครั้งต่อไป เราสามารถเรียกใช้คำสั่งจาก nodemon ได้โดย เช่น

```
nodemon app.js
```

ทุกครั้งที่เราทำการ save ไฟล์ที่เกี่ยวข้องกับ app.js เรียก หรือ ตัว app.js เอง nodemon จะทำการ rerun โปรแกรมของเราให้ชึ้นทำให้เราไม่เสียเวลาในการทดสอบโปรแกรม

รายละเอียดเพิ่มเติมเกี่ยวกับ nodemon สามารถได้ที่นี่ครับ

<https://nodemon.io/>

มาทำความรู้จัก npm กัน

โปรแกรม npm หรือ node package manager จะถูกติดตั้งมาเมื่อเราทำการติดตั้ง nodejs ในเครื่องของเรา ตามที่เคยกล่าวไปแล้วเบื้องต้นเกี่ยวกับ nodejs เบื้องต้น

npm ถ้าเปลี่ยนความหมายเลย คือ ตัวจัดการ node package นั่นเอง

แล้ว node package คืออะไรล่ะ คือ package ที่ถูกสร้างจาก node ทำไว้ให้สามารถนำกลับมาใช้งานใหม่ได้ทั้งตัวผู้เขียนเอง หรือ upload ขึ้น npm website ไว้คนอื่นใช้ด้วยครับ โดยเราสามารถใช้งาน package ที่คนอื่นสร้างขึ้นไว้ได้ผ่าน npm นั่นเอง ตัวอย่างเช่น express ที่เราจะใช้ทำ webserver กันด้วยครับ

เริ่มต้น ทำ webserver ด้วย Expressjs

มาถึงตอนนี้เราจะทำ webserver ด้วย nodejs กัน อย่างแรกที่เราต้องทำ คือ สร้าง nodejs package ของเรากัน การสร้าง node package ฟังอาจไม่คุ้นหู จริงๆ มันคือ การสร้าง project ในส่วน Sever ของเรานั่นเองครับ โดยพิมพ์คำสั่งดังภาพไฟล์เดอร์ server ของเรา จากนั้นจะให้ใส่ข้อมูลเกี่ยวกับโปรเจค หรือ nodejs package ของเราไปจนครบ npm ของเราจะสร้างไฟล์ package.json ให้อัตโนมัติ ดังภาพ

เข้าไปใน folder server

```
192:nv-webblog kornchayapon$ ls
client server
192:nv-webblog kornchayapon$ cd server
192:server kornchayapon$ npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See 'npm help json' for definitive documentation on these fields
and exactly what they do.

Press ^C at any time to quit.
package name: (server) nv-webblog-server ชื่อ โปรเจกของเรา
version: (1.0.0)
description: nodejs vuejs weblog server (back end) รายละเอียด
entry point: (index.js) src/app.js
test command:
git repository: ← ไม่ใส่ enter ผ่านได้
keywords:
author: chayapon chandra ชื่อคนเขียน
license: (ISC)
About to write to /Users/kornchayapon/NodeRootBook/nv-webblog/server/package.json:

{
  "name": "nv-webblog-server",
  "version": "1.0.0",
  "description": "nodejs vuejs weblog server (back end)",
  "main": "src/app.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" & exit 1"
  },
  "author": "chayapon chandra",
  "license": "ISC"
}

Is this ok? (yes) yes
192:server kornchayapon$
```

รายละเอียด โปรเจกที่ เรากำลัง

EXPLORER JS app.js package.json

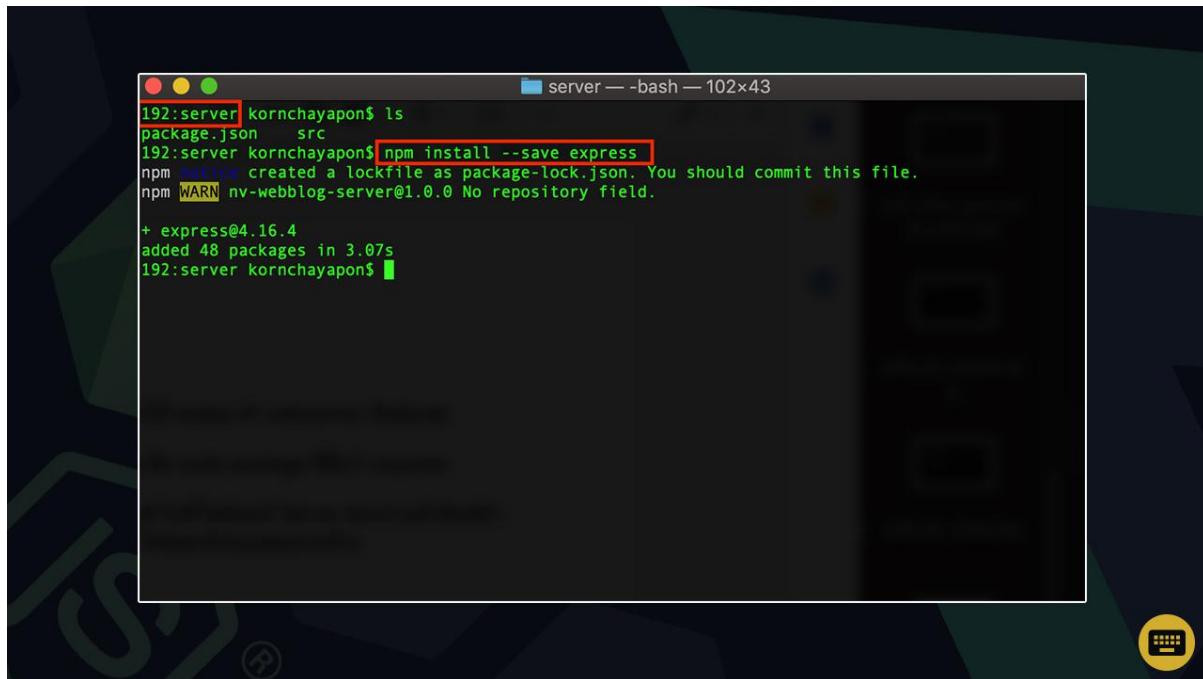
```
{
  "name": "nv-webblog-server",
  "version": "1.0.0",
  "description": "nodejs vuejs weblog server (back end)",
  "main": "src/app.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" & exit 1"
  },
  "author": "chayapon chandra",
  "license": "ISC"
}
```

ติดตั้งและใช้งาน express

เมื่อเราสร้าง node package ของเราเสร็จแล้วเราจะทำการใช้ nodejs ทำ webserver กันต่อเลย

โดย node package ที่เราจะใช้ทำ webserver ของเราคือ node package ที่ชื่อว่า express

ทำการติดตั้ง express ผ่าน npm โดยเปิด Terminal และเข้าไปที่โฟล์เดอร์ Server ของเราแล้วพิมพ์คำสั่งดังนี้ (จำไว้ว่าถ้าเราทำอะไรเกี่ยวกับ server ต้องเข้าไป โฟล์เดอร์ก่อนเสมอนะครับ)



```
192:server kornchayapon$ ls
package.json  src
192:server kornchayapon$ npm install --save express
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN nv-webblog-server@1.0.0 No repository field.

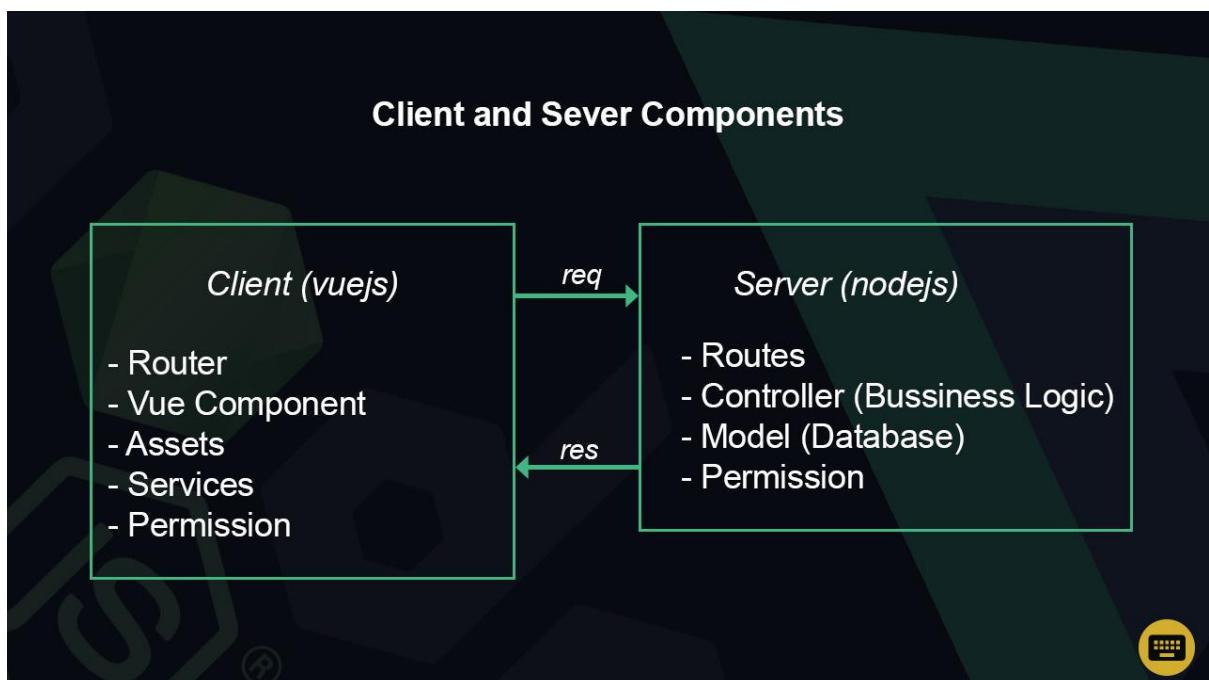
+ express@4.16.4
added 48 packages in 3.07s
192:server kornchayapon$
```

เมื่อทำการติดตั้งเสร็จ เรา마다ูที่ package.json ของเราจะพบว่า มีการเพิ่มชื่อ package ที่เราใช้เข้ามาในข้อมูล package ของเราด้วยนั่นเอง เราเข้าไปโฟล์เดอร์ node module ใน server ของเราจะมี module ที่ชื่อว่า express ปรากฏอยู่ เป็นอันว่าเราได้ติดตั้งเสร็จเรียบร้อยพร้อมใช้งาน

```
package.json — NodeRootBook
{
  "name": "nv-webblog-server",
  "version": "1.0.0",
  "description": "nodejs vuejs weblog server (back end)",
  "main": "src/app.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "author": "chayapon chandra",
  "license": "ISC",
  "dependencies": {
    "express": "^4.16.4"
  }
}
```

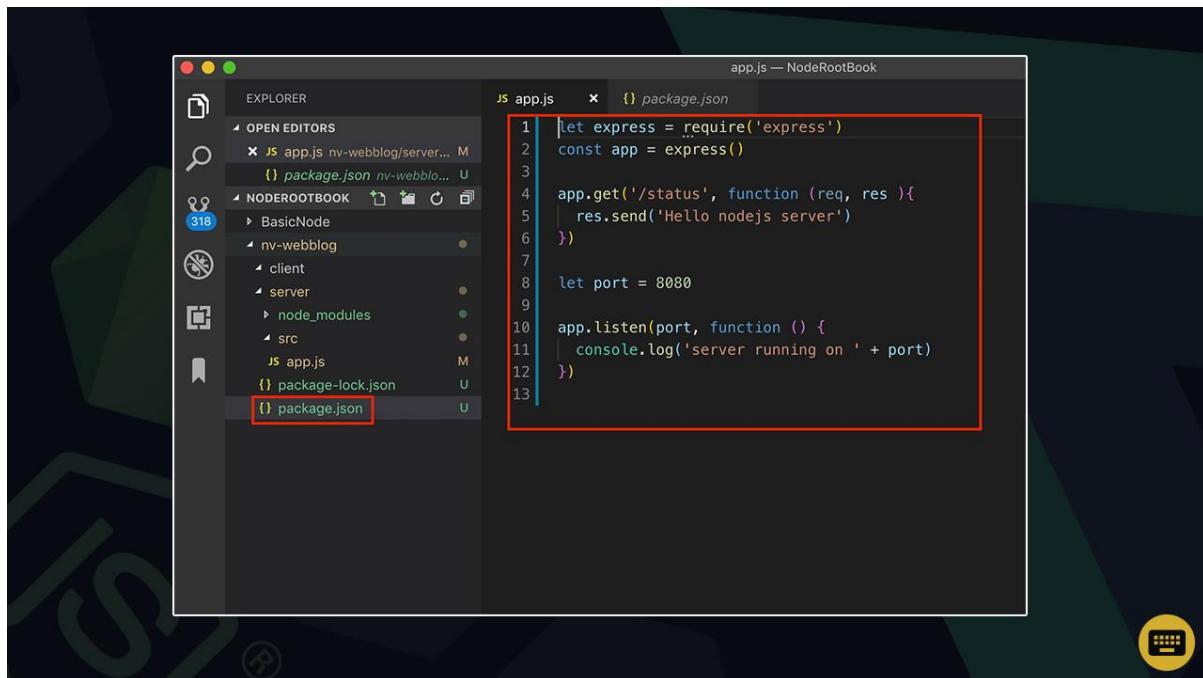
หลังจากเราติดตั้งเครื่องมือทำ webserver ของเราด้วย nodejs

จากบทที่ผ่านๆ มา ทำให้เรามีพื้นฐานเพียงพอ ที่จะเริ่มต้นทำเว็บไซต์ด้วย nodejs ได้แล้ว โดยระบบที่เรารอออกแบบไว้จะประกอบด้วย ส่วนที่เป็น Server ที่เราทำด้วย nodejs และส่วนที่เป็น client ที่เราทำด้วย vuejs ดังแผนผังที่แสดงไว้ด้านล่าง



มาเริ่มกันเลยครับ เปิด vscode ของเราขึ้นมา

จากนั้นเปิด nv-webblog/server/src/app.js ขึ้นมา ในโปรเจคของเราขึ้นมา แล้วทำการเขียน code ตามผมได้เลย



The screenshot shows the VS Code interface with the following details:

- Explorer View:** Shows the project structure with files like app.js, package.json, and node_modules.
- Code Editor:** The file app.js is open, containing the following code:

```
1 let express = require('express')
2 const app = express()
3
4 app.get('/status', function (req, res) {
5   res.send('Hello nodejs server')
6 }
7
8 let port = 8080
9
10 app.listen(port, function () {
11   console.log('server running on ' + port)
12 })
13
```
- Terminal:** A yellow terminal icon is visible in the bottom right corner.

* โปรแกรม web server ของเรา * แก้ port เป็น port = 8081 นะครับ

code ข้างต้นลองไป ดูครับ ไม่ยากเลย เราติดตั้ง express แล้ว เราจะสามารถเรียกใช้ โดย require เข้ามาได้เลย หรือ การ include ในภาษาซี แล้วกำหนด ตัวแบบคงที่ ชื่อ app เป็นตัวอังกฤษเพื่อใช้งานคำสั่งอื่น ต่อไป

```
res.send('Hello nodejs server')
```

คือสิ่งข้อความ string "Hello nodejs server" ออกไปทาง HTTP หรือ ซึ่งสามารถดูได้ผ่าน Web Browser ของเราเอง

แต่ทั้งหมดที่ว่านั้นจะไม่ทำงาน หาก Webserver ของเราไม่ทำงาน โดยเราจะสั่ง Webserver ของเราทำงานได้นั้น เราต้องกำหนด port ขึ้นมาเสียก่อน แล้วให้ app ของเราอยู่อีียงหุฟังตลอดเวลา ว่าเค้าเรียกมาทาง port ที่เรากำหนดไว้มั้ย เรียก

มาแบบไหน ที่อยู่ตรงกับที่เราสร้างมั้ย ก้าตรงก็ให้ตอบ ตามที่กำหนดไว้นั้นเอง นั่นคือ code ส่วนนี้ครับ

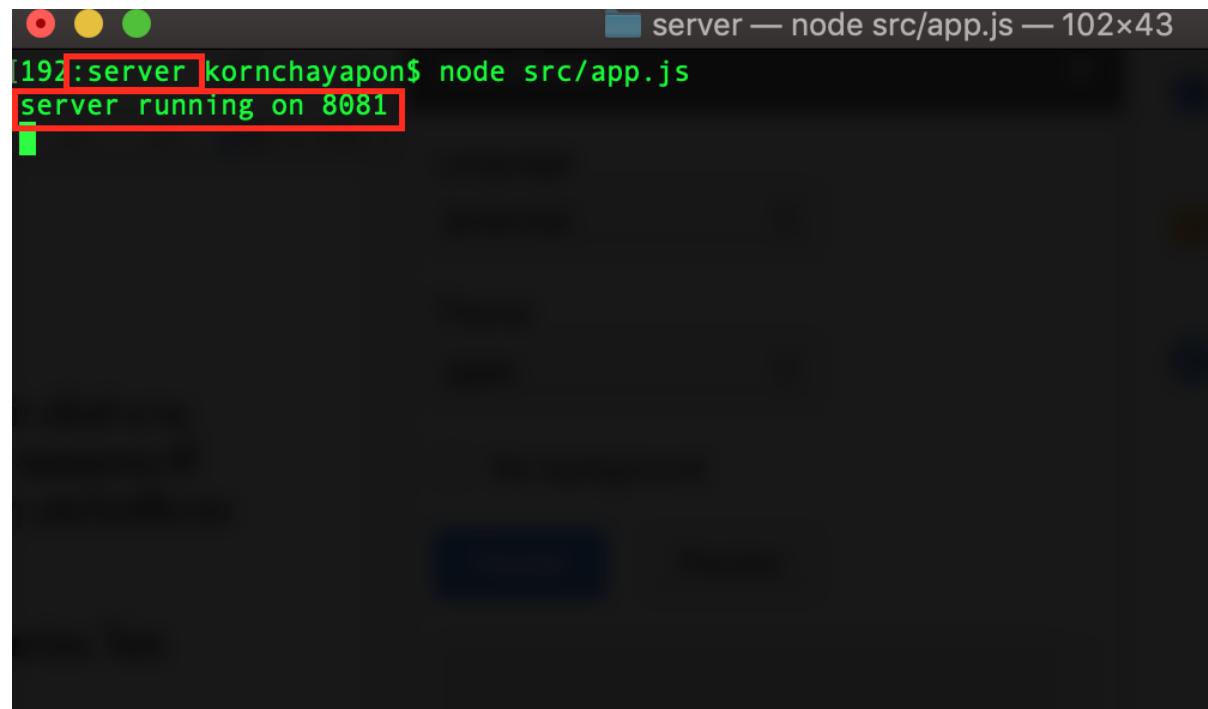
```
let port = 8081
app.listen(port, function () {
  console.log('server running on ' + port)
})
```

แต่เดี๋ยวก่อน ตอนที่เราสั่งให้ app ของเรารออยເອີ້ນຫຼັງພິບນີ້ ເວົ້າໃສ່ໃໝ່ມັນແສດງຜລດ້ວຍວ່າ ມັນທຳການໂຄງການຢູ່ຕື່ມີ້ຍ ທຳການທີ່ພວັດທະນາ ໂດຍບໍ່ມີຄວາມນີ້ເປັນຄຳສັ່ງທີ່ມາຈາກ console.log() ມັນຈະແສດງຜລອອກມາທີ່ Terminal ທີ່ເຮັດວຽກຄຳສັ່ງ node ນະຄົນ ໄນໃຊ້ຕອບກັບຜ່ານ http ແບນ res.send() ຄວຍ ຈະເລີ່ມຕົ້ນໄປເຖິງຈະຈໍາໄດ້ເອັນຄົນ

ເມື່ອເຮັດວຽກ code ໂປຣແກຣມຂອງເວົ້າເສົ້າຈະແລ້ວ ມັນຄົງທຳການເອງໄມ່ໄດ້ ເວົ້າຕ້ອງໄປສັ່ງຮັນ ເສີຍກ່ອນ ໂດຍເຂົ້າໄປທີ່ຕໍ່ແນ່ງທີ່ເກີນໂປຣເຈັດຂອງເວົ້າກ່ອນເສົ້າ ເຊັ່ນ

d:/nv-webblog/server

ຈາກນີ້ທີ່ການຮັນ ຄຳສັ່ງ



The screenshot shows a terminal window titled "server — node src/app.js — 102x43". The command "node src/app.js" was run, and the output "server running on 8081" is displayed. The line "server running on 8081" is highlighted with a red rectangle.

```
[192:server kornchayapon$ node src/app.js
server running on 8081]
```

```
node src/app.js
```

ผลการรันโปรแกรมที่หน้า console ของเราจะเป็นดังนี้

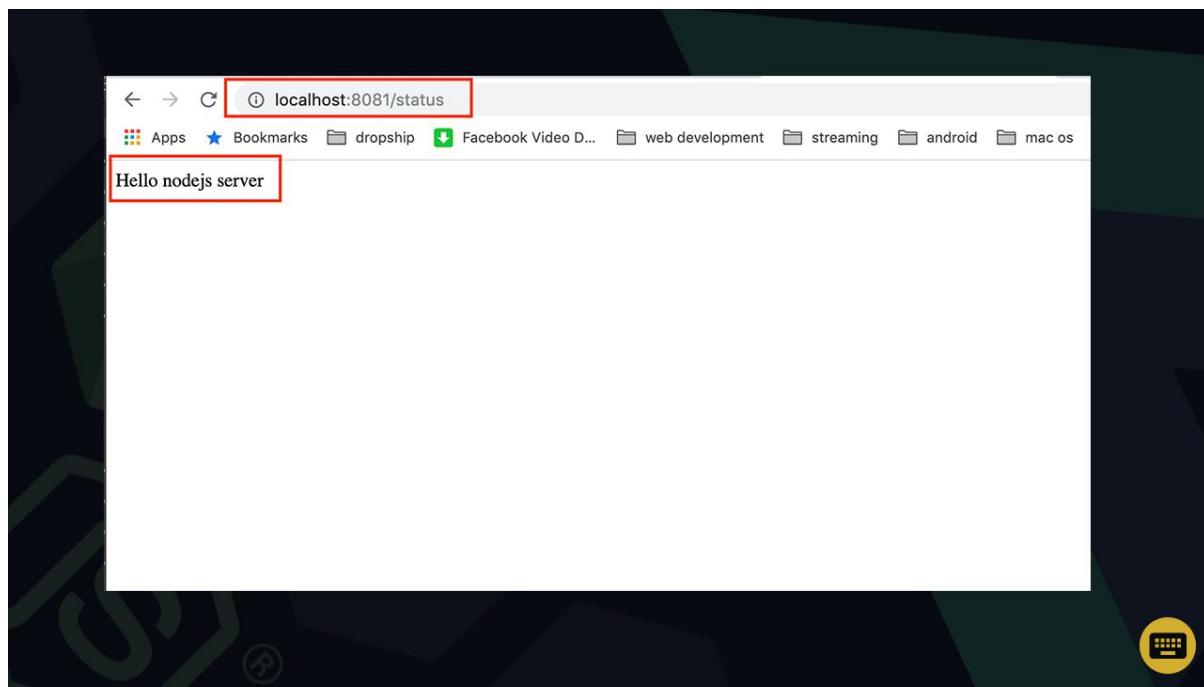
```
server running on 8081
```

โปรแกรมของเราจะตอบกลับมาตามที่กำหนดไว้ ในส่วนของ Terminal จากนั้นเรา จะทำการเรียกไปที่ Webserver ของเรา โดยทำการเรียกไปที่

โดยเรียกไปที่ localhost ตามหมายเลข port ที่เรากำหนดไว้นั้นเอง ผลลัพธ์ที่ได้ เว็บเซิร์ฟเวอร์ของเราจะตอบกลับมาดังนี้ครับ

ทำการเปิด browser เช่น chrome ของเราแล้วทำการใส่ URL ตามนี้นะครับ

```
http://localhost:8081/status
```



ตอนนี้เราได้ Webserver ของเราแล้ว นี่เป็นตัวอย่างการทำเว็บเซิร์ฟเวอร์ของเราแบบง่าย ๆ ซึ่งเวลาทำงาน เราค่อยทำจากง่าย ๆ และค่อย ๆ เพิ่มเข้าไปเมื่อไห้ขึ้นบันไดทีละขั้น พอเราชำนาญแล้วมันจะเร็วไปเอง

*** มาถึงตอนนี้ รูปเราจะน้อยลงแล้วนะครับ เพราะเริ่มเก่งกันแล้ว และส่วนของการ coding จะเป็น *code format* แทน เพื่อให้ทุกคนสามารถ copy ไปวางได้ จะได้ทดสอบกันได้ง่ายกัน แต่อย่า copy อย่างเดียวนะครับ ลองเขียนเองก่อน ถ้าติดแล้ว ค่อยทดสอบโดย copy code ไปวางครับ ***

มาเริ่มทำ RESTful API แบบ get กันครับ

ในบทนี้เขียน RESTful API แบบง่าย ๆ กันนะครับ ผู้เดยอธิบายเรื่อง RESTful ไปแล้วนะครับ ว่ามันคืออะไร มาถึงตอนนี้ผู้เดยมาเขียนมันกันครับ

API ส่วนแรกที่ผู้เดยจะทำก่อน คือ ส่วนของ "ระบบสมาชิก" เพราะถ้าเราทำส่วนนี้ได้ เราจะทำได้เกือบหมดทั้ง Website โดย พังก์ชันของเว็บไซต์ที่ผู้เดยต้องการเบื้องต้น คือ

1. สร้างผู้ใช้
2. แก้ไขผู้ใช้
3. ลบผู้ใช้
4. หยุดสถานะการใช้งาน
5. ให้สถานะปกติ

โดยในหัวข้อนี้ เป็นเพียงสร้าง url เพื่อวิ่งไปแสดงผลแต่ละส่วนได้ ถ้าเราจำเรื่องสำนักพิมพ์ ในตอนต้นได้ นั่นคือ เราจะสร้าง URL เพื่อวิ่งແນกต่าง ๆ หรือ Controller ของเรานั่นเอง เพื่อบอกว่าเราส่งข้อมูลไปที่ແນกต่าง ๆ ในໂຮງພິມພົບ (Back End) ของเราได้อย่างถูกต้อง โดยเราจะลงรายละเอียดในส่วนอื่นในบทต่อไปนะครับ

ก่อนหน้านี้เราได้สร้าง URL ไว้หนึ่ง URL คือ <http://localhost:8080/status> ถ้าเราเปรียบเทียบกับสำนักพิมพ์ของเรา ส่วนนี้ก็คือการส่งข้อมูลไปที่ แผนกเชิญสถานะของเรา พอเราส่งข้อมูลไป เค้าก็ตอบกลับมาเป็น

```
Hello nodejs server
```

นั้นเองครับ ตอนนี้เราจะทำการเพิ่มตำแหน่ง แผนกอื่น ๆ บน Back End หรือ โรงพิมพ์ของเรา กันครับ โดยผมกำหนดให้แผนกต่าง ๆ ของผมเป็นดังนี้ครับ

1. สร้างผู้ใช้, <http://localhost:8080/user>, method post
2. แก้ไขผู้ใช้, หยุดสถานะการใช้งาน, ให้สถานะปกติ,<http://localhost:8080/user/:userId>, method put
3. ลบผู้ใช้, <http://localhost:8080/user/:userId>, delete
4. ดูข้อมูลผู้ใช้, <http://localhost:8080/user/:userId>, method get
5. เรียกข้อมูลผู้ใช้ทั้งหมด, <http://localhost:8080/users>, method get

เมื่อเราวางแผนกต่าง ๆ ของและ URL ของแต่ละแผนกได้แล้ว เราจะมาเขียนโปรแกรมตามที่เรากำหนดให้ เว็บเซิร์ฟเวอร์ซองเราวิ่งตาม บริ ที่เราสร้างขึ้นกันโดยทำการแก้ไข app.js ใน src ของเราดังนี้ครับ

```
let express = require('express')

const app = express()

app.get('/status', function (req, res) {
  res.send('Hello nodejs server')
})

app.get('/hello/:person', function (req, res) {
  console.log('hello - ' + req.params.person)
  res.send('say hello with ' + req.params.person)
})
```

```

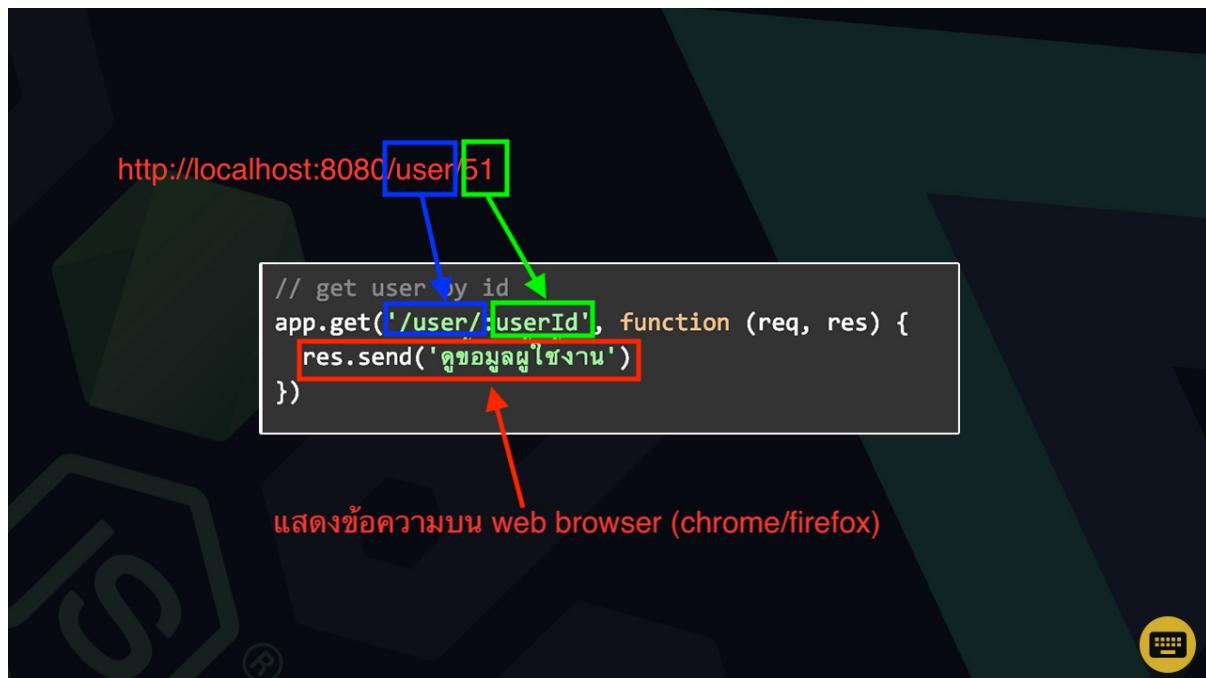
// get user by id
app.get('/user/:userId', function (req, res) {
  res.send('ดูข้อมูลผู้ใช้งาน')
})

// get all user
app.get('/users', function (req, res) {
  res.send('เรียกข้อมูลผู้ใช้งานทั้งหมด')
})

let port = 8081
app.listen(port, function () {
  console.log('server running on ' + port)
})

```

จาก code ที่เขียนข้างต้น ไม่ได้มีอะไรแตกต่างจาก '/status' มาก เพียงแต่ให้บริการตามเงื่อนไขของ method พ่วงมาด้วยเท่านั้น ในเบื้องต้นนี้ผมเพียงแต่ให้ make sure ว่าให้บริการถูกต้องตาม link และ method ของ http ที่เรียกได้ถูกต้องตามที่กำหนดไว้เท่านั้น



ภาพอธิบาย *format* ในการเขียน code เพื่อจัดการตำแหน่งของแทนก หรือ *url* ว่าแต่ส่วนของ code เราคืออะไรนะครับ ทำความเข้าใจคร คร่าว ๆ ไม่เข้าใจก็ไม่ต้องไปกังวลมากมาย เขียนเยอะ ๆ เดี๋ยวเข้าใจเองครับ

นี่จะครับ RESTful ของผมแบบง่าย ๆ เพื่อตอบรับบริการ ตาม method และ url ที่เรียกมาครับ

ในรูปแบบนี้จะเป็นการใช้ออกคำสั่งไปที่แท็ลเลฟอน (URL) แบบ get นะครับ ส่วนแบบ get คืออะไร ถ้าจำไม่ได้ ย้อนไปอ่านบทแรก ๆ นะครับ

ทำความรู้จักคนส่งไปรษณีย์ Postman

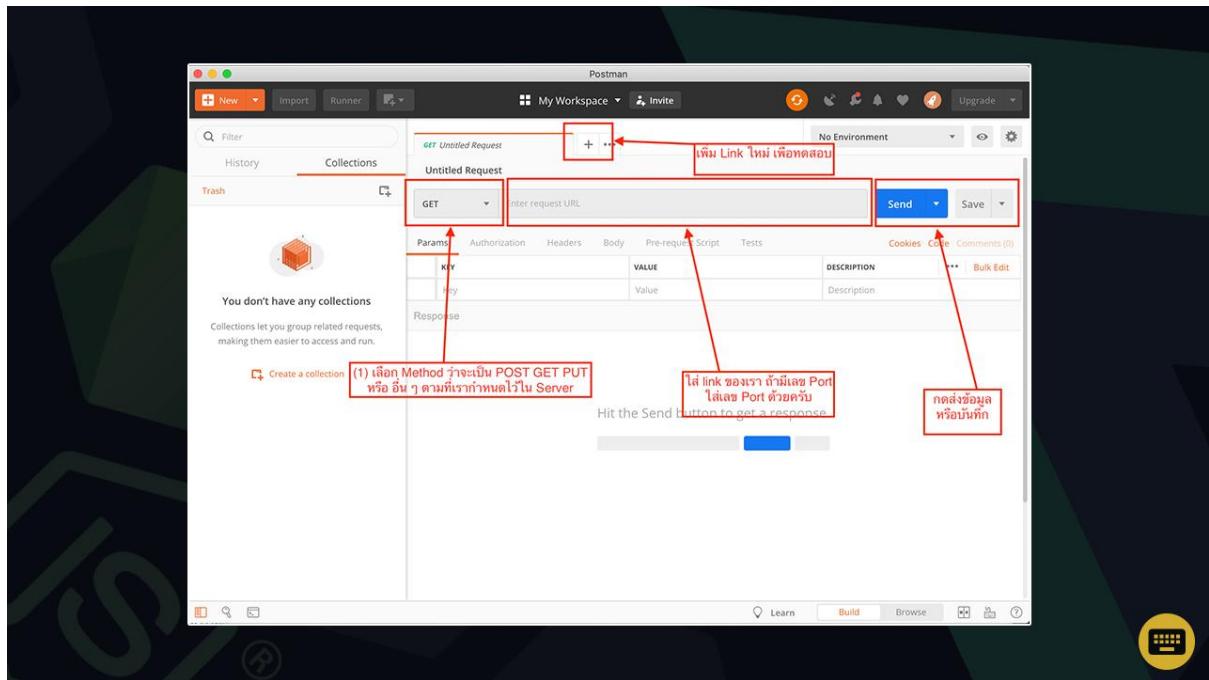
ในบทความนี้ผมจะมาแนะนำโปรแกรม Postman เอาไว้สำหรับ ทดสอบ API ที่เราเขียนโปรแกรมขึ้นกันครับ ก่อนหน้าที่ผมจะรู้จักโปรแกรมตัวนี้ การ Debug API ที่เขียนขึ้น นั้นยากและค่อนข้างเสียเวลาอย่างมาก ผมต้องมานั่งเขียนหน้าบ้าน เพื่อโพส JSON ไปที่ API Server ของเรา ยิ่งเป็นยุค PHP ยิ่งลำบากหนัก ต้องมานั่งเขียน Form เพื่อส่ง Form Data ไป โอ้ย!!! ชีวิตมันซ่างผุ้งยาก ลำบากเสียเหลือเกิน (แอบดราม่า)

พอพบ Postman ชีวิตดีขึ้นเยอะ ทดสอบ API ได้แบบง่าย ๆ รวมเร็ว รวมถึงสามารถ Post เป็น Form Data ได้ด้วย

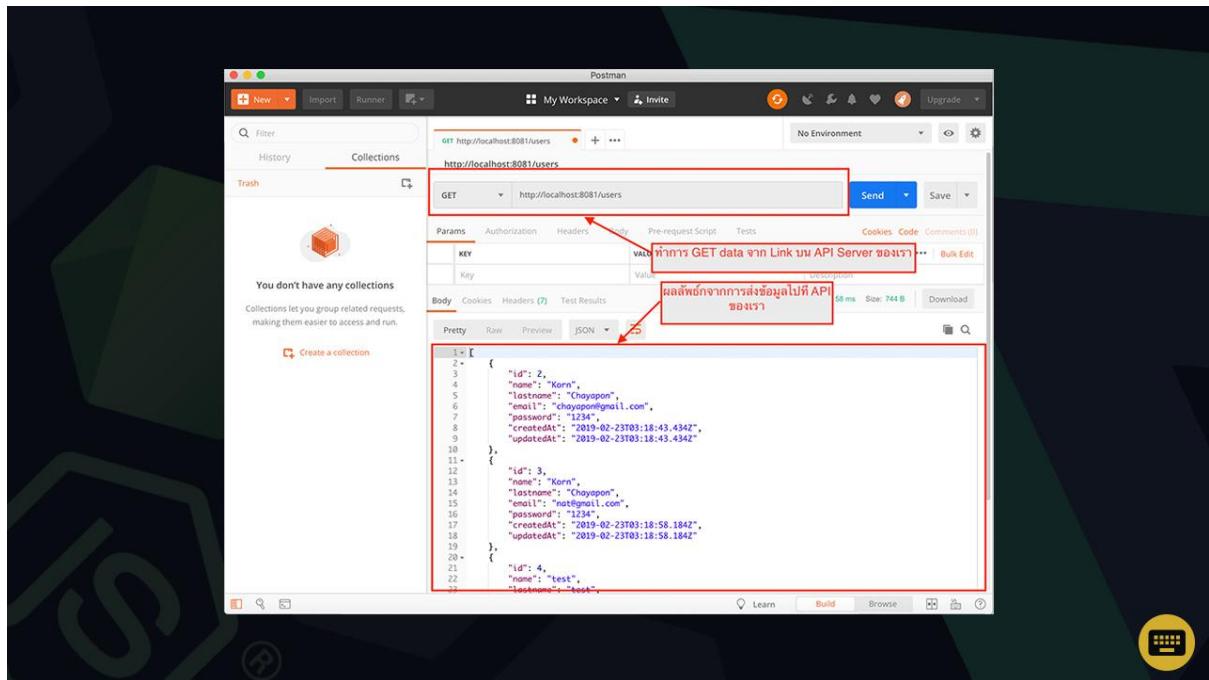
ก่อนอื่นเลยไปดาวน์โหลดกันได้ที่นี่เลยครับ

<https://www.getpostman.com/downloads/> เลือกตาม OS ที่ท่านใช้ ดาวโหลดเสร็จ ก็ทำการติดตั้งโปรแกรมกันเสร็จเรียบร้อย ไม่สอนแล้วนะครับ น่าจะทำได้กันทุกคนครับ

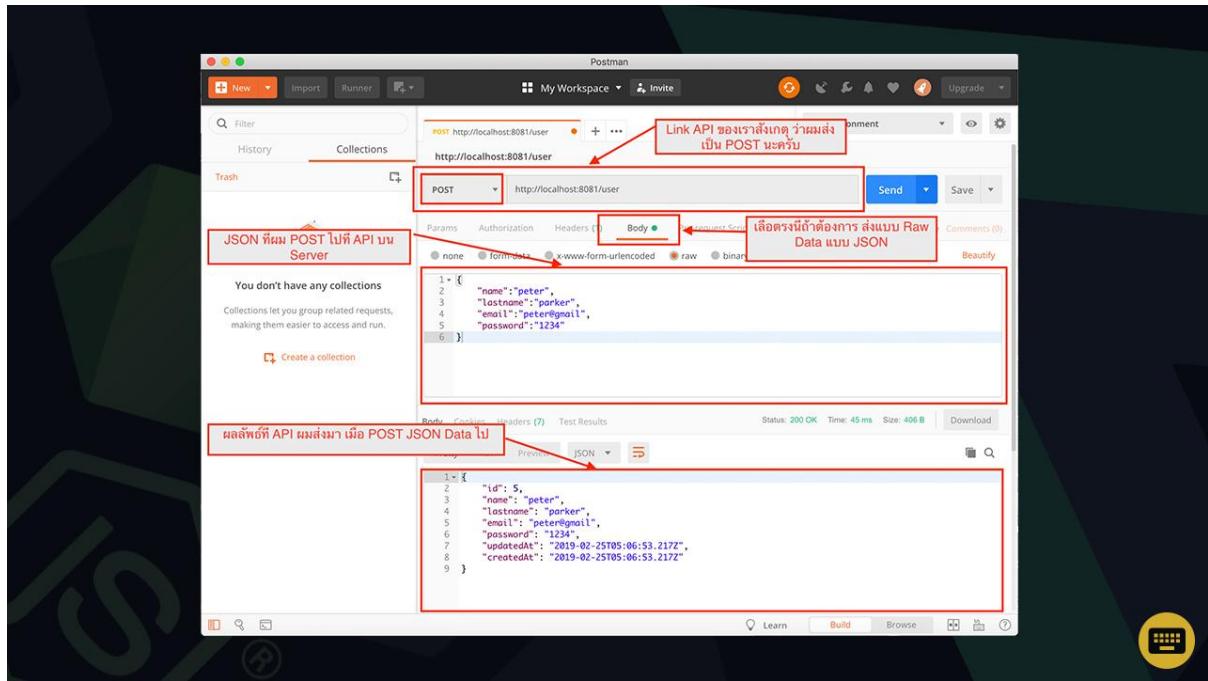
เปิดโปรแกรมขึ้นมาหน้าตาจะเป็นอย่างในรูป หรือ ถ้า Version เปลี่ยนไป เราก็อย่าไปสนใจหน้าตามาก ให้ดูโครงสร้าง และวิธีใช้งานที่ผมจะอธิบายจากรูปนะครับ



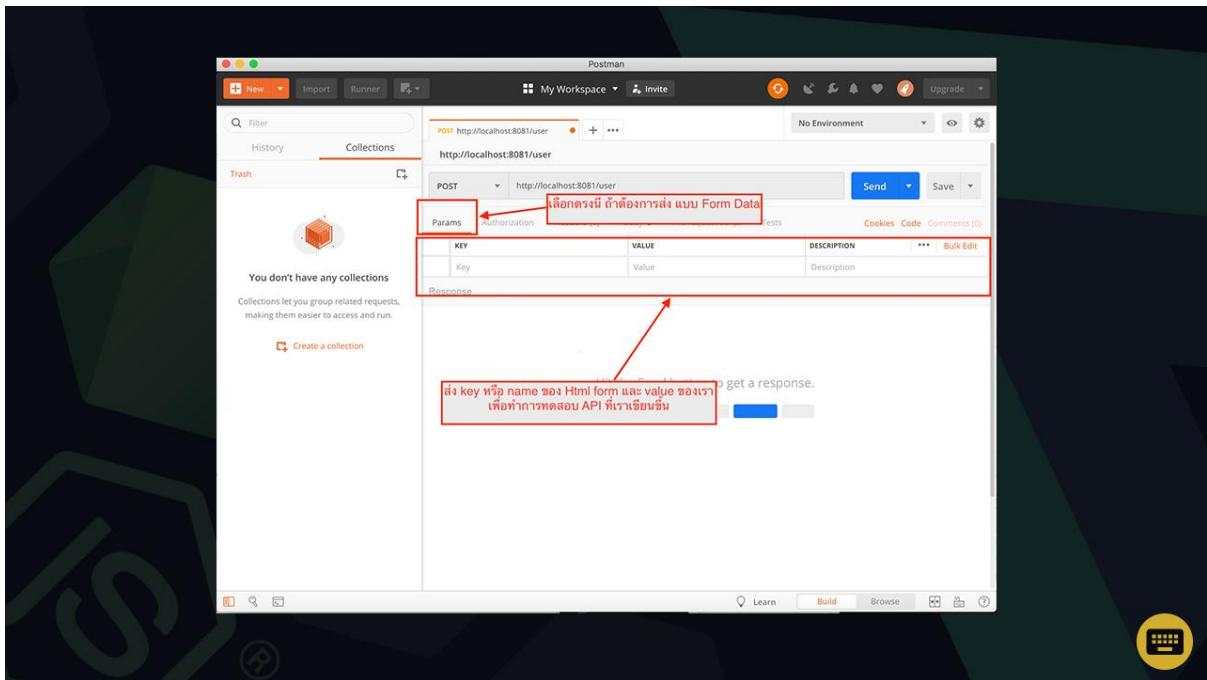
เริ่มต้นที่ (1) ทำการเลือก HTTP Method ที่เราต้องการจะทดสอบ จากนั้นทำการใส่ URL ของ API บน Server ที่เราต้องการจะส่งข้อมูล แล้วทำการกด SEND เพื่อส่งข้อมูลไปที่ Server ของเรา หรือทำการ Save เอาไว้ใช้ทดสอบอีกภายหลังก็ได้ครับ



จากภาพ ผู้ทำการดึงข้อมูลจาก Server แบบ Get HTTP Method โดยโปรแกรม POSTMAN จะทำการดึงข้อมูลมาแสดงให้เราดูได้อย่างสะดวกมาก



ในขั้นตอนนี้ ผู้ทำการส่งข้อมูล JSON ในแบบ POST HTTP Method ไปที่ Server ของผู้ โดย โปรแกรม API ที่ผู้เขียนไว้นี้ จะทำการสร้าง User ใหม่ ในระบบ แล้วทำการตอบกลับข้อมูลเดียวกันกลับมาเมื่อทำงานสำเร็จตามที่เขียน โปรแกรมไว้ ถ้าไม่สำเร็จหรือล้มเหลว API ของผู้จะตอบข้อความอีนกลับมา โดย การส่งข้อมูลแบบ JSON ไปที่ API นี้ เรายังต้องเลือกไปที่ Body และพิมพ์ JSON ของเราด้วยตัวเองนะครับ จากนั้น กด SEND Server ของผู้จะทำการตอบกลับมาดังภาพครับข้างบนครับ



ในกรณีที่เราต้องการส่งข้อมูลแบบ Form POST เรา ก็สามารถทำได้ เช่นกัน โดยเลือกไปที่ Params และทำการส่งข้อมูลไปได้เลย โดย key = name ของ FORM Html ของเรา และ Value คือเราจะส่งอะไรไปนั่นเอง

จริง ๆ แล้ว โปรแกรม POSTMAN นั้นทำอะไร ได้มากน้อย แต่สำหรับผู้ใช้งานเพียงเท่านี้นั้นเพียงพอแล้วสำหรับการเริ่มต้น Debug API บน Server ที่เราเขียนโปรแกรมขึ้น ส่วนในการใช้งานลึก ๆ ถ้าผมมีเวลา ผมจะมาเขียนเพิ่มให้นะครับ เพราะจริง ๆ แล้ว POSTMAN สำหรับผู้ใช้นั้นเยี่ยมมาก สามารถทำ Testing Script เพื่อวินท์ดู API ในครั้งเดียวได้ รวมถึงยังทำ Documentation ได้อย่างดีอีกด้วยครับ ไว้มีเวลาจะมาเล่าต่อนะครับ

รับข้อมูลจาก POST Method ด้วย Body Parser

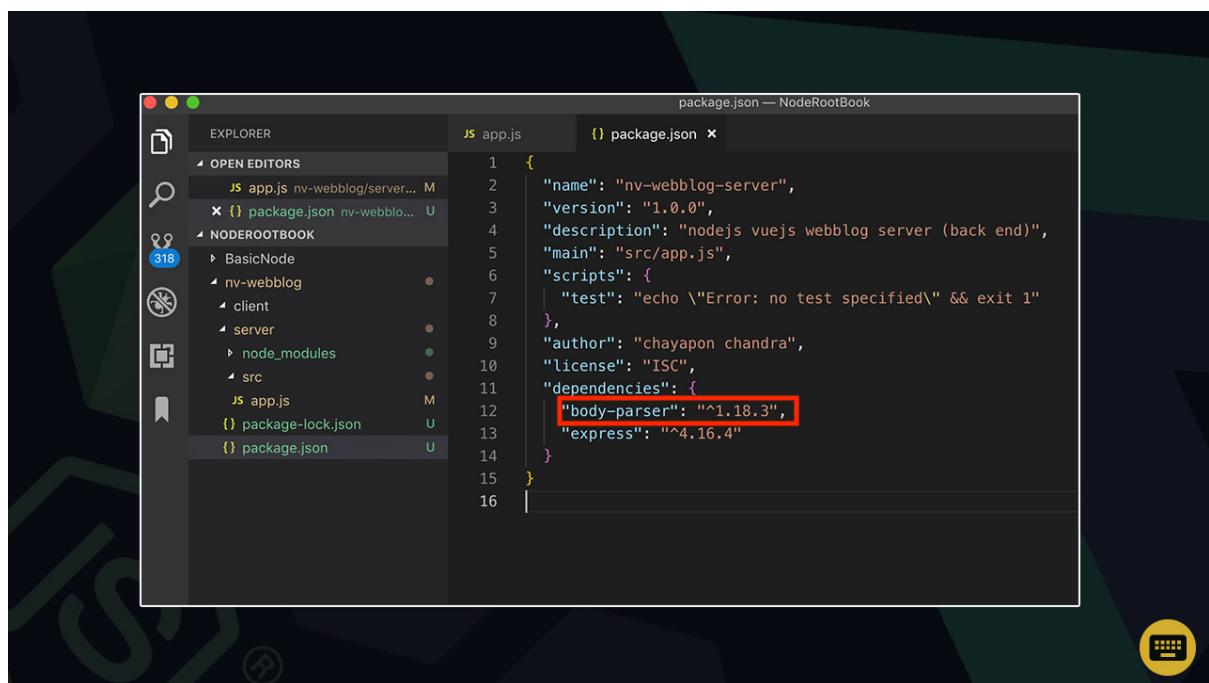
จาก API ที่เราทำการเขียนไว้ในขึ้นต้นนั้น ยังมีปัญหาอยู่เรื่องนึง คือ express จะไม่สามารถดึงข้อมูลแบบ HTTP Post Method ผ่าน ตัวแปร req.body ได้ พูดให้ง่าย ๆ คือ มาถึงตอนนี้ Server หรือ Back End ของเรา ยังไม่สามารถดึงข้อมูลที่ Front End มาใช้งานได้

Body Parser เป็น node package ตัวหนึ่งเหมือน express จะช่วยให้เราสามารถอ่านข้อมูลที่ฝั่ง client POST มาได้ผ่าน req.body โดยทำงานร่วมกันกับ express

การใช้งาน body-parser เริ่มจาก ทำการติดตั้ง body.parser ก่อนเลยครับ อป่าลีมเข้าไปที่ folder server ของเราก่อนนะครับ เพราะทำการติดตั้งที่ server ของเรา

```
npm install --save body-parser
```

เมื่อทำการติดตั้งสำเร็จแล้ว เราไปเปิด package.json ของเราขึ้นมา จะเห็นว่ามี dependencies ของ body-parser เพิ่มขึ้นมา



จากนั้นเราทำการแก้ code โปรแกรมของเราใน app.js เป็น

```
let express = require('express')  
let bodyParser = require('body-parser')  
  
const app = express()  
  
app.use(bodyParser.json())  
app.use(bodyParser.urlencoded({extended: true}))  
  
app.get('/status', function (req, res ){  
  res.send('Hello nodejs server')
```

```

    })

app.get('/hello/:person', function (req, res) {
  console.log('hello - ' + req.params.person)
  res.send('sey hello with ' + req.params.person)
})

// get user by id
app.get('/user/:userId', function (req, res) {
  res.send('ទូខ័ម្ពស់ឱ្យរាយ')
})

// get all user
app.get('/users', function (req, res) {
  res.send('ទិន្នន័យអ្នកឱ្យរាយនៃអ្នក')
})

let port = 8081
app.listen(port, function () {
  console.log('server running on ' + port)
})

```

ខាងលើ code ទាំងនេះគឺជាដំឡើង ដែលបានរាយការណ៍ឡើង។ តើវាបានរាយការណ៍ដូចណានៅក្នុងកីឡាសាស្ត្របានដឹងទៀត។

```

let bodyParser = require('body-parser')

app.use(bodyParser.json())
app.use(bodyParser.urlencoded({extended: true}))

```

ក្នុងកីឡាសាស្ត្រនេះ ពួកគេបានរាយការណ៍ការបញ្ចូនការណ៍ដែលបានរាយការណ៍ឡើង។ តើវាបានរាយការណ៍ដូចណានៅក្នុងកីឡាសាស្ត្របានដឹងទៀត។

```

let express = require('express')
let bodyParser = require('body-parser')

const app = express()

app.use(bodyParser.json())
app.use(bodyParser.urlencoded({extended: true}))

app.get('/status', function (req, res) {
  res.send('Hello nodejs server')
})

app.get('/hello/:person', function (req, res) {
  console.log('hello - ' + req.params.person)
  res.send('sey hello with ' + req.params.person)
})

// get user by id
app.get('/user/:userId', function (req, res) {
  res.send('គ្នាដំឡើងដោយ ID: ' + req.params.userId)
})

// get all user
app.get('/users', function (req, res) {
  res.send('ទីនាក់ខែមួលស្ថិតិថាមពេល')
})

// create user
app.post('/user/', function (req, res) {
  res.send('ការបង្កើតគ្នាដំឡើង: ' + JSON.stringify(req.body))
})

// edit user
app.put('/user/:userId', function (req, res) {
  res.send('ការរៀបចំគ្នាដំឡើង: ' + req.params.userId + ' : ' +
    JSON.stringify(req.body))
})

// delete user
app.delete('/user/:userId', function (req, res) {
  res.send('ការលើកគ្នាដំឡើង: ' + req.params.userId + ' : ' +
    JSON.stringify(req.body))
})

```

```

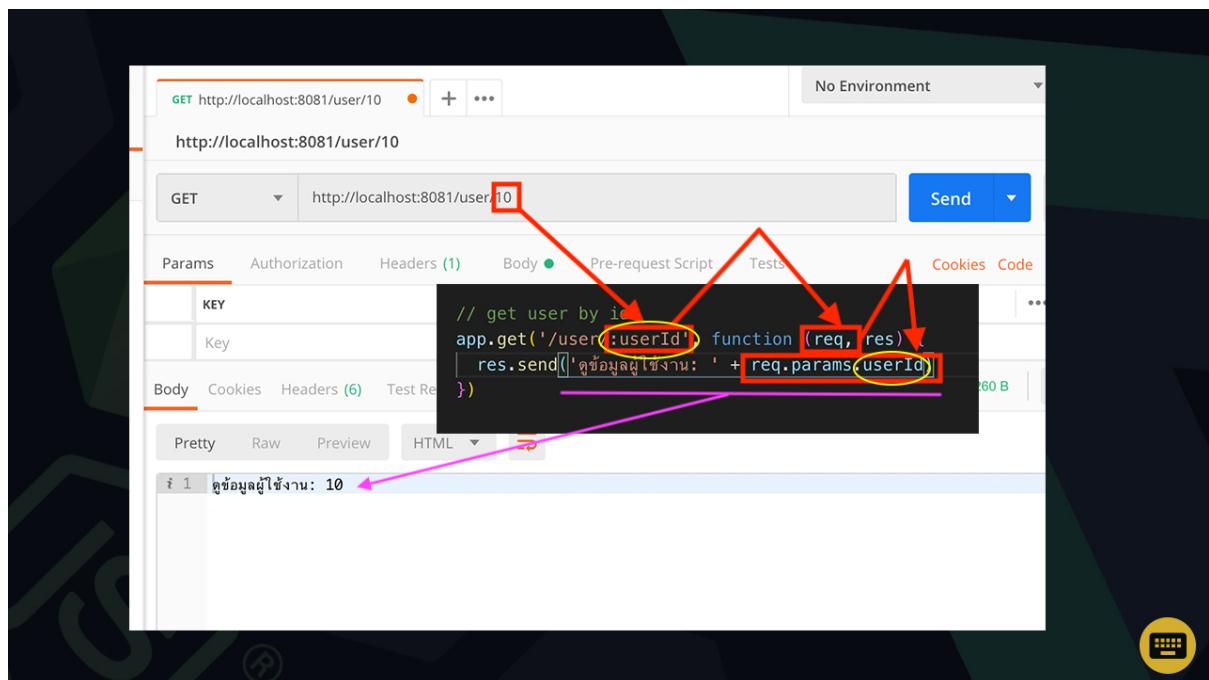
})
let port = 8081
app.listen(port, function () {
  console.log('server running on ' + port)
})

```

ซึ่งเป็นตึง require หรือ ตึง body-parser มาใช้งานผ่าน bodyParser ที่เราอ้างอิง และเรียกใช้งานผ่าน body-parser ผ่าน app หรือ express ของเรา บรรทัด สุดท้าย กำหนดให้มัน เข้ารหัสในการรับส่งข้อมูลสำหรับ Post Method ด้วยครับ

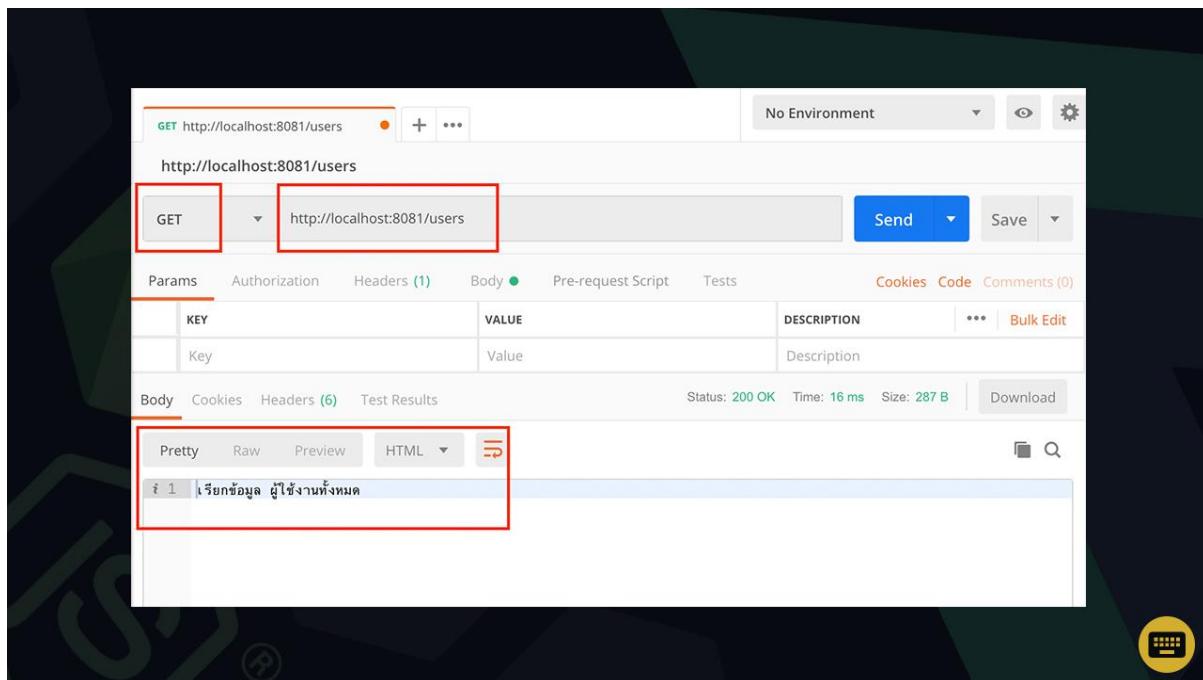
หลังจากแก้ code แล้ว ตอนนี้ webserver ของเรารองรับการ post data จากฝั่ง client แล้วนำมาใช้ผ่าน req.body แล้วครับ

ในส่วนนี้จะมี function ของ javascript ที่นำสันใจคือ **JSON.stringify** คือ การ แปลง JSON ของเราให้กลายเป็น String เพื่อแสดงผลให้เราเห็นนั่นเองครับ

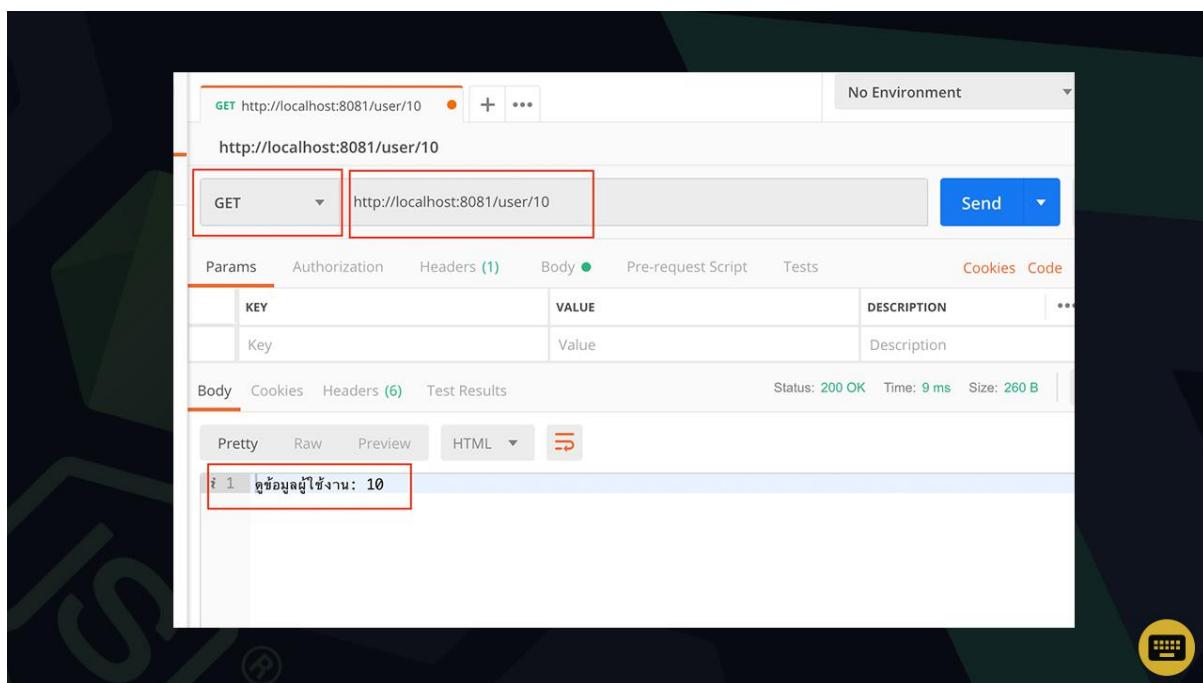


ทดสอบ API ของเราด้วย POSTMAN

เมื่อเราเรียนรู้การใช้งานโปรแกรม POSTMAN เป็นอย่างต้นแล้ว เรา ก็ทำการทดสอบ URL ต่าง ๆ ที่เราเขียนโปรแกรมไว้รองรับบน Server หรือ Back End ของเรา กัน ครับ ผมการทดสอบของผมมีดังนี้ครับ



The screenshot shows the POSTMAN interface with a successful API call. The request method is 'GET' and the URL is 'http://localhost:8081/users'. The response body contains the message 'เรียกข้อมูล ผู้ใช้งานทั้งหมด'.



The screenshot shows the POSTMAN interface with a successful API call. The request method is 'GET' and the URL is 'http://localhost:8081/user/10'. The response body contains the message 'ผู้ใช้ข้อมูลที่ ID: 10'.

The screenshot shows the Postman interface with a GET request to `http://localhost:8081/user/10`. The response body is displayed in Pretty format, showing the message "ผู้ใช้ไม่พบ: 10".

```
GET http://localhost:8081/user/10
```

Body (Pretty):

```
i 1 ผู้ใช้ไม่พบ: 10
```

The screenshot shows the Postman interface with a PUT request to `http://localhost:8081/user/15`. The request body contains the following JSON data:

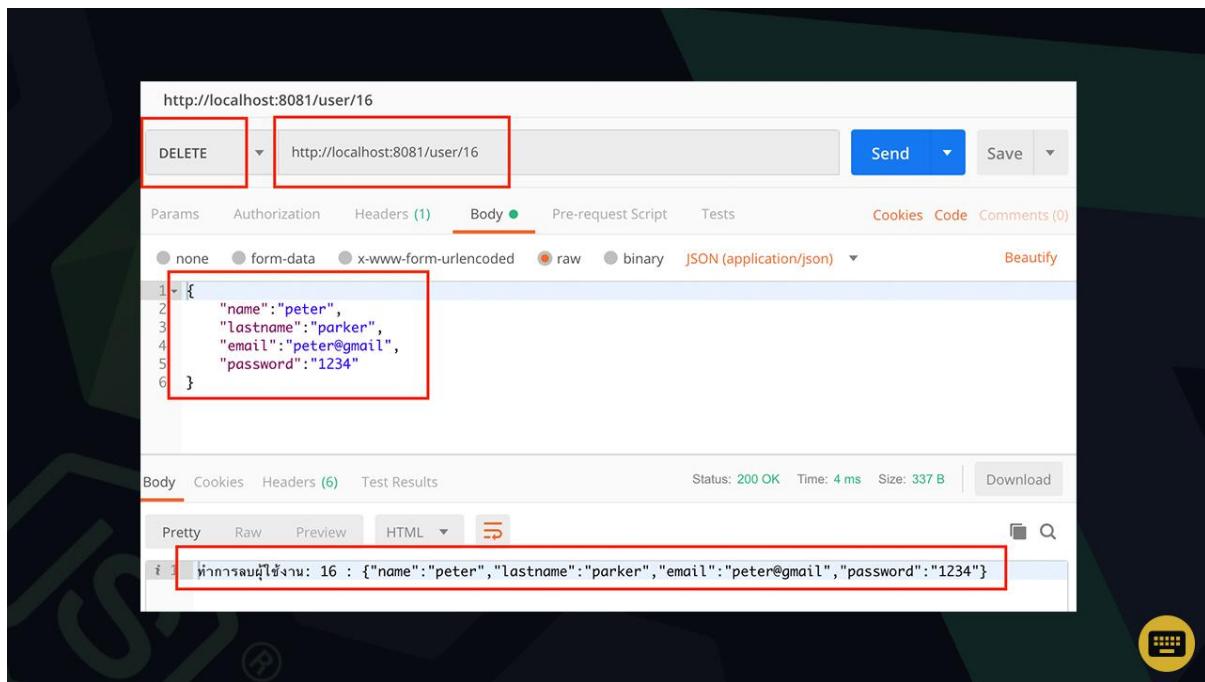
```
{"name": "peter", "lastname": "parker", "email": "peter@gmail.com", "password": "1234"}
```

The response body is displayed in Pretty format, showing the message "ทำการเพิ่มผู้ใช้: 15 : {"name": "peter", "lastname": "parker", "email": "peter@gmail.com", "password": "1234"}".

```
PUT http://localhost:8081/user/15
```

Body (Pretty):

```
i 1 ทำการเพิ่มผู้ใช้: 15 : {"name": "peter", "lastname": "parker", "email": "peter@gmail.com", "password": "1234"}
```



ตอนนี้เราได้สร้าง Web Server และ RESTFul API อย่างง่ายๆ กันได้แล้วนะครับ ในบทต่อไปจะมาสอนการแยกส่วนของโปรแกรม เพื่อให้ง่ายกับการทำงาน แก้ไข และ พัฒนาต่อในอนาคต ได้อย่างมีประสิทธิภาพมากขึ้นนะครับ อย่าลืม commit code ที่เราเขียนขึ้น github.com กันด้วยนะครับ

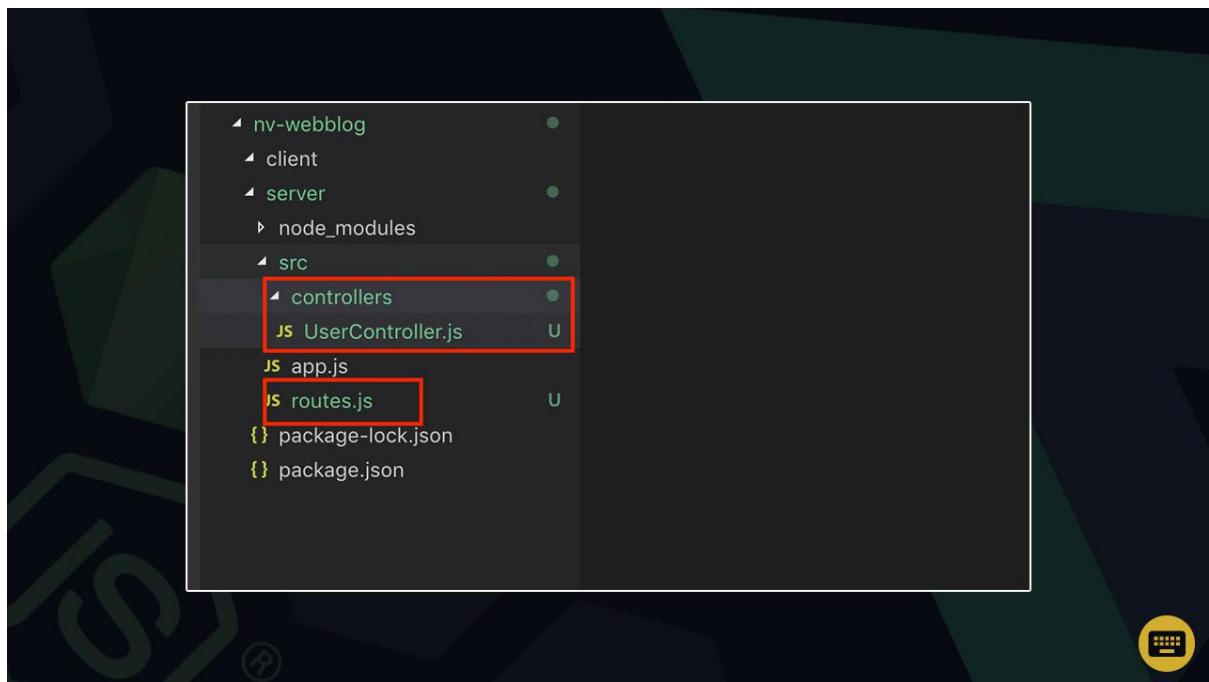
บทที่ 5 จัดสรร **code** อย่างมีอิทธิพลด้วย **MVC**

สร้าง Controllers และ Router

จากบทที่ผ่านมา เราได้ทำ RESTful API แบบ บ้าน ๆ แล้ว แต่ทุกอย่างยังคงรวมกันอยู่ในไฟล์เดียว คือ `src/app.js` ซึ่งถ้าเขียนต่อ กันยาว ๆ ไป เวลาเราแก้ไข เราจะลำบากมาก เพราะ code จะเยอะ ผสมเลยจะมาแยก code และ จัดวางให้เป็นระบบแบบ MVC คือ Model View และ Controller โดย การแยกไฟล์แบบนี้จะทำให้เราทำงานง่ายและเป็นสัดส่วนขึ้นนั่นเองครับ แต่ความจริงแล้ว Back End ของเราจะมีเพียงแต่ Model และ Controller เท่านั้น เพราะ ส่วน View นั้น ผู้จะไปใช้ `vuejs` ในการ Render หน้าบ้าน (Front End) นั่นเอง

ผู้ทำการเพิ่มโฟล์เดอร์ `controllers` ไว้เก็บไฟล์ Controller ของผู้ทั้งหมด จากในบทก่อนหน้านี้ เราจะเริ่มการสร้างส่วนจัดการผู้ใช้งานก่อน ผู้โดยสร้าง `UserController.js` เอาไว้จัดการส่วนนี้ โดยผู้สร้างไว้ในโฟล์เดอร์ "controllers" ต่อไปนี้ โปรแกรมการให้บริการในแต่ละแพนก์ ของผู้นั้นจะเก็บไว้ในโฟล์เดอร์นี้ ทั้งหมด ซึ่งเวลาแก้ไขผู้จะทำได้ง่าย ไม่งง

ต่อมา ผู้ทำการเพิ่มไฟล์ `routes.js` ไว้ที่โฟล์เดอร์ `src` ของผู้โดย ไฟล์นี้ผู้จะเอาไว้ ตอบสนองการเรียกจาก URL พร้อม Method ที่ส่งมาจากผู้ร้องขอ (Front End) และส่งมอบต่อให้ ส่วน Controller เป็นคนให้บริการ ตามที่ขอมาครับ



เมื่อทำการสร้างโฟล์เดอร์ และไฟล์ตามข้างต้นแล้ว อย่างแรกจะทำการเขียนโปรแกรมส่วนของ UserController.js ก่อน โดย ผมทำการเขียน code โปรแกรมดังนี้

```
module.exports = {
  // get all user
  index (req, res) {
    res.send('ดึงข้อมูล ผู้ใช้งานทั้งหมด')
  },

  // create user
  create (req, res) {
    res.send('ทำการสร้างผู้ใช้งาน: ' + JSON.stringify(req.body))
  },

  // edit user, suspend, active
  put (req, res) {
    res.send('ทำการแก้ไขผู้ใช้งาน: ' + req.params.userId + ' : ' +
    JSON.stringify(req.body))
  },

  // delete user
```

```

remove (req, res) {
    res.send('ทำการลบผู้ใช้งาน: ' + req.params.userId + ' : ' +
JSON.stringify(req.body))
},

// get user by id
show (req, res) {
    res.send('ดูข้อมูลผู้ใช้งาน: ' + req.params.userId)
}
}

```

คุณๆ มั้ยครับ ผม copy จาก app.js มาแก่นั้นเอง แต่ตอนนี้ UserController.js ของเรายังไม่ได้ถูกเรียกใช้งานแต่ อย่างใด และ module.exports ก็ คือ การทำให้เรียกใช้ fuction ที่เราเขียน เช่น show หรือ remove จากไฟล์อื่นได้นั้นเองครับ
มาจัดการเรื่อง Route กันต่อเลย สืบเนื่องจากครั้งแรกผมยัด code ทั้งหมดไว้ใน app.js แต่ตอนนี้ ผมอยากระบบส่วนมันให้ดูหล่อเป็นโปรแกรมเมอร์มีชั้น ขึ้นอีกนิด แต่ความจริงแล้วการจัดการลักษณะนี้จะทำให้ Implement และ Update ได้ง่ายครับ

การทำงานที่เป็นระบบ คนอื่นมาทำงานด้วย เค้าจะได้ไม่ว่าເອາ Controller ของผม ก็จะเปรียบเหมือนแผนกต่าง ๆ ในสำนักพิมพ์ของผมนั้นเอง เมื่อมีการเรียกเข้ามาที่ แต่ละแผนก (URL) พร้อมคำสั่ง (Method) ทุกความต้องการจะวิ่งผ่าน routes.js ก่อนเสมอจากนั้น จึงส่งแต่ไปที่แต่ละแผนกของผม (Controller) โดยผมจะทำการเขียนโปรแกรมในส่วนของ routes.js จะเป็นดังนี้ครับ

```

const UserController = require('./controllers/UserController')

module.exports = (app) => {
    /* RESTful API for users management */
    // create user
    app.post('/user',
        UserController.create
    )

    // edit user, suspend, active
    app.put('/user/:userId',

```

```

    UserController.put
  )

  // delete user
  app.delete('/user/:userId',
    UserController.remove
  )

  // get user by id
  app.get('/user/:userId',
    UserController.show
  )

  // get all user
  app.get('/users',
    UserController.index
  )
}

```

จาก code ข้างต้น ผม require หรือดึงเอา UserController ที่ผมสร้างไว้ที่

`src -> controllers -> UserController.js`

มาใช้งาน ผ่านตัวแปรอ้างอิงที่ชื่อว่า UserController ใน app.js ของผมนั่นเอง

โดย routes.js ของผม นั่นจะคุ้น ๆ อีกครั้ง ผมไป copy จาก app.js มาแก้ เหมือนเดิม แต่ที่ผมง ๆ ตอนเขียนครั้งแรก มันใส่แค่ "." ต่อจากตัวอ้างอิง UserController ของผม แล้วตามด้วย function ใน Controller ไม่ต้องส่งค่าไป หรือ แล้วมันรับค่ายังไง ผมจะอธิบายแบบบ้าน ๆ คือ ทั้ง routes.js และ UserController.js นั่น รับค่า function ผ่าน (req, res) เมื่อถูกเรียก ก็จะส่งค่าไป อัตโนมัติโดยครับ นั่นหมายความว่า (req, res) ของ routes.js จะเป็น ค่าเดียวกับ (req, res) ที่ส่งผ่านไปหา UserController.js นั่นเอง ไม่เชื่อผมลองไป log result ดูครับ

เสร็จแล้วหรือยัง ??? ยังครับ ตอนนี้ routes.js เรียกใช้ UserController.js ของเรา แต่ routes.js ยังไม่ได้เรียกใช้ เพราะเวลาเราสั่งรันโปรแกรม nodejs เราเรียกไป

ที่ จุดเริ่มต้น (Entry Point) คือ src/app.js นั้นหมายความว่า app.js ของเราก็ต้อง แก้ให้ไปเรียก routes มาใช้ นั่นเอง ผมทำการแก้ไขตามนี้ครับ

```
let express = require('express')
let bodyParser = require('body-parser')

const app = express()

app.use(bodyParser.json())
app.use(bodyParser.urlencoded({extended: true}))

require('./routes')(app)

app.get('/status', function (req, res ){
    res.send('Hello nodejs server')
})

app.get('/hello/:person', function (req,res) {
    console.log('hello - ' + req.params.person)
    res.send('sey hello with ' + req.params.person)
})

app.post('/hello', function (req, res) {
    res.send('OK you post - ' + req.body.name)
})

let port = 8081

app.listen(port, function () {
    console.log('server running on ' + port)
})
```

จากนั้นทำการทดสอบด้วยโปรแกรม Postman ผลลัพธ์จะเหมือนบทที่ผ่านมาจะ ครับ ไม่มีอะไรแตกต่างกันเพียงแต่ในบทความนี้ เราจัดการเรื่องโครงสร้างการ จัดเก็บ และแยก Route ออกจาก app.js รวมถึงแยก controller ออกมาไว้ให้เป็น ระบบเท่านั้นเอง

เขียนโปรแกรมเสร็จแล้วทุกครั้งอย่าลืม commit ขึ้น github กันด้วยนะครับ จะได้ backup ไปในตัว

บทที่ 6 เริ่มต้นเขียนโปรแกรมกับฐานข้อมูลกัน

ติดตั้ง sequelize , SQLite3 เพื่อใช้งาน SQLite

ในบทนี้เราจะมาทำในส่วนของ database กันนะครับ โดยในตอนนี้ผมจะใช้งาน database ยี่ห้อ SQLite นะครับ โดย ข้อดีของ SQLite นั้นจะไม่ใช่ database server แต่จะเก็บเป็นไฟล์ .sqlite ไว้ในเครื่อง server ของเราเลย ทำให้ไม่ยุ่งยากในการติดตั้งใช้งาน เพราะจัดเก็บในรูปแบบของ text file ไม่ต้องมี database server นั่นเอง

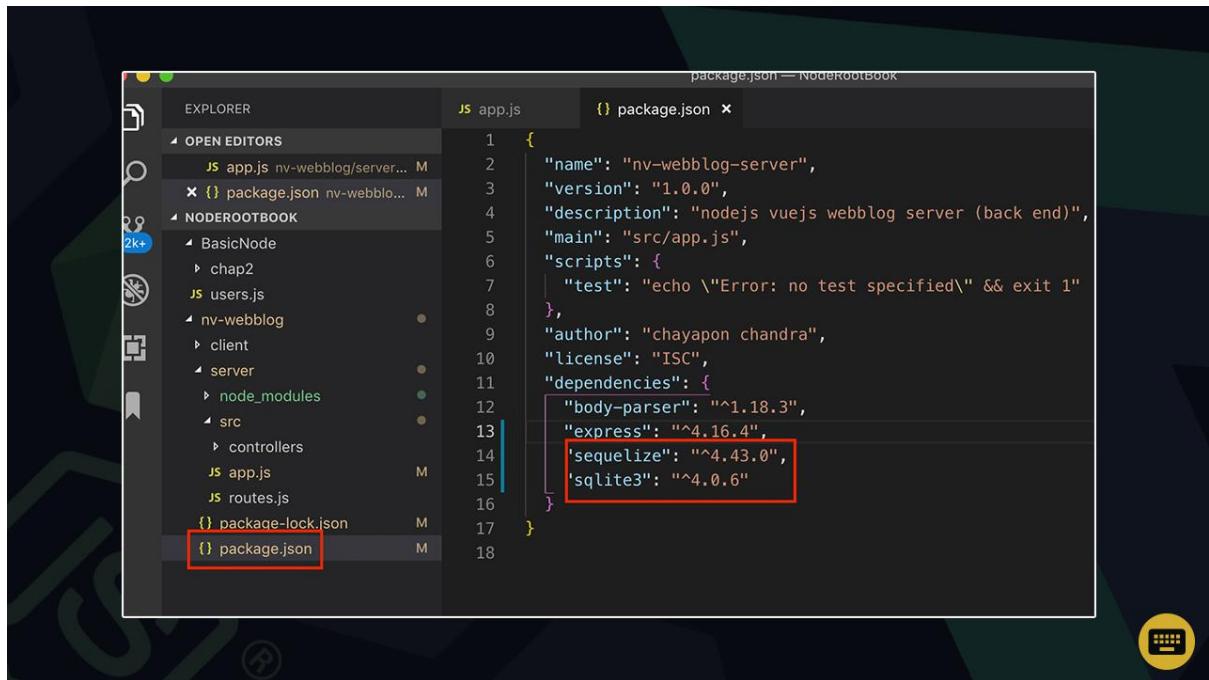
แนะนำ Sequelize

ในตอนนี้ผมจะแนะนำ node package ตัวนึงที่ชื่อว่า Sequelize ครับ เป็น package การจัดการ database ของเรา ไม่ว่าจะเป็นการสร้างฐานข้อมูล create update delete หรือที่เรารู้กันว่า CRUD โดยที่เราไม่ต้องเขียนคำสั่ง SQL ใด ๆ เลย หรือ เราอยากรู้ว่า sql query เอง ก็สามารถทำได้เช่นกันครับ

เริ่มต้นจากการ ติดตั้ง package ที่จำเป็นก่อนเลยครับ package ที่ผมใช้ คือ sqlite3 และ sequelize ครับ เราทำการติดตั้งโดย เข้าไปที่ root folder ของเว็บไซต์ส่วน server เช่น c:/nv-webblog/server หรือ ที่อื่น ๆ ตามที่เราเก็บ server ของเราวันนี้เอง ใช้คำสั่ง npm ดังนี้ครับ

```
npm install --save sqlite3  
npm install --save sequelize
```

เมื่อติดตั้ง เสร็จแล้ว เราไปดูที่ package.json ใน server ของเราจะพบว่ามี package dependencies เพิ่มขึ้นมา คือ sqlite3 และ sequelize นั่นเอง



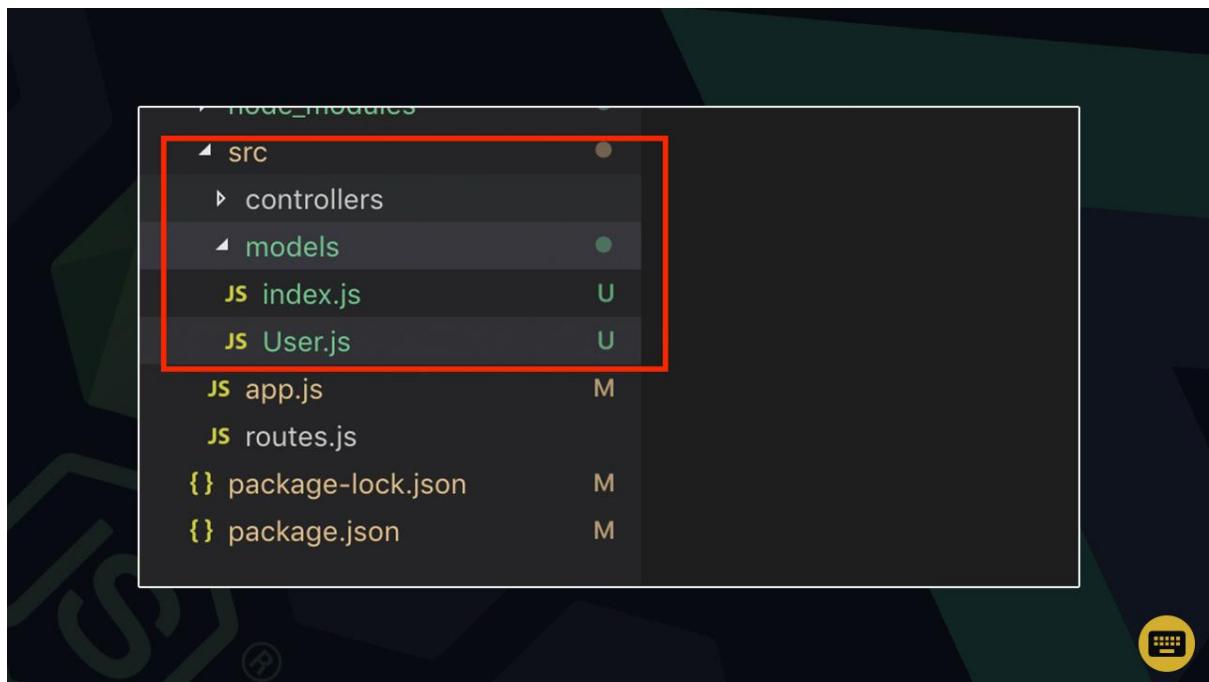
The screenshot shows the VS Code interface with the package.json file open in the editor. The file contains the following JSON code:

```
1 {  
2   "name": "nv-webblog-server",  
3   "version": "1.0.0",  
4   "description": "nodejs vuejs weblog server (back end)",  
5   "main": "src/app.js",  
6   "scripts": {  
7     "test": "echo \\\"Error: no test specified\\\" && exit 1"  
8   },  
9   "author": "chayapon chandra",  
10  "license": "ISC",  
11  "dependencies": {  
12    "body-parser": "^1.18.3",  
13    "express": "^4.16.4",  
14    "sequelize": "^4.43.0",  
15    "sqlite3": "^4.0.6"  
16  }  
17}  
18}
```

A red box highlights the dependencies section of the code, specifically the entries for body-parser, express, sequelize, and sqlite3.

สร้าง models/index.js และ model/user.js

เราจะเริ่มต้นทำ model เพื่อ สร้างและติดต่อ ฐานข้อมูลกันครับ โดยผมเริ่มต้นสร้างไฟล์เดอร์ชื่อ models ขึ้นมาครับ เพื่อเอาไว้เก็บไฟล์โปรแกรมในส่วนของ model หรือจัดการฐานข้อมูลของเรานั่นเอง จากนั้นผมทำการสร้างไฟล์ขึ้นมาสองไฟล์ ครับ ชื่อ index.js และ User.js โดย index.js ของผมจะเป็นตัวเลือก model หรือ ตารางในไฟล์เดอร์ตามที่เราต้องการเรียกใช้งาน และ User.js ของผมก็จะแทน การใช้งานฐานข้อมูลใน ตาราง User นั่นเอง



ตามระเบียบในการใช้งานฐานข้อมูล นั้นจะต้องทำ database config เสียก่อน database config นี้ เราเอาไว้กำหนด นั่น นี่ โน่น ที่เกี่ยวข้องกับ database ที่เราใช้ ผมเก็บมันไว้เป็นระเบียบ เช่นเดย โดยสร้าง folder config ขึ้นมา แล้วสร้าง config.js ไว้ภายใน โดย config.js ของผม จะทำการเขียนโปรแกรมไว้ดังนี้ครับ

```
module.exports = {
  port: 8081,
  db: {
    database: process.env.DB_NAME || 'nvWebblogDb',
    user: process.env.DB_User || 'root',
    password: process.env.DB_PASS || '',
    options: {
      dialect: process.env.DIALECT || 'sqlite',
      storage: './nvwebblog-db.sqlite'
    }
  }
}
```

ไหน ๆ ผู้พัฒนาทำการสร้าง config file ไว้แล้วผู้พัฒนาจะทำการกำหนดค่า port ของผู้ใช้งานนี้โดย เนื่องจากเวลาใครอยากรัน port จะได้ไม่ต้องไปยุ่งกับ app.js ของเรา

โดย code ที่ผู้พัฒนาเขียนขึ้นนี้จะเป็นรูปแบบของ json นั่นเอง มีจุดหน้าสนใจอยู่นิดหน่อย มีหลายคนชอบถามว่า คือ

```
database: process.env.DB_NAME || 'nvWebsiteDb'
```

code ลักษณะนี้ คือ ให้เลือกว่า ถ้ามีการกำหนด process environment หรือกำหนดการตั้งค่าหลักของโปรแกรม ให้ทำการเลือกการกำหนดค่าหลัก หากไม่มีให้ทำการเลือกค่าที่เรากำหนด เพราะบางครั้ง เราทำ config file อีกตัวภายนอก แต่ผู้ใช้งานไม่ต้องมาตั้งค่า ผู้พัฒนาแค่เปลี่ยนส่วน ๆ รู้สึกเป็นระบบมากกว่าครับ

จากนั้นผู้พัฒนาเพิ่ม code ที่ index.js ใน model ของเราครับ ผู้พัฒนาเพิ่ม code ดังนี้

```
const fs = require('fs')
const path = require('path')
const Sequelize = require('sequelize')
```

```

const config = require('../config/config')
const db = {}

const sequelize = new Sequelize (
  config.db.database,
  config.db.user,
  config.db.password,
  config.db.options
)

fs.readdirSync(__dirname)
  .filter((file) =>
    file !== 'index.js'
  )
  .forEach((file) => {
    const model = sequelize.import(path.join(__dirname, file))
    db[model.name] = model
  })

db.sequelize = sequelize
db.Sequelize = Sequelize

module.exports = db

```

code ที่ผมเขียนเริ่มต้นจาก ดึง package fs เข้ามาร่วมด้วย โดยอ้างอิงผ่านตัวแปร fs และ package path ผ่านตัวแปร path โดยทั้งสองตัวนี้เป็น package ที่ถูกติดตั้งมาตั้งแต่เราเริ่มสร้าง project node ของเราในตอนที่เราทำการติดต้อง express ครับ

แน่นอนเมื่อเราใช้ sequelize เราต้องทำ require หรือตึง package ที่ติดตั้งเข้ามา โดยผมอ้างอิงผ่านตัวแปร Sequelize นั่นเอง จากนั้น ดึง config ไฟล์ของเราตาม path ที่เราสร้างไว้ครับ

ทำการสร้างตัวแปร db = {} เอาไว้วาง ๆ เอาไว้เก็บ data ของเราในเอง แน่นอน ครับ เราจะติดต่อกับ database เราต้องสร้าง database object ขึ้นมาก่อน โดยใช้ตัวแปรตามที่เรา config ไว้

จากนั้นทำการอ่านไฟล์ database ของเรา ตาม config ไฟล์และ ชื่อ model หรือ ตาราง จากที่ Controller ส่งมา โดยการทำงานส่วนนี้ เราจะเรียกใช้งานผ่าน controller โดยจะส่งชื่อ model ที่เราต้องการใช้มาให้ครับ

```
UserController (ชื่อ model) -> Model -> index.js -> fs (config, เลือกตามชื่อ Model ที่ส่งมา) -> สร้าง Database Object เก็บตาราง ตาม model นั้น ๆ -> ส่ง data กลับ ตามคำสั่งที่กำหนด
```

ในไฟล์ index.js นี้เรียกว่าเป็นรูปแบบ สามารถ copy หรือนำไปใช้งานได้เลยครับ แต่ concept จะเป็นอย่างที่อธิบายไปในขั้นต้น ส่วนโครงสร้างจะเรื่องการใช้งานอ่าน และเขียนไฟล์ สามารถดูได้เพิ่มเติมที่นี่ครับ

<https://www.npmjs.com/package/fs>

ต่อมาผมทำการเขียนโปรแกรมในส่วนของ User.js ดังนี้ครับ

```
module.exports = (sequelize, DataTypes) => {
  const User = sequelize.define('User', {
    email: DataTypes.STRING,
    password: DataTypes.STRING,
    name: DataTypes.STRING,
    lastname: DataTypes.STRING,
    status: DataTypes.STRING,
    type: DataTypes.STRING
  })

  return User
}
```

code ในส่วนนี้ จะเป็นการกำหนด ชื่อของพิลด์และ ชนิดการเก็บค่าของพิลด์นั้น ๆ นั่นเอง แต่จะมีจุดสนใจอยู่สองส่วน คือ

```
const User = sequelize.define('User', {
```

และ

```
return User
```

ซึ่ง User หมายถึงชื่อ ตารางในฐานข้อมูลของเรานั้นเอง จากการเขียนโปรแกรมที่ผ่านมา เราได้ทำการสร้าง User Model ด้วย Sequelize ไว้แล้ว พร้อมทั้งสามารถเลือก Model อื่น ๆ โดยกำหนดจากการเรียกใช้งาน จาก index.js ในกรณีที่เราเพิ่ม model อื่นเข้ามาอีก ก็สามารถทำการเพิ่มไฟล์ Model เข้าไปได้เลย

มาถึงตอนนี้เราจะทำการเรียกใช้งาน และสร้าง database ของเราผ่าน app.js โดยจะแก้ไขโปรแกรมใน app.js ดังนี้

```
let express = require('express')
let bodyParser = require('body-parser')
const {sequelize} = require('./models')

const config = require('./config/config')

const app = express()

app.use(bodyParser.json())
app.use(bodyParser.urlencoded({extended: true}))

require('./routes')(app)

app.get('/status', function (req, res ){
  res.send('Hello nodejs server')
})

app.get('/hello/:person', function (req,res) {
  console.log('hello - ' + req.params.person)
  res.send('sey hello with ' + req.params.person)
})
```

```
app.post('/hello', function (req, res) {
  res.send('OK you post - ' + req.body.name)
})

let port = process.env.PORT || config.port

sequelize.sync({force: false}).then(() => {
  app.listen(port, function () {
    console.log('Server running on ' + port)
  })
})
```

ชี้ง

code

โปรแกรมที่ผมเขียนเพิ่มขึ้นมา

คือ

```
const {sequelize} = require('./models')

sequelize.sync({force: false}).then(() => {
  app.listen(port, function () {
    console.log('Server running on ' + port)
  })
})
```

โดยผม require './models' เข้ามาในระบบ ในลักษณะนี้ จะเป็นการ require มาทั้งโฟล์เดอร์และทำการเลือกจาก index.js โดยส่งชื่อ model เข้าไป ตอนเราเรียกใช้ใน controller เราจะติดต่อกับ Model นั้นได้ทันทีครับ

เมื่อเรา require model โดยการอ้างอิง เข้า sequelize จะมี function ชื่อว่า sync เพื่อทำการสร้าง database หรือติดต่อกับ database ของเราในกรณีที่สร้างไว้แล้ว จึงไปทำงาน ในส่วนของ

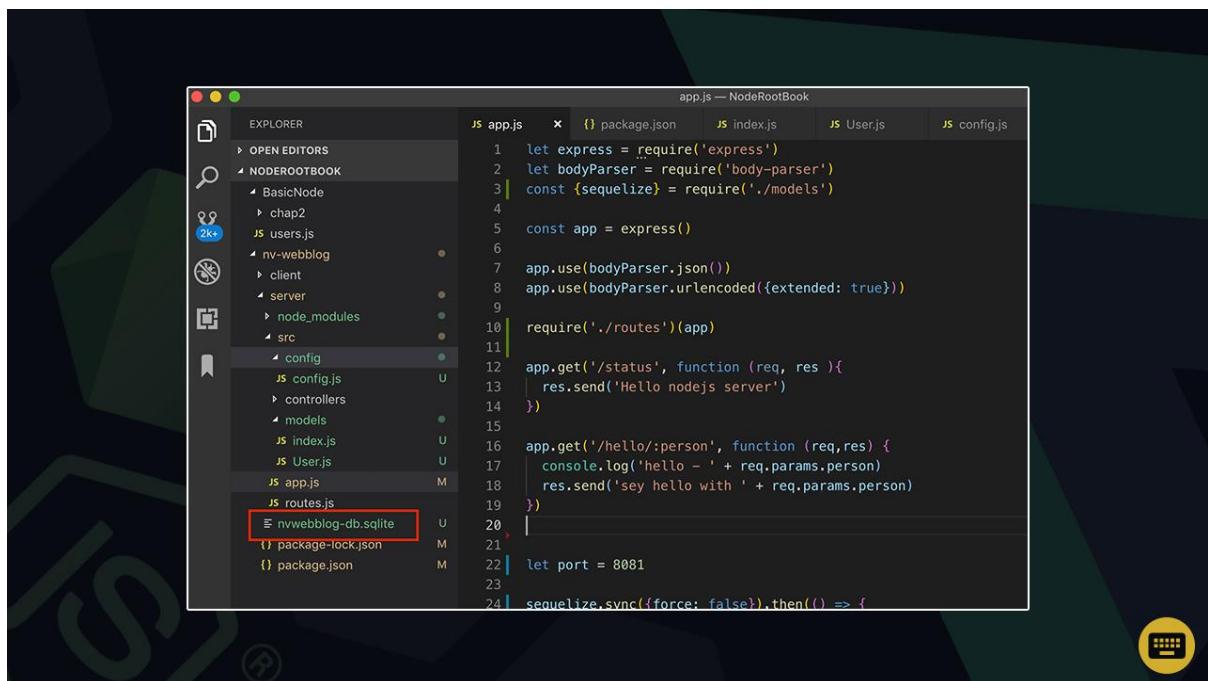
```
app.listen()
```

ต่อไปนี้เอง ในส่วนนี้จะมี code ที่นำเสนอ อยู่อีกส่วนหนึ่งคือ

```
sequelize.sync({force: false}).then(() => {
```

โดย force = true จะเป็นการลบแล้วสร้างตารางใหม่ตามที่กำหนดไว้ใน model นั้น ๆ หลังจากทำการรันครั้งแรกหลังจากการแก้ไข model หรือ สร้าง model แล้ว จะทำการแก้ส่วนนี้เป็น force = false เพื่อไม่ให้ไปยุ่งกับโครงสร้าง ที่เราได้ทำไว้แล้วรวมถึง force = true จะทำการลบข้อมูลของเรามาด้วย จะทำอะไรอย่าลืม backup ก่อนนะครับ โดยวิธีการ backup ของ sqlite นั้นง่ายมาก เราก็ทำการ copy ไฟล์ .sqlite ไว้ได้เลยครับ

เมื่อเราทำการรันโปรแกรม webserver ของเรา โปรแกรมที่เราเขียนขึ้น จะทำการสร้าง file nvwebblog-db.sqlite ให้อัตโนมัติทันที



The screenshot shows a dark-themed code editor interface. On the left is the Explorer sidebar displaying a project structure:

- OPEN EDITORS
- NODEROOTBOOK
- BasicNode
- chap2
- 2K+
- nv-webblog

 - client
 - server
 - node_modules
 - src
 - config
 - config.js
 - controllers
 - models
 - index.js
 - User.js
 - app.js
 - routes.js

A file named "nvwebblog-db.sqlite" is highlighted with a red rectangle in the Explorer sidebar.

The main editor area shows the "app.js" file content:

```
let express = require('express')
let bodyParser = require('body-parser')
const {sequelize} = require('./models')

const app = express()

app.use(bodyParser.json())
app.use(bodyParser.urlencoded({extended: true}))

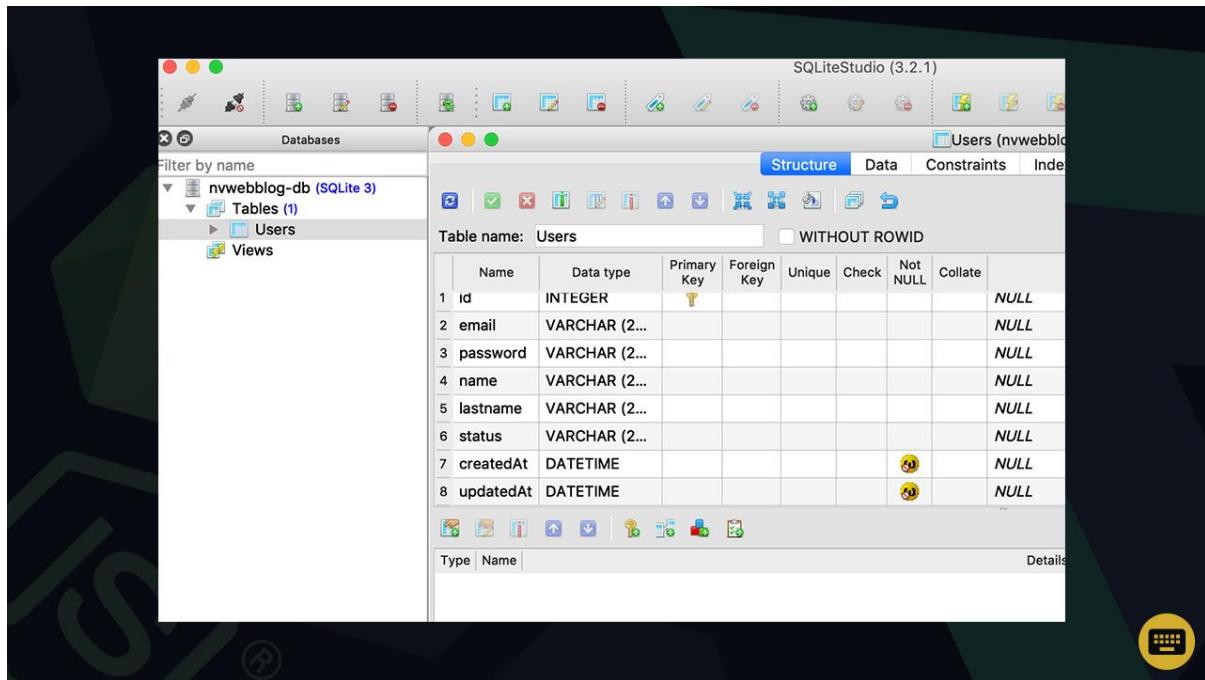
require('./routes')(app)

app.get('/status', function (req, res){
  res.send('Hello nodejs server')
})

app.get('/hello/:person', function (req,res) {
  console.log('hello - ' + req.params.person)
  res.send('sey hello with ' + req.params.person)
})

let port = 8081

sequelize.sync({force: false}).then(() => {
```



ผู้ใช้โปรแกรม SQLiteStudio เปิดดูโครงสร้างภายในฐานข้อมูล จะเห็นดังนี้ หรือท่านจะใช้ SQLite IDE อื่น ๆ ก็ได้นะครับ ไม่ต้องกังวล โค้ดสามารถ download ฟรีได้ที่นี่ครับ <https://sqlitestudio.pl/index.rvt>

ซึ่งจะตรงกับ User Model ที่เราสร้างไว้นั่นเอง

```
module.exports = (sequelize, DataTypes) => {
  const User = sequelize.define('User', {
    email: DataTypes.STRING,
    password: DataTypes.STRING,
    name: DataTypes.STRING,
    lastname: DataTypes.STRING,
    status: DataTypes.STRING,
    type: DataTypes.STRING
  })

  return User
}
```

ตอนนี้เรารสามารถสร้าง Model ด้วย sequelize เพื่อใช้งานฐานข้อมูล SQLite ได้แล้ว ในบทต่อไปเรามาเริ่ม สร้าง แก้ไข และ ลบ ข้อมูลในฐานข้อมูลของเราผ่าน sequelize กันนะครับ

synchronize กับ asynchronous

ก่อนเราจะทำเรื่องการ สร้าง แก้ไข และลบข้อมูลในฐานข้อมูลของเรา เราควรทำความเข้าใจ nodejs หรือคุณใช้ปัจจุบันทำงานของ nodejs หรือ javascript กันก่อน javascript นั้นทำงานแบบ Asynchronous หรือเรียกว่าแบบ “ไปต่อไม่รอแล้วนะ” สิ่งที่ทำงานแล้วจะรอดไป สิ่ง code ลำดับต่อไป โดยไม่รอการตอบผลลัพธ์การทำงาน แล้วไปเขียน call back ตักไว้ พอทำงานเสร็จจะวิ่งไปที่ call back ที่ผูกกับสิ่งที่สั่งออกไป ทำให้มันเร็วมาก แต่สำหรับการทำงานร่วมกับฐานข้อมูลถึงจะเป็น asynchronous ผมว่ามันไม่แตกต่างกันมาก เพราะยังไงเราต้องรอให้ได้ข้อมูลจนครบก่อน ถึงจะ Render Page นั้นอยู่ดี ถ้าเราจะ แก้ไขข้อมูลแล้วรอให้ข้อมูลวิ่งกลับมาทีละไฟล์ไปเรื่อย ๆ User คงงงงวยน่าดู แต่ถ้า Form ของเราเมื่อปิดแล้ว แสดงผลเป็นหน้า ๆ อันนี้เร็วกว่ากันอย่างเห็นได้ชัดเลยครับ

แล้วการทำงานแบบ synchronize ล่ะ ?

การทำงานแบบ Synchronize คือ สิ่งที่ทำงาน รอผลลัพธ์ แล้วจึงไปทำงานในคำสั่งลำดับต่อไป ตัวอย่าง synchronize system ได้แก่ PHP นั่นเอง และเนื่องจากเราทำงานกับ SQL การทำงานกับ SQL สามารถทำได้ทั้งสองอย่าง ทั้ง synchronize และ asynchronous ก็ได้ ถ้าเราเขียนโปรแกรมแบบ async เราต้องวางแผนการจัดการ call back และการแสดงผลให้ดี ไม่งั้น bug จะมาเยี่ยมแบบหาไม่ค่อยเจอคับ

แต่การใช้งานฐานข้อมูลของเราในหนังสือเล่มนี้ ในการทำ webblog ของเรานั้น เราจะใช้งานในรูปแบบ synchronize ครับ เนื่องจากเขียนง่าย ใช้งานได้จริงและ debug ได้ง่าย เนื่องจากเราสามารถเขียนโปรแกรมพื้นฐาน แต่สร้างงานได้จริงครับ

มาเล่นกับ CRUD กันเถอะ

ในบทนี้เราจะมาทำการเขียนโปรแกรม เพื่อทำการสร้าง แก้ไข และลบ User หรือผู้ใช้งานของเรากันนะครับ ในบทที่ผ่าน ๆ มาเราได้ทำการออกแบบ URL รวมถึง Controller ที่ทำงานไว้แล้ว ตอนนี้เราจะทำการเขียนโปรแกรมในส่วนของ Controller เพื่อทำการจัดการฐานข้อมูลหรือ Model ของเราเองครับ

โดยเราทำการแก้ code โปรแกรมของเราที่ UserController.js กันดังนี้ครับ

```
const {User} = require('../models')

module.exports = {
  // get all user
  async index (req, res) {
    try {
      const users = await User.findAll()
      res.send(users)
    } catch (err){
      res.status(500).send({
        error: 'The users information was incorrect'
      })
    }
  },
  // create user
  async create (req, res) {
    try {
      const user = await User.create(req.body)
      res.send(user.toJSON())
    } catch (err) {
      res.status(500).send({
        error: 'Create user incorrect'
      })
    }
  },
  // edit user, suspend, active
  async put (req, res) {
    try {
      await User.update(req.body, {
```

```

        where: {
            id: req.params.userId
        }
    })
    res.send(req.body)
} catch (err) {
    res.status(500).send({
        error: 'Update user incorrect'
    })
}
},
// delete user
async remove (req, res) {
try {
    const user = await User.findOne({
        where: {
            id: req.params.userId
        }
    })

    if(!user){
        return res.status(403).send({
            error: 'The user information was incorrect'
        })
    }

    await user.destroy()
    res.send(user)
} catch (err) {
    res.status(500).send({
        error: 'The user information was incorrect'
    })
}
},
// get user by id
async show (req, res) {
try {
    const user = await User.findById(req.params.userId)
    res.send(user)
} catch (err) {
    req.status(500).send({
        error: 'The user information was incorrect'
    })
}
}

```

```
        })
    }
},
}
```

โปรแกรมของเรา มีจุดที่น่าสนใจอยู่ไม่กี่ส่วน คือ

```
const {User} = require('../models')
```

เป็นการอ้างถึงโฟล์เดอร์ models ของเรานั้นเอง โดย {User} จะเป็นการบอกว่า ไปเลือกไฟล์ User.js หรือ ชื่อ Model ว่าเป็น User Model ตรงตามที่เรา return และสร้างไว้ตอนเราเขียน Model ใน User.js ครับ

ผมจะยกตัวอย่าง code โปรแกรม ในการ post data เพื่อไปสร้าง User หรือผู้ใช้งาน มาอธิบายนะครับ ถ้าเข้าใจส่วนนี้ก็นำจะเข้าใจทั้งหมดละครับ

```
async create (req, res) {
  // res.send(JSON.stringify(req.body))
  try {
    const user = await User.create(req.body)
    res.send(user.toJSON())
  } catch (err) {
    res.status(500).send({
      error: 'Create user incorrect'
    })
  }
},
```

โปรแกรมส่วนนี้ ก็คือ function create มีการรับค่าจาก req และ res ส่งมาเป็นแบบ object สำหรับอ้างอิงและเรียกใช้งานผ่านทั้งสองตัวเอง โดย req นี้เราใช้อ่านค่าตัวแปร ที่ url แบบมา ส่วน res เอาไว้ส่งสิ่งที่เราจะส่งออกไปที่ web browser นั้นเอง ใน code ส่วนนี้ มีคุ้สร้างคู่สม ออยู่สองคู่ครับ คือ

try และ catch

เมื่อมี try ต้องมี catch เสมอ ใน code block ของ try จะ-copy ตรวจจับ error จับอะไร ยังไงไม่ต้องไปยุ่งกับมัน แค่รู้ว่า ถ้ามี error จะวิ่งไป catch ทันทีพอกรับ เรา ก็ทำการเขียนโปรแกรมจัดว่า error แล้วจะทำยังไง จะวิ่งไปไหนต่อ แต่สำหรับผม ให้แสดงผลที่ console หน้าจอค่า ๆ ที่เราัน webserver ของเราเลย ว่า error อะไร จะได้แก้ถูก คู่พระรองไปแล้ว ต่อมาที่คู่พระเอกกันครับ

async และ await

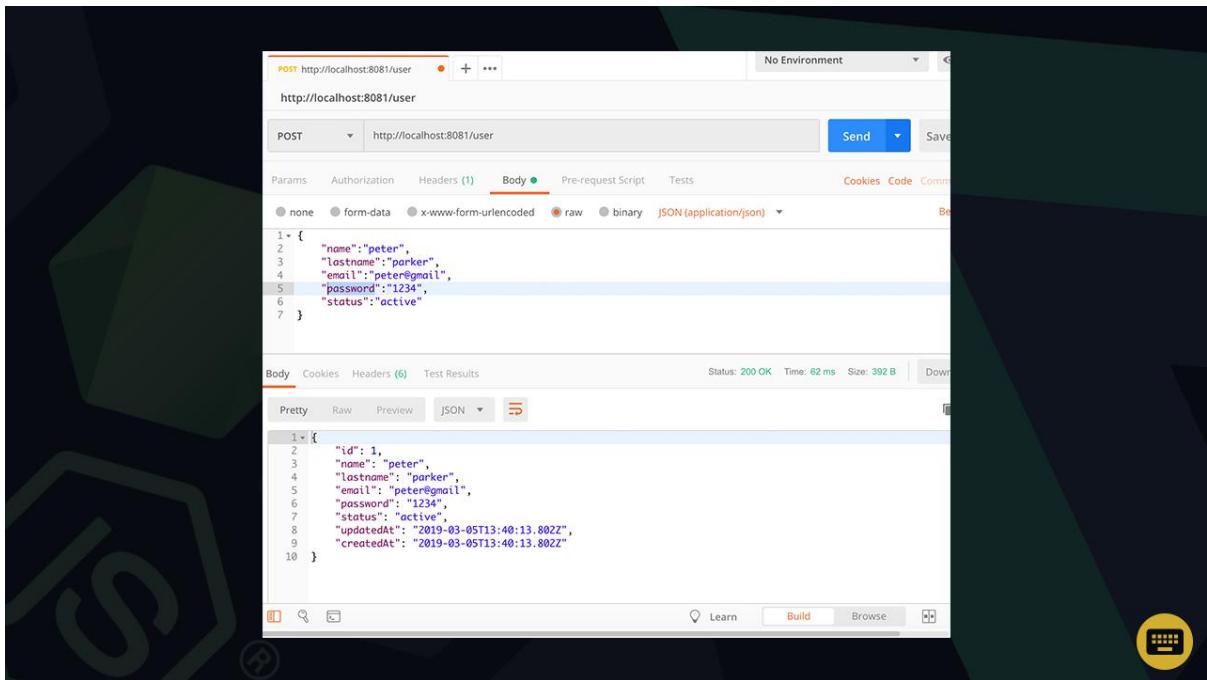
เมื่อ await ต้องมี async เสมอ ว่าแต่ทำไมต้องมี เพราะผมเคยเล่าไว้ การทำงาน ของ javascript นั้นแบบ synchronize คือ “ไปต่อไม่รอแล้วนะ” ทำงานโดยไม่รอ การตอบกลับจาก server แต่การทำงานกับ ฐานข้อมูลนั้น บอยครึ้งที่เราไปต่อ ไม่ได้ เราเลยต้องรอ await กับ async คือ บอกให้หยุดรอผลลัพธ์ ก่อนแล้วค่อยไป ต่อนั้นเอง โดย await ใส่ไว้ที่ คำสั่งที่ต้องการจะหยุดรอ และ async ใส่ไว้ที่หน้า function ครับ

ต่อมารามาดู code ส่วนที่จัดการสร้าง User ให้เรา กัน โดย การ Post ค่ามาทาง HTTP ผ่าน url ที่เรากำหนดไว้นั้น การสร้าง user หรือผู้ใช้งานใหม่ (create user) จะทำการ post ค่า json มาให้ โดยเรา express จะทำการอ่านค่าผ่าน body-parser อ่านยังไง ทำยังไง ไม่ต้องสนใจให้เสียเวลา รู้ว่า เค้า post json ลงมา เราอ่านผ่าน req.body ได้เลย และก็ทำการโยน json ของเราต่อผ่านเข้าไปที่ User.create() ซึ่งเป็น Function ที่ sequelize เค้าทำให้ เป็นอันเสร็จพิธี sequelize ก็จะไป insert user ของเราใน Database ให้ทันทีครับ ง่ายแสนง่าย เลย

```
const user = await User.create(req.body)
```

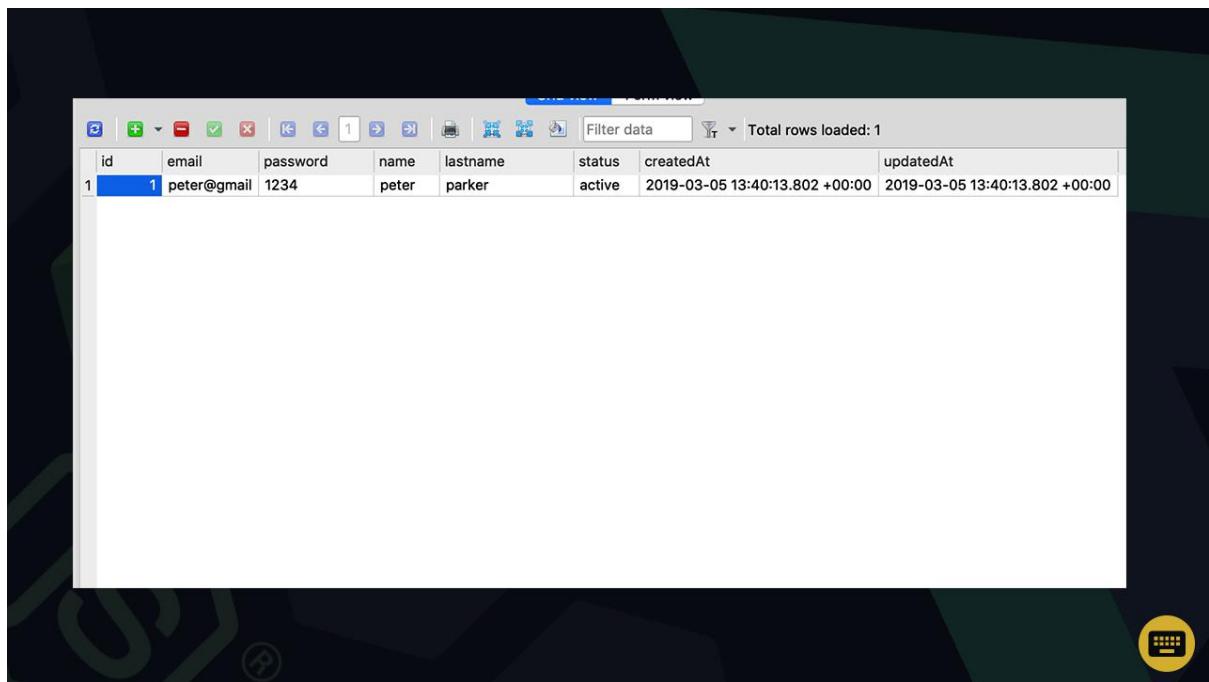
เมื่อเราเข้าใจ Code โปรแกรมของเราแล้ว เราจะทำการทดสอบด้วยโปรแกรม Postman ของเรากันครับ

ผมเริ่มจากการทำการสร้าง User ก่อนเลย โดยทำการ Post Json ตาม Format ที่เรา สร้างไว้ใน models/User.js ครึ่งไม่ได้ไปเปิดดูครับ



จากการพิมพ์คำสั่ง POST JSON ไปที่ <http://localhost:8081/user/> เพื่อทำการสร้าง User ของผู้ใช้ code โปรแกรมที่ผมเขียนไว้ คือ เมื่อสร้างเสร็จแล้วให้อ่านค่าจาก ฐานข้อมูลส่งกลับมาแสดงผลด้วย ซึ่งได้ผลลัพธ์ดังภาพ

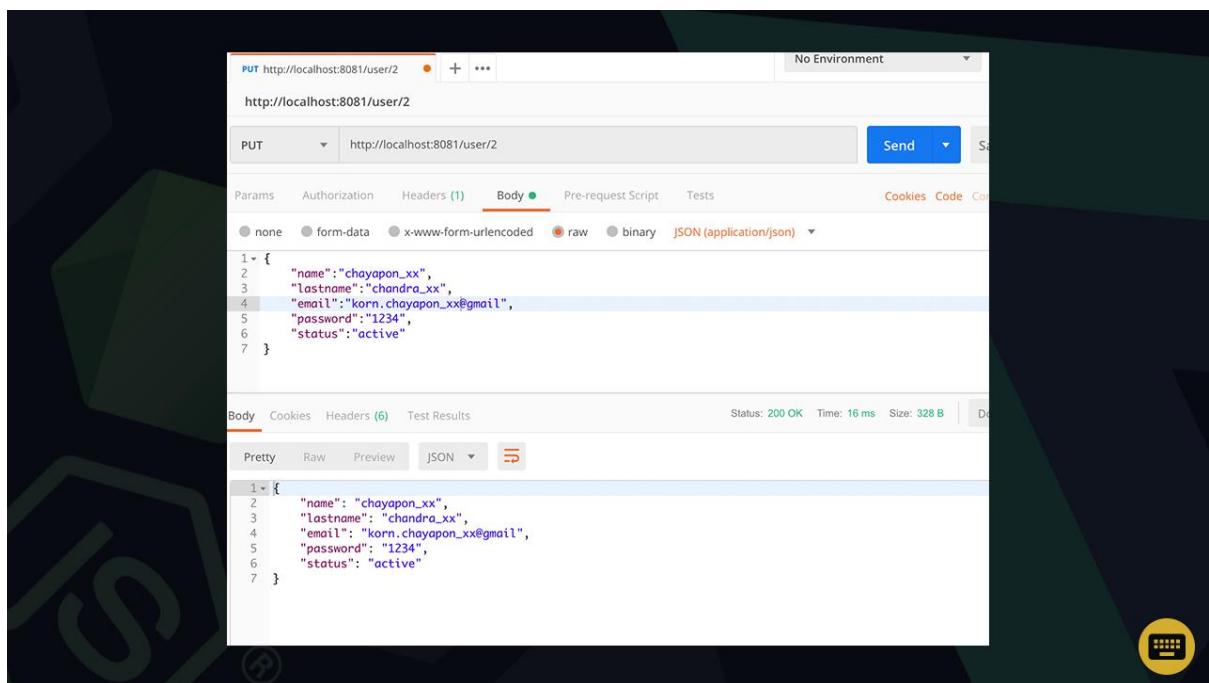
เมื่อเราเปิด ฐานข้อมูล Sqlite ของเราด้วย SQLite Studio หากโปรแกรมของเราทำงานไม่ผิดพลาดจะมีข้อมูลที่เราสร้าง หรือ Insert เข้าไปในตาราง User ของเรา



A screenshot of a database viewer application. At the top, there's a toolbar with various icons. Below it, a message says "Total rows loaded: 1". The main area is a table with the following columns: id, email, password, name, lastname, status, createdAt, and updatedAt. There is one row of data:

	id	email	password	name	lastname	status	createdAt	updatedAt
1	1	peter@gmail	1234	peter	parker	active	2019-03-05 13:40:13.802 +00:00	2019-03-05 13:40:13.802 +00:00

จากนั้นผมทำการแก้ไขข้อมูลของ user ของเรา โดยการ put json พร้อมระบุ id ไปที่ <http://localhost:8081/user/{id ของเรา}>



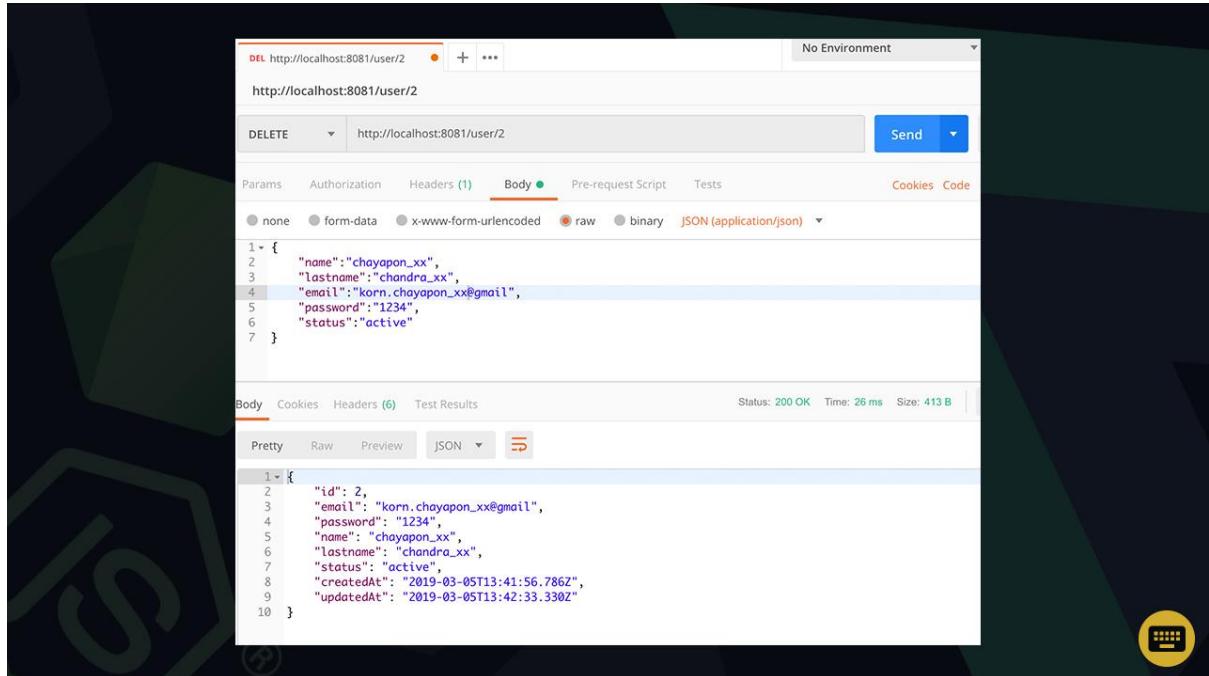
A screenshot of the Postman application interface. The request is a PUT to `http://localhost:8081/user/2`. The Body tab is selected, showing the JSON payload:

```

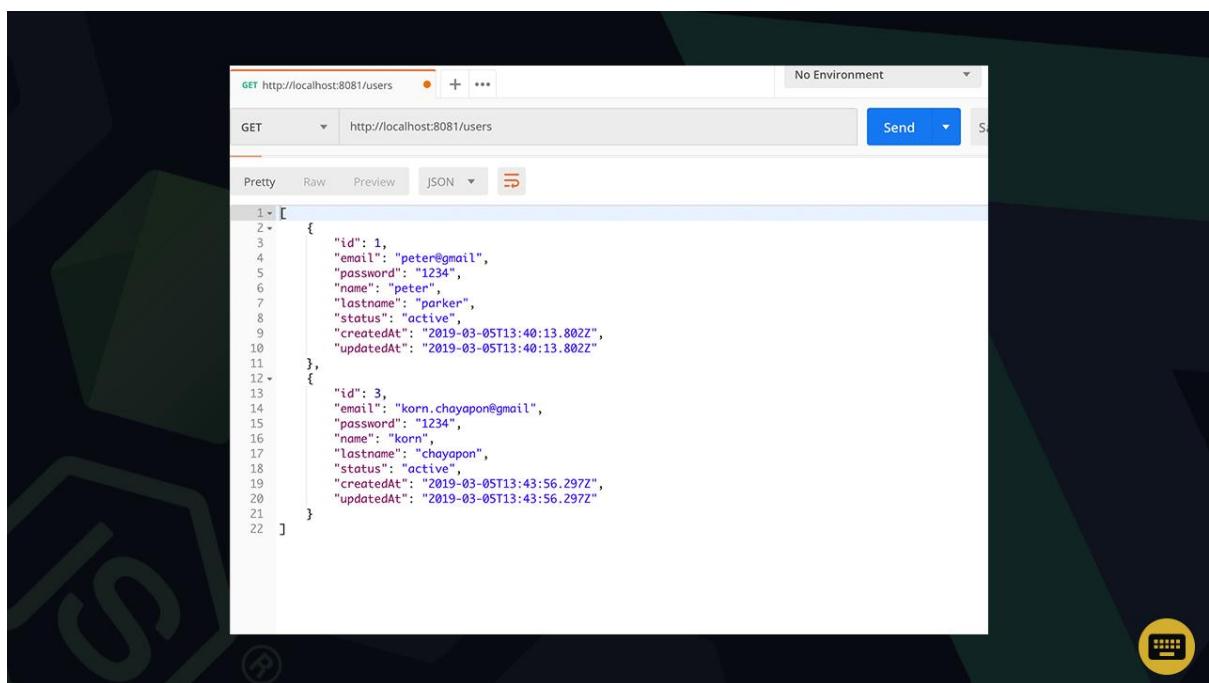
1 {
2   "name": "chayapon_xx",
3   "lastname": "chandra_xx",
4   "email": "korn.chayapon_xx@gmail",
5   "password": "1234",
6   "status": "active"
7 }
  
```

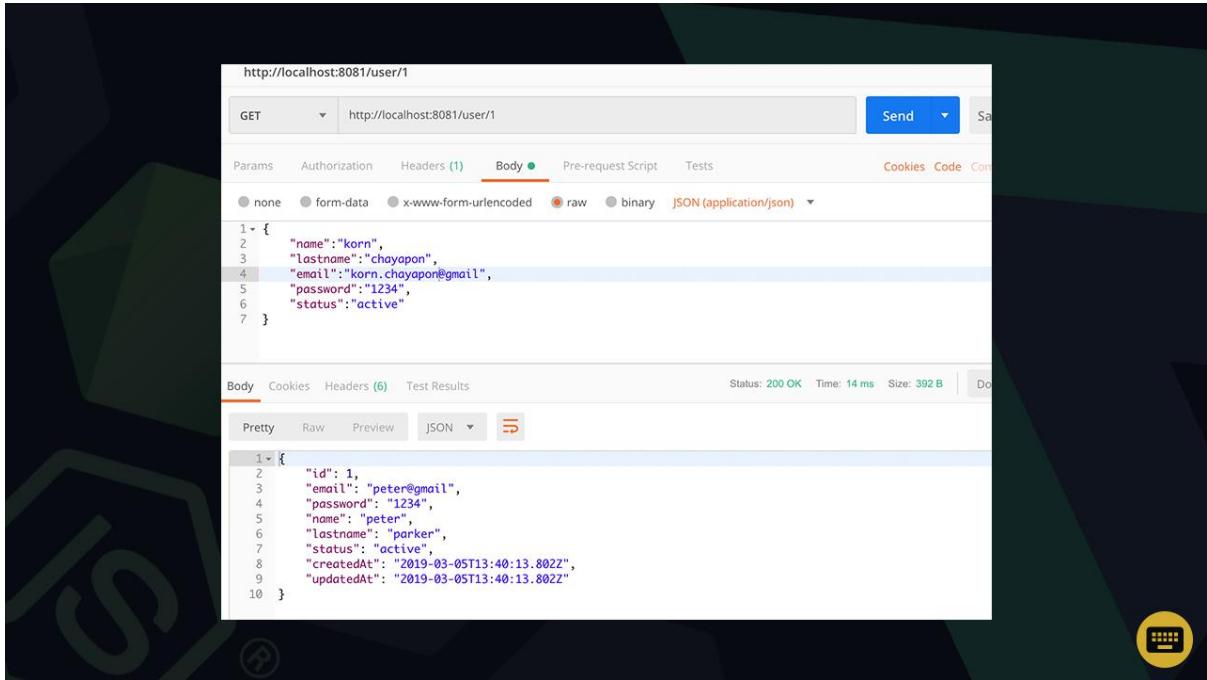
The response shows a status of 200 OK with a response body identical to the request payload.

ในทางเดียวกันเมื่อเรา Delete Method ไปที่ <http://localhost:8081/user/{id> ของเรา} จะได้ผลลัพธ์ดังภาพ เพราะผู้สั่งให้ลบกลับข้อมูล User ที่ส่งไปทำทำงานได้สำเร็จ



เมื่อผมทำการ get ค่าอุปกรณ์ทั้งหมด หรือระบุ id ของ user ที่ผมต้องการอ่านค่า จะได้ข้อมูลดังภาพ





มาถึงตอนนี้ Back End ของเราในส่วน User เกือบจะสมบูรณ์แล้วครับ

บทที่ 7 (crash course) เริ่มต้นกับ vuejs

ทำไมต้อง vuejs

vuejs เป็น javascript framework เอาไว้ทำหน้าบ้าน หรือ Front End ตัวนึง เมื่อ่อน react และ angularJS แล้วทำไม่ถึงต้องเป็น vuejs ไม่เป็นตัวอื่น ผมตอบว่า มันเล็ก มันไม่กินทรัพยากร เครื่องมากนัก syntax ของมัน มีการแทรก data ไปใน tag HTML ารมณ์คล้าย PHP ไม่เขียน syntax วุ่นวายเหมือน React ทำให้ง่าย ต่อการเรียนรู้และเข้าใจได้มากกว่า

work เหรอ คนใช้ react เยอะนะครับ ยี่ห้อ facebook ด้วยอีก

องกรค์ ระดับโลกเค้าก็ใช้กันนะครับ เช่น Alibaba, Adobe, IBM, NASA, Nintendo, Trivago และอีกเยอะครับ แต่ผมไม่ได้สนใจว่าใครจะใช้นะครับ ผม สนใจว่า feature ที่มันมีตอบโจทย์ งานที่เราจะทำหรือเปล่า แค่นั้นเอง สำหรับผม vuejs ตอบโจทย์มากครับ ตัวอย่างเช่นผมเขียน webblog ของผมด้วย PHP เป็นเดือน ๆ แต่ผมใช้ vuejs + nodejs ทำเว็บไวต์ gooddev.me ของผมเสร็จใน 10 วัน ผม apply ไปทำ web อีกอีก ใช้เวลาเพียงแค่ 4-5 วันเองครับ จะมี เหตุผลอะไร ที่จะไม่ใช้ vuejs

การใช้งานผ่าน CDN

เราสามารถใช้งาน vuejs CDN ได้เหมือน jQuery เลยครับ สำหรับคนที่ไม่ว่าการใช้งานแบบ CDN คืออะไร การใช้งานหรือติดตั้งแบบ CDN คือ เอา Code ของ library หรือ framework ที่เราใช้มาแปะครับ เช่น

```
<script src="https://cdn.jsdelivr.net/npm/vue"></script>
```

พอแปะแล้ว เราจะทำการเขียน code เพื่อใช้งาน vuejs ได้ทันทีเลย สำหรับ CDN ข้างบน คือ สำหรับ Development หรือ ตอนพัฒนา ส่วนสำหรับ Production หรือ ใช้งานบน Server เค้าให้ใช้ตัวนี้

```
<script  
src="https://cdn.jsdelivr.net/npm/vue@2.6.7/dist/vue.js"></script>
```

แต่เราความจริงเลยครับ ผมไม่เคยใช้ทั้งสองตัว ผมใช้แต่ webpack ครับ

เริ่มสร้าง project vuejs โดยใช้ webpack

การสร้าง project vuejs โดยใช้ webpack หรือ ผมเรียกว่า vue-cli นั้น ข้อดีของ มัน คือ จัดการโปรเจกของเราเป็นได้อย่างระบบ และติดตั้ง library หรือ package ที่จำเป็นมาให้ เวลาเราพัฒนาโปรแกรม ไม่ต้องมาค่อยกด refresh 保存 ระบบที่เราพัฒนา หรือ Front End ของเรา จะ update ให้ทันที รวมถึงการสั่ง build เพื่อ deploy หรือติดตั้งบน Server จริง ได้ง่าย ปรับ config ต่างได้ง่าย

ทำการติดตั้ง vue-cli กันก่อนเลยครับ เปิด Terminal ของเราขึ้นมา ไม่ต้องกังวล เรื่องโฟล์เดอร์นะครับ เพราะการติดตั้งแบบ -g คือ global หรือติดตั้งให้เรียกใช้งาน ได้ทุกที่นี่เอง

```
npm install -g vue-cli
```

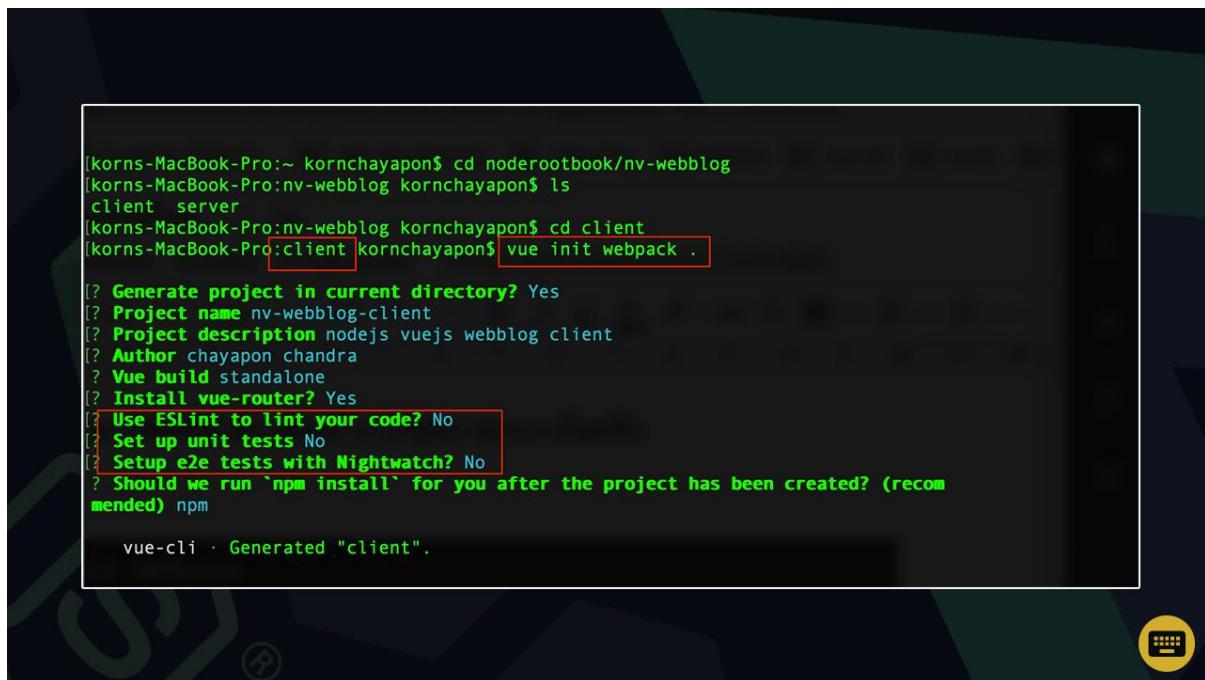
ทำการติดตั้งเสร็จแล้ว เรา ก็เข้าไปที่โฟล์เดอร์ client ของเรา กันครับ เช่น

```
d:/NodeRootBook/nv-webblog/client
```

จากนั้นทำการพิมพ์คำสั่ง เพื่อสร้าง project ของเรา กันครับ

```
vue init webpack .
```

โดยปกติ “.” ที่ต่อจาก webpack มักจะเป็นชื่อโปรเจกของเรา แต่ในกรณีนี้ผม ต้องการใช้มันไว้ในโฟลเดอร์ client นี่ล่ะครับ จะได้ไม่สับสน เมื่อทำการรันคำสั่ง เราจะทำการป้อน ข้อมูลให้มันตามภาพครับ



```
[korns-MacBook-Pro:~ kornchayapon$ cd noderootbook/nv-webblog
[korns-MacBook-Pro:nv-webblog kornchayapon$ ls
client server
[korns-MacBook-Pro:nv-webblog kornchayapon$ cd client
[korns-MacBook-Pro:client kornchayapon$ vue init webpack .

[?] Generate project in current directory? Yes
[?] Project name nv-webblog-client
[?] Project description nodejs vuejs weblog client
[?] Author chayapon chandra
[?] Vue build standalone
[?] Install vue-router? Yes
[?] Use ESLint to lint your code? No
[?] Set up unit tests No
[?] Setup e2e tests with Nightwatch? No
[?] Should we run `npm install` for you after the project has been created? (recommended) npm

vue-cli · Generated "client".
```

ในการพัฒนาติดตั้ง vuejs ด้วย vue-cli เพื่อทำการเขียนโปรแกรมในส่วน Front End ของเราที่โฟล์เดอร์ client โดยเราต้องเข้าไปที่ โฟล์เดอร์ client ในโปรเจคของเรา ก่อนจะครับ ถึงใช้คำสั่งติดตั้ง vue-cli

จากนั้นก็ทำการใส่ข้อมูลโปรเจคของเราเข้าไป แล้วกด Enter ครับ จะมีส่วน ESLint ซึ่งเป็นแจ้ง warning การเขียนโปรแกรมในรูปแบบของ ES6 ผิดไม่ใช้ เพราะเขียนใหม่ ๆ จะแยกไม่ออกรวบรวมใน error จาก code ผิดพลาด หรือพิมพ์ code ไม่สวยงามรูปแบบของ ES6 (ESLint สำหรับผมคือ ตัวเช็ค code สวยงามเอง) และ พาก Unit Test ต่าง ๆ ผิดไม่ได้ทำการติดตั้งด้วย เช่นกัน เพราะไม่ค่อยได้ใช้ ถ้าต้องการ UnitTest เราเขียนเองทำงานได้ดีและยืดหยุ่นมากกว่าครับ แต่เราจะไม่ได้เรียนในหนังสือเล่มนี้นะครับ แต่ไม่ต้องห่วงครับ ทำงานเยอะ ๆ จะ Debug เก่งเองครับ และสุดท้ายผมเลือก npm เป็นคำสั่งในการติดตั้ง node package ของเรานะครับ

เมื่อเราทำการติดตั้งเสร็จแล้ว ให้เราทำการพิมพ์

```
npm run dev
```

Terminal ของเราจะรัน และโหลด อญญาติพัก แล้วจะไปค้างอยู่ที่

```
DONE          Compiled      successfully      in      3002ms
12:03:00

| Your application is running here: http://localhost:8080
```

แสดงว่า ตอนนี้เรารวมเขียน vuejs กันแล้วคับ

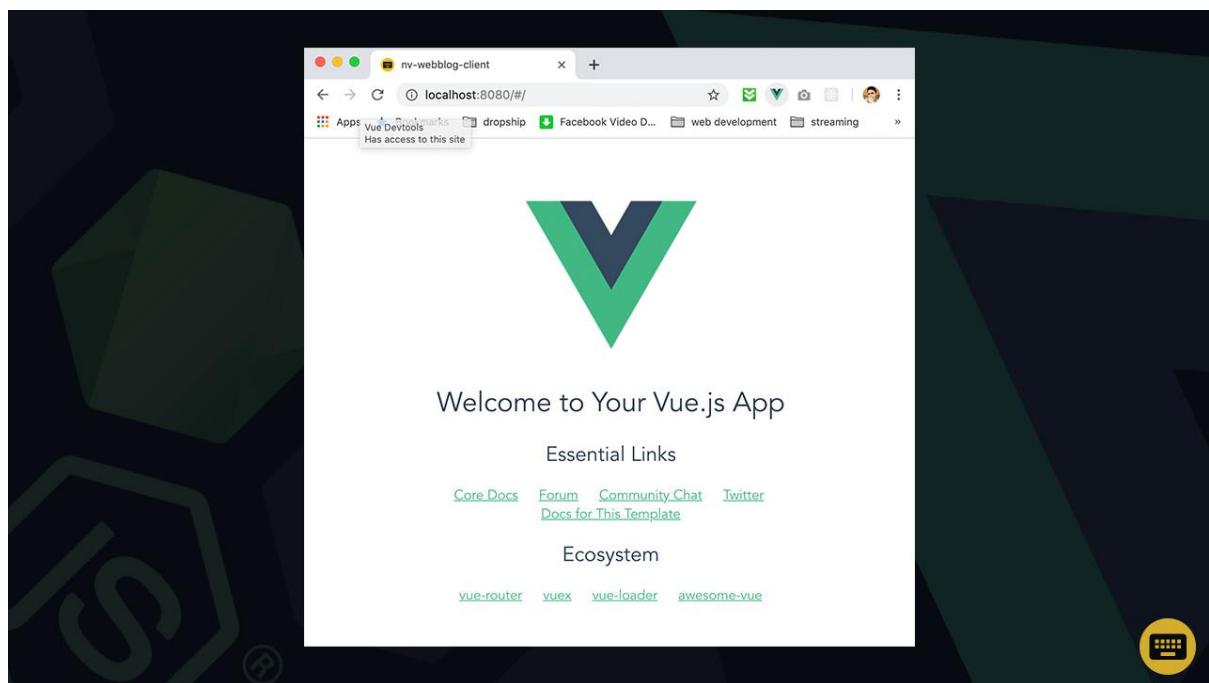
เปิด Browser ของเราขึ้นมา แล้วทำการ URL ของเราดังนี้

```
http://localhost:8080/
```

Browser ของเราจะถูก Redirect ไปที่

```
http://localhost:8080/#/
```

แล้ว Browser ของเรา แสดงผลดังภาพ นั่นคือ vuejs ของเราทำงานแล้วนั่นเอง ครับ



หลังจากเราทำการติดตั้ง vuejs ด้วย vue-cli กันเรียบร้อยแล้ว เราทำการเปิดโปรแกรม VS Code ขึ้นมา แล้วเปิดเข้าที่โฟล์เดอร์ nv-webblog ของเราจะเห็นว่า vue-cli ทำการติดตั้งอะไรมาให้เราอย่างละเอียด เราไม่ต้องไปสนใจทั้งหมดให้ปวดหัวครับ เรามาดูแค่ส่วนต่าง ๆ ดังภาพ ส่วนอื่น ๆ เราปล่อยไปก่อนครับ

The screenshot shows a code editor with a dark theme. On the left is a file tree for a Vue.js project:

```

client
├ build
├ config
├ node_modules
└ src
  ├ assets
  └ components
    └ HelloWorld.vue
  ┌ router
  ┌ index.js
  └ App.vue
  ┌ main.js
  ┌ static
  ┌ .babelrc
  ┌ .editorconfig
  ┌ .gitignore
  ┌ .postcssrc.js
  ┌ index.html
  ┌ package-lock.json
  ┌ package.json
  ┌ README.md

```

The `main.js` file is open on the right, showing the following code:

```

3 import Vue from 'vue'
4 import App from './App'
5 import router from './router'
6
7 Vue.config.productionTip = false
8
9 /* eslint-disable no-new */
10 new Vue({
11   el: '#app',
12   router,
13   components: { App },
14   template: '<App/>'
15 })

```

ความสัมพันธ์ระหว่าง index.html, main.js, App.vue, router และ components

ส่วนประกอบที่สำคัญที่เราสนใจ คือ `index.html`, `main.js`, `App.vue`, `router/index.js` โดยการทำงาน จะเริ่มที่ `index.html` และ `main.js`

เมื่อ `main.js` ทำงานจะทำการสร้าง `Vue object` หรือสร้าง `Vue` ของเราขึ้นมาด้วยคำสั่ง `new` โดยส่ง `Pararmeter` หรือ `ตัวแปรเข้าไป` เพื่อบอกเงื่อนไขในการสร้าง `object` นั้นเอง โดย

```

el: '#app', -> ให้名ของ id="app" ใน index.html และ render
router, -> เชื่อมโยงกับ router ที่โหลดมา
components: { App }, -> ใช้ component นี้มา render
template: '<App/>' -> บอกว่า App คือ component นั้น

```

```

index.html
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="utf-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1">
6     <title>nv-webblog-client</title>
7   </head>
8   <body>
9     <div id="app"></div>
10    <!-- built files will be auto injected -->
11  </body>
12 </html>
13

main.js
1 // The Vue build version to load with the `vue-loader`
2 // (runtime-only or standalone) has been swapped out by
3 import Vue from 'vue'
4 import App from './App'
5 import router from './router'
6
7 Vue.config.productionTip = false
8
9 /* eslint-disable no-new */
10 new Vue({
11   el: '#app',
12   router,
13   components: { App },
14   template: '<App/>'
15 })

```

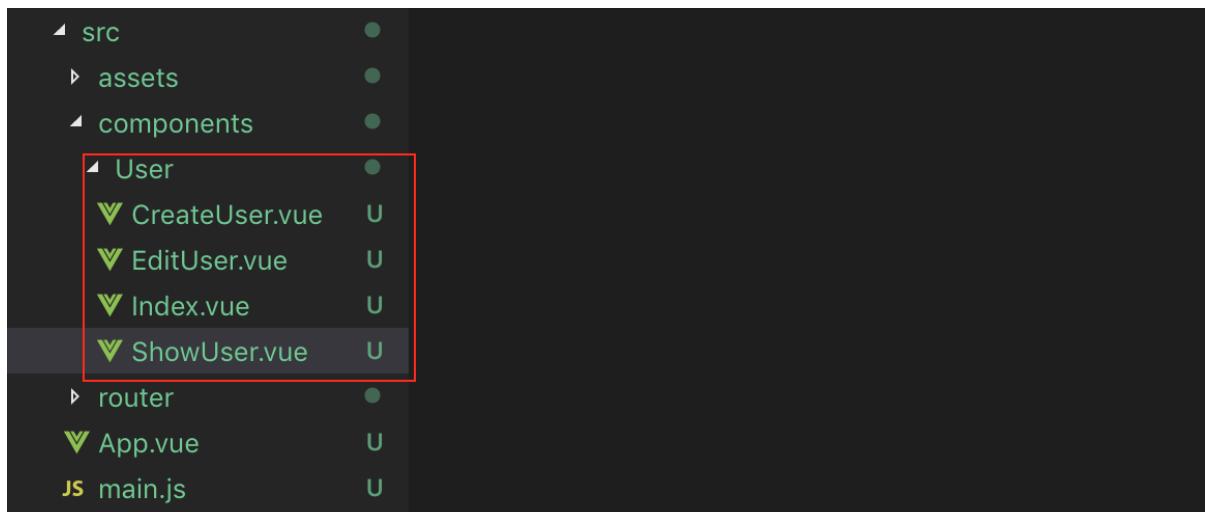
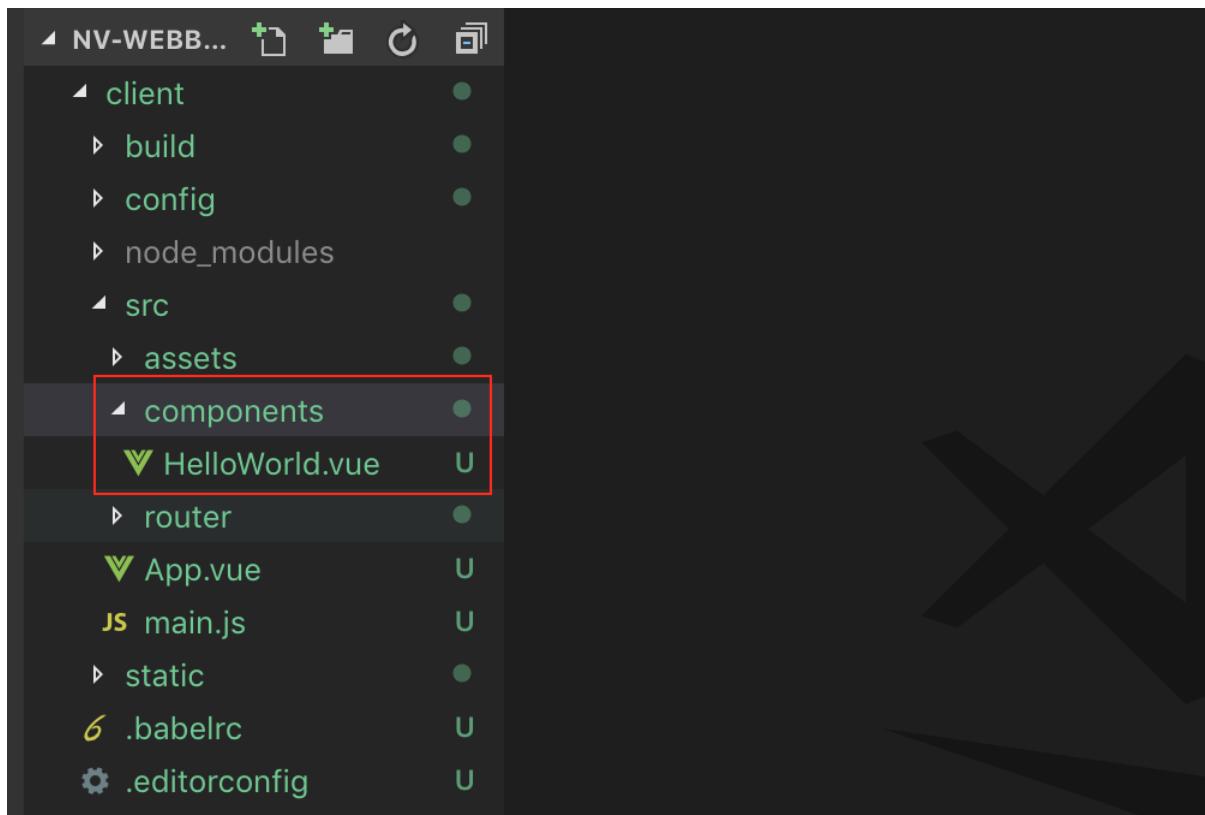
เป็นการบอกว่า App ของเราเป็น Vue Component

ในหัวข้อนี้ผมเพียงอธิบายว่ามันโหลดอะไรมาบ้าง ทำงานตอนไหน และเขื่อมโยงกันอย่างไร ถึงตอนนี้ถ้ายังงง ยังไม่ต้องไปกังวลนะครับ เข้าใจบ้างไม่เข้าใจบ้างก็ปล่อยผ่านกันไปก่อน เพราะตอนนี้เรายังไม่ได้เริ่มเขียนอะไร์กัน พอเราเขียนๆ ไปจะเริ่มค่อยๆ คุ้นเคยและเข้าใจไปเองครับ เพราะจริงๆ เราเกี่ยวกับไฟล์เหล่านี้ไม่มาก จะไปแก้ไขมากหน่อยก็ที่ router/index.js เท่านั้นครับ

เริ่มต้นทำ back office

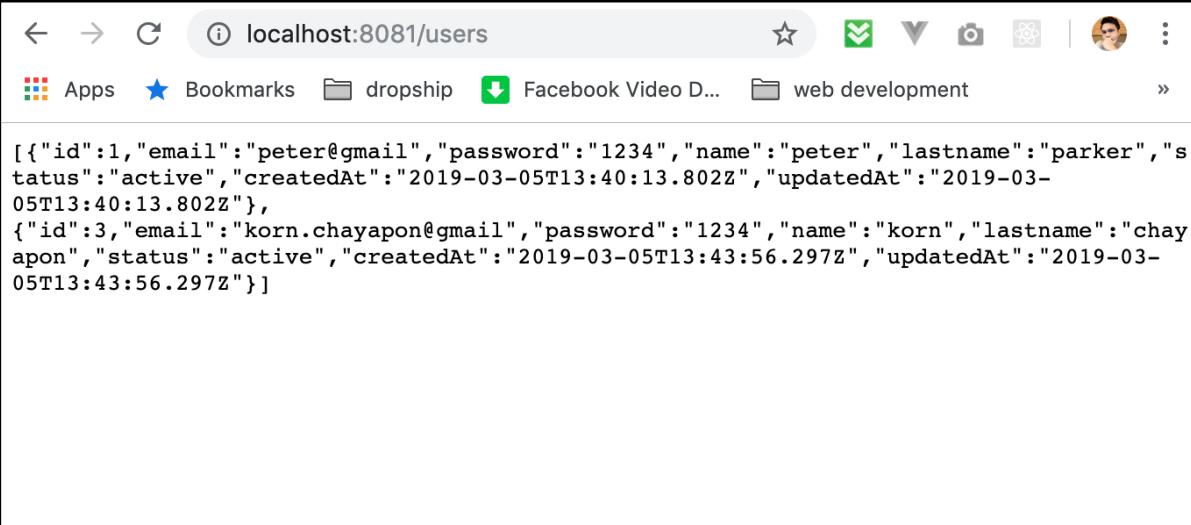
มาถึงตอนนี้เราจะเริ่มต้นทำ Front End ในส่วนของ Back Office เพื่อใช้จัดการ Back End ในส่วนของ Controller กันก่อนเลย

เปิด code ของเราขึ้นมาด้วย VS Code ครับ แล้วเปิดไปที่ folder ตามภาพ ทำการลบ HelloWorld.vue ออก และสร้างโฟล์เดอร์ชื่อ Users ใน โฟล์เดอร์ Components จากนั้นทำการสร้าง Index.vue, CreateUser.vue, EditUser.vue, ShowUser.vue ดังภาพ



เรามาเริ่มกันที่ API ที่เราเขียนนะครับ เราทำการทดสอบ API ของเราโดยการพิมพ์ url ของเราดังนี้ (อย่าลืมรัน node server ของเรา ก่อนนะครับ)

```
http://localhost:8081/users
```



```
[{"id":1,"email":"peter@gmail.com","password":"1234","name":"peter","lastname":"parker","status":"active","createdAt":"2019-03-05T13:40:13.802Z","updatedAt":"2019-03-05T13:40:13.802Z"}, {"id":3,"email":"korn.chayapon@gmail.com","password":"1234","name":"korn","lastname":"chayapon","status":"active","createdAt":"2019-03-05T13:43:56.297Z","updatedAt":"2019-03-05T13:43:56.297Z"}]
```

นั่นคือ API ที่เราทำไว้ พร้อมใช้งานแล้วครับ

เราเปิดไฟล์ Client ชื่อว่า

```
src/components/Users/Index.vue
```

ขึ้นมาครับ

แล้วทำการใส่ Code ดังนี้ครับ

```
<template>

</template>
<script>
export default {

}
</script>
<style scoped>

</style>
```

เราจำง่าย ๆ code จะแบ่งเป็นสามส่วนคือ template, script และ style

template เราเอาไว้ใส่ code html ของเรา script เราเอาไว้ใส่ vue code ที่เราจะเขียนด้วย javascript ส่วน style คือ css เอาไว้แต่งหน้าทาปาก เว็บไซต์ของเรา scoped คือ ให้ style นี้ใช้ภายใน components นี้เท่านั้น ไม่ไปมีร่วกับคนอื่น

ผมจะใส่ ข้อความ ไว้ว่า ที่นี่คือ get all user ไว้ก่อนนะครับ เวลาดึง component นี้มาใช้งานจะได้รู้ว่ามาจาก component นี้ครับ ผมเพิ่ม code เข้าไปใน template ด้วยนี้ครับ

```
<template>
  <div>
    <h1>Get All Users</h1>
  </div>
</template>
<script>
export default {

}
</script>
<style scoped>

</style>
```

ผมทำการใส่ Code ใน CreateUser.vue, EditUser.vue และ ShowUser.vue ตามลำดับดังนี้นะครับ

CreateUser.vue

```
<template>
  <div>
    <h1>Create User</h1>
  </div>
</template>
<script>
export default {

}
```

```
</script>
<style scoped>

</style>
```

EditUser.vue

```
<template>
  <div>
    <h1>Edit User</h1>
  </div>
</template>
<script>
export default {

}

</script>
<style scoped>

</style>
```

ShowUser.vue

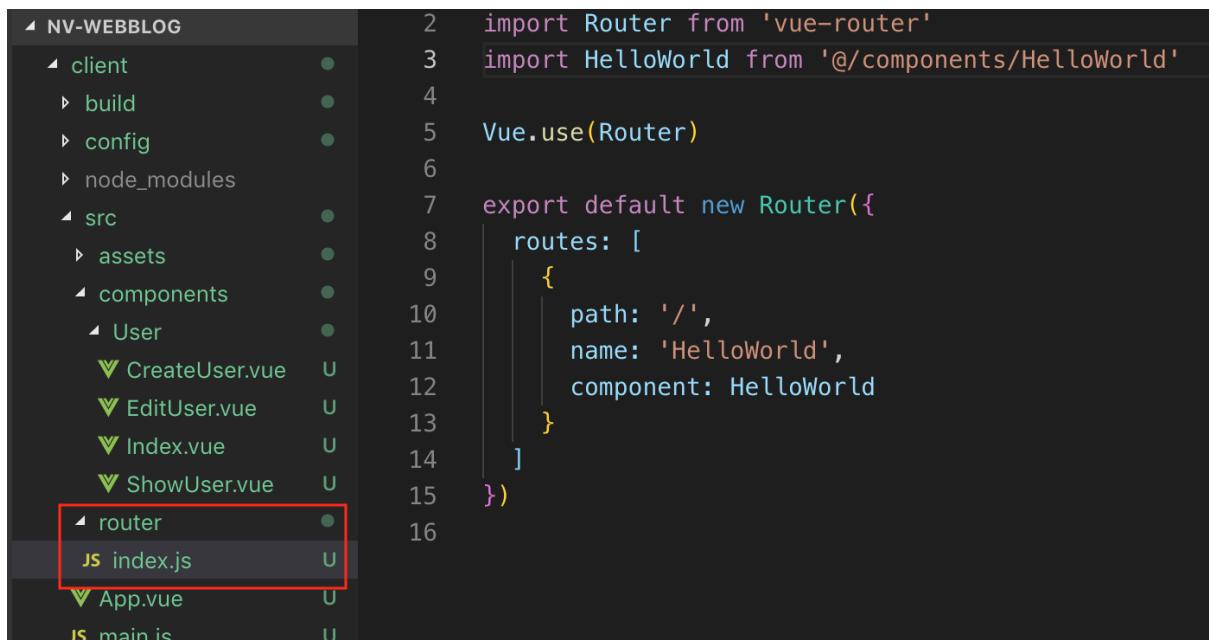
```
<template>
  <div>
    <h1>Get User By Id</h1>
  </div>
</template>
<script>
export default {

}

</script>
<style scoped>

</style>
```

เมื่อเราสร้าง Vue Component ในส่วนของ User ไว้เรียแล้ว ที่นี่เราจะมา Link ทุกอย่างเข้าด้วยกันครับ โดยทำการเปิด router/index.js ขึ้นมาครับ



```
2 import Router from 'vue-router'
3 import HelloWorld from '@/components/HelloWorld'
4
5 Vue.use(Router)
6
7 export default new Router({
8   routes: [
9     {
10       path: '/',
11       name: 'HelloWorld',
12       component: HelloWorld
13     }
14   ]
15 })
16
```

จากนั้นทำการแก้ไข code ที่ router/index.js เราเป็นดังนี้ครับ

```
import Vue from 'vue'
import Router from 'vue-router'

// Users
import UserIndex from '@/components/Users/Index'

Vue.use(Router)

export default new Router({
  routes: [
    {
      path: '/users',
      name: 'users',
      component: UserIndex
    }
  ]
})
```

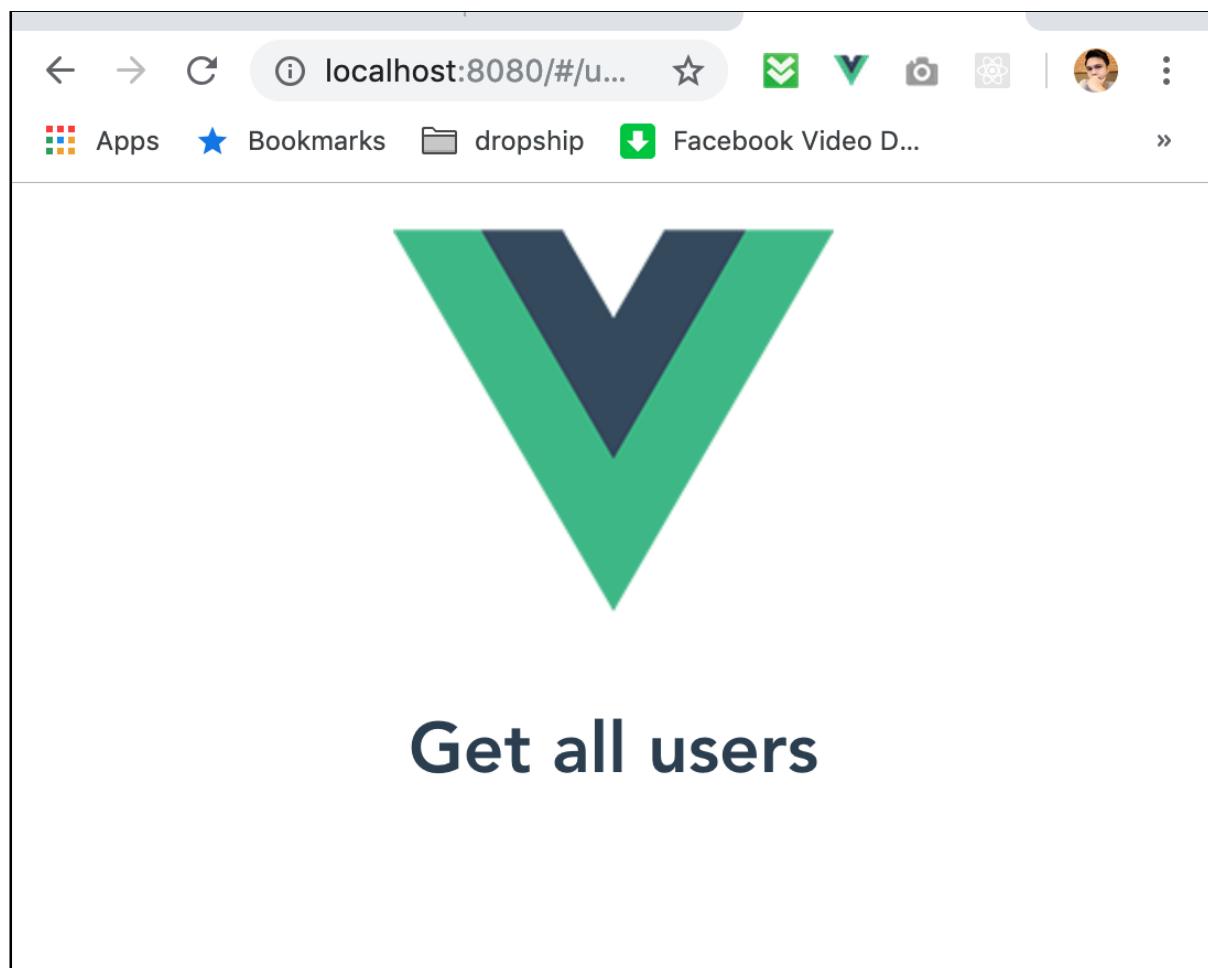
ทำการรัน client ของเราเลยครับ ไม่ลืมใช้มั้ยครับ เข้าไปที่ nv-webblog/client จากนั้น พิมพ์คำสั่ง

```
npm run dev
```

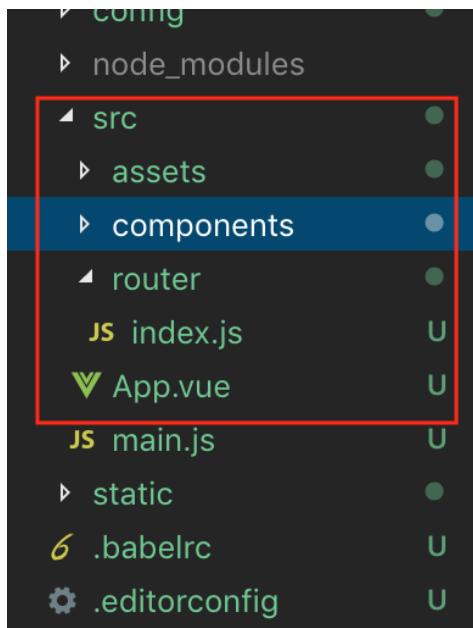
เมื่อ client หรือ front end ของผมทำงานแล้ว ผมเปิด Web Browser ของผมไปที่

```
http://localhost:8080/#/user
```

สังเกตว่าผมรัน Back End ของผมไว้ที่ port 8081 และ Front End ของผมที่ port 8080 ครับ เมื่อผมทำการใช้ Web Browser ของเปิดไปที่ url ด้านบนลิ้งที่ได้ คือ



ก่อนผมจะอธิบาย code ผมจะไปเจ้า Logo ออกก่อนนะครับ เห็นแล้วชอบหนุดใจ
 เพราะเราไม่ได้แสดงผลอย่างที่เราต้องการนั่นเอง เปิด src/App.vue ขึ้นมาครับ



```
6   </template>
7
8   <script>
9   export default {
10    name: 'App'
11  }
12  </script>
13
14  <style>
15  #app {
16    font-family: 'Avenir', He
17    -webkit-font-smoothing: a
```

จากนั้นผมจะลบ Logo และ style ที่เค้าให้มาที่อยากไม่ใช้ออกครับ เหลือไว้เพียง style ที่ผมชอบ โดยผมทำการแก้ code ดังนี้

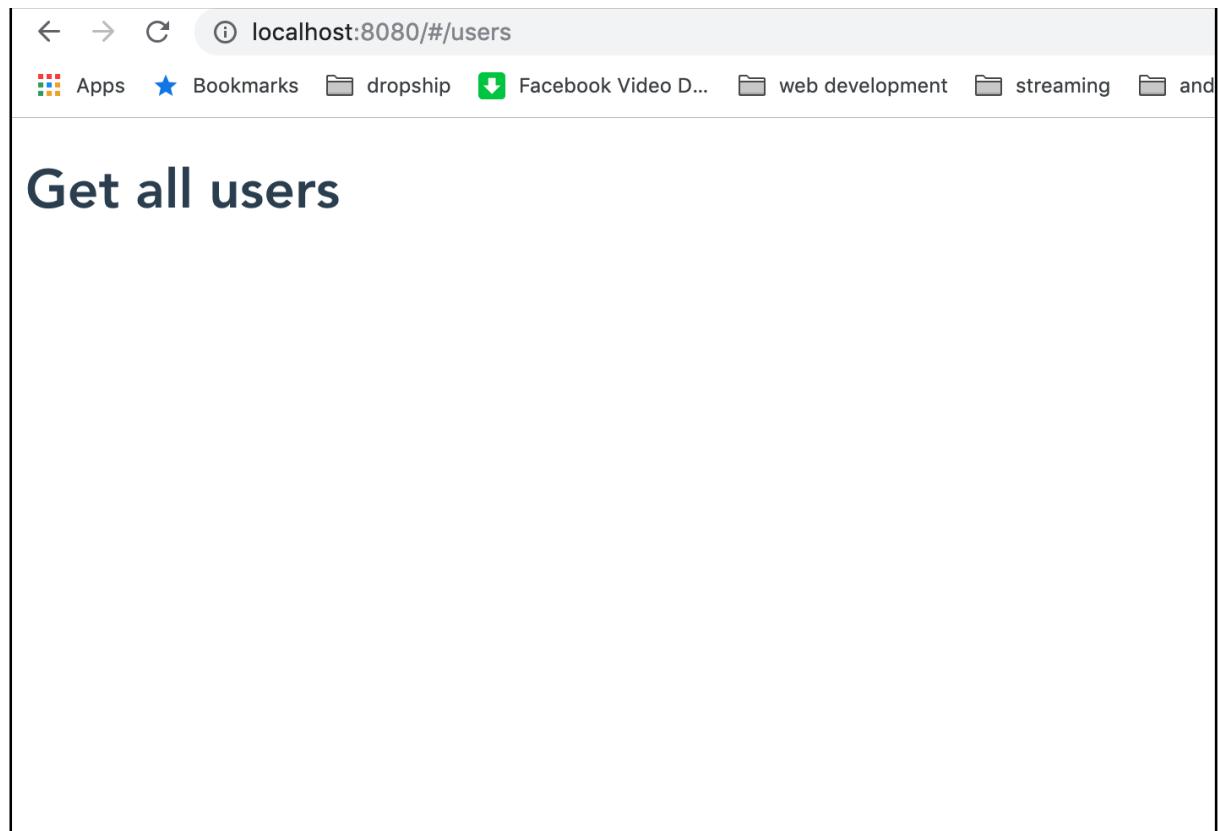
```
<template>
  <div id="app">
    <router-view/>
  </div>
</template>

<script>
export default {
  name: 'App'
}
</script>

<style>
#app {
  font-family: 'Avenir', Helvetica, Arial, sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  color: #2c3e50;
}
```

```
</style>
```

ผลการเปิดผ่าน Web Browser ของผมจะเป็นดังนี้ครับ



เปิด code ในส่วนของ router/index.js ขึ้นมาครับ จะมีสองส่วนที่เราต้องสนใจ คือ

```
import UserIndex from '@/components/Users/Index'
```

คือ โหลด component Users/Index.vue ของเรามาใช้งาน และ มาดูส่วนนี้กันครับ

```
{  
  path: '/users',  
  name: 'users',  
  component: UserIndex  
}
```

path หมายถึง ใช้ที่เรียกต่อจาก url หลัก ซึ่งในที่นี่คือ <http://localhost:8080> นั่นเอง แต่เวลาเรา Deploy หรือเราเอาขึ้น Server เราจะไม่แสดง Port ต่อท้ายนะครับ ไม่ต้องห่วง เช่น <http://sanook.com/users> จะเป็นทำนองนี้ครับ

name ก็คือชื่อ ตอนนี้เรายังไม่ได้ใช้ แค่รู้ว่าเอาไว้ อ้างอิงครับ เสร็จแล้ว เมื่อถูกเรียกมาตาม path แล้ว เรา ก็ส่ง component ของเราไปให้ครับ

สังเกตว่า ก่อนหน้านี้ เราไปลบ logo ออก มีผลกระทบเว็บไซต์ของเราทันทีเลย นั่นหมายความว่า App.vue มันทำงานร่วมด้วย

ถูกต้องครับ App.vue เป็นตัวแสดง Component ของเราผ่าน <router-view/> นั่นเองครับ

```
<router-view/>
```

เปิด App.vue ขึ้นมาประกอบนะครับ tag นี้ล่ะครับ จะเป็นตัวเลือกการแสดงผล แต่ละ Component จาก path ที่ส่งมา ซึ่งเค้าเรียกว่า vue-router แต่เราไม่ต้องเข้าไปลึกอะไรมันมาก รู้แค่ว่ามันแสดงผลตรงนี้ ส่ง component มาแสดงตรงนี้ตาม path ครับ

จากนั้น เราทำการใส่ path และ component ที่เราเตรียมไว้ทั้งหมดให้ครับ โดยแก้ไข code เป็น ดังนี้ครับ

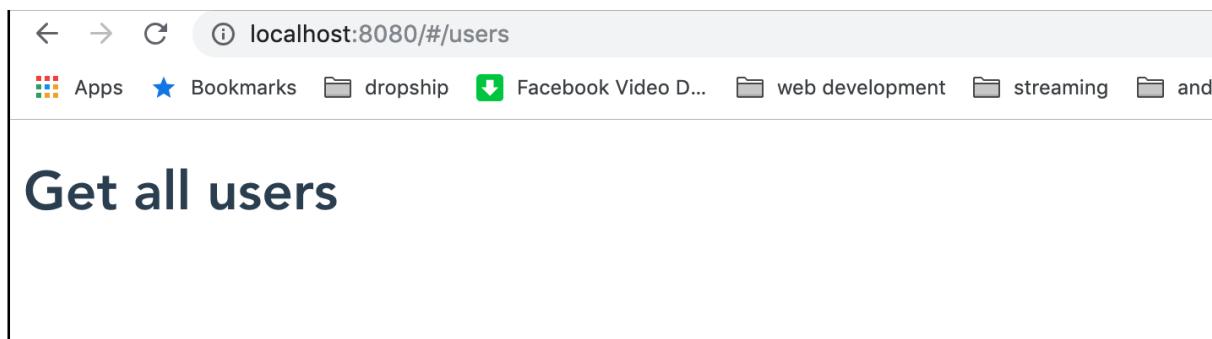
```
import Vue from 'vue'  
import Router from 'vue-router'  
  
// Users  
import UserIndex from '@/components/Users/Index'  
import UserCreate from '@/components/Users/CreateUser'
```

```
import UserEdit from '@/components/Users/EditUser'
import UserShow from '@/components/Users>ShowUser'

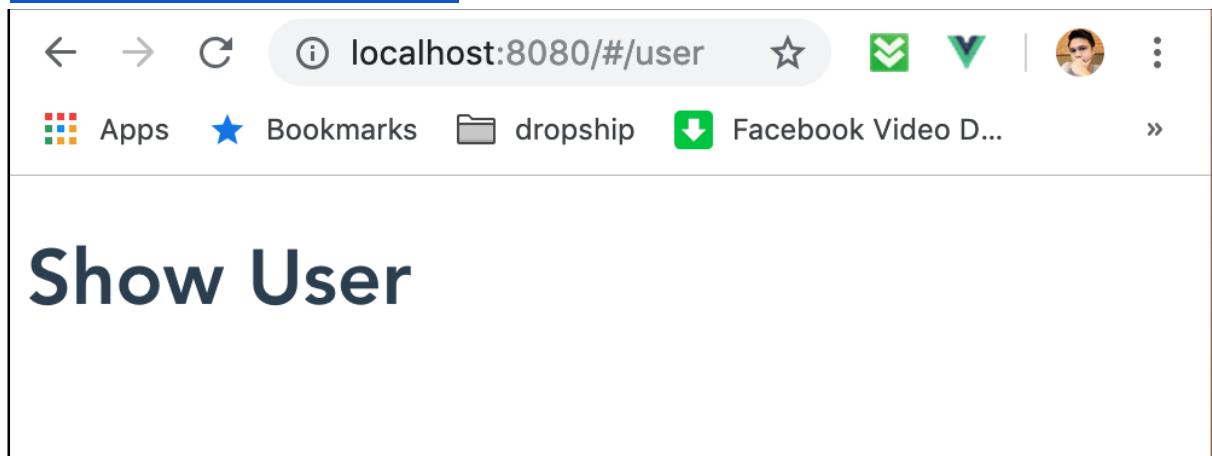
Vue.use(Router)

export default new Router({
  routes: [
    {
      path: '/users',
      name: 'users',
      component: UserIndex
    },
    {
      path: '/user/create',
      name: 'users-create',
      component: UserCreate
    },
    {
      path: '/user/edit',
      name: 'user-edit',
      component: UserEdit
    },
    {
      path: '/user',
      name: 'user',
      component: UserShow
    },
  ],
})
```

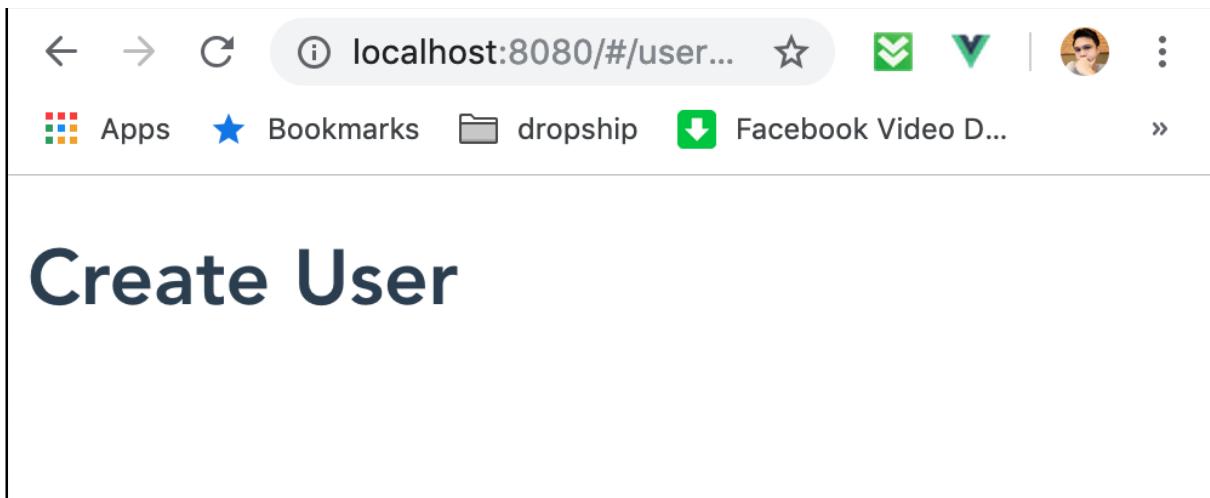
จากนั้นทำการทดสอบไปที่ URL ต่าง ๆ ดังนี้ จากตอบกลับมาตามรูปครับ
<http://localhost:8080/#/users>



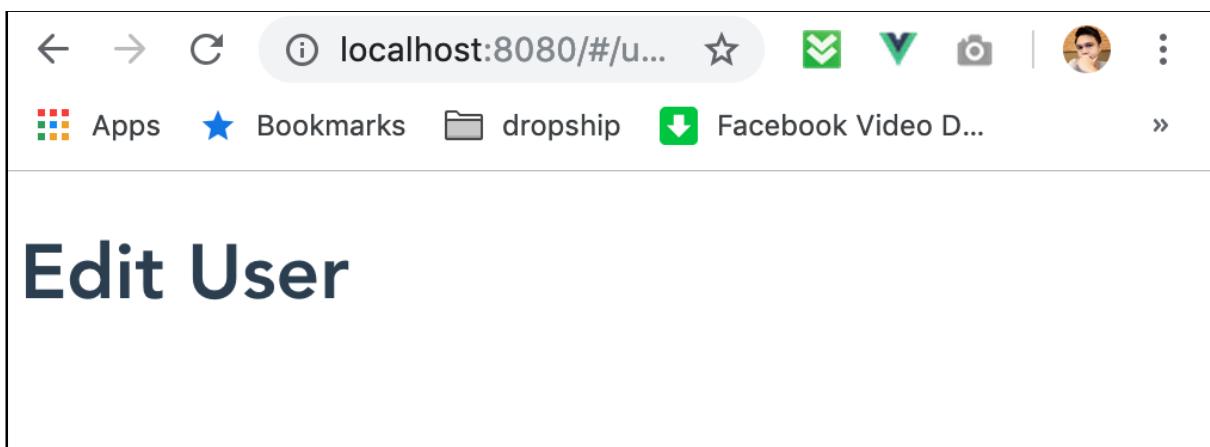
<http://localhost:8080/#/user>



<http://localhost:8080/#/user/create>



<http://localhost:8080/#/user/edit>



มาถึงตอนนี้เรา สามารถสร้าง Vue Component และ สามารถ เรียกใช้งานผ่าน url ที่เราต้องการ ได้เรียบร้อยแล้วนะครับ ในบทต่อไป เราจะดึงข้อมูล และสั่งงาน Server หรือ Back End ของเราราให้ทำงานคำสั่งนะครับ

บทที่ 8 ติดต่อกับ Server หรือ Back End ผ่าน Front End

ในบทที่ผ่านมาเราได้ทำ route ไป เพื่อจัดการ path หรือ url ของเราให้ไปดึง Vue Component ของมาทำงานได้อย่างถูกต้องแล้วนะครับ ตอนนี้เราจะมาสร้าง Service เพื่อติดต่อกับ Server หรือ Back End ของเรา กัน จะริงแล้วเรารสามารถสร้างการติดต่อกับ Server ไว้ใน Vue Component ของเราเลยก็ได้ แต่เนื่องจาก การทำงานจริงมี Component จำนวนมาก เราสร้าง Service ไว้ให้ Component ของเราใช้งานร่วมกันจะดีกว่าครับ

ทำความรู้จัก Vue-Resource

การติดต่อผ่าน HTTP นั้น จะมี package ตัวนึงให้เราใช้งาน ชื่อว่า Vue Resource คับ เราต้องทำการติดตั้งก่อนครับ โดย หยุดการทำงานของ client ของเราก่อนด้วยการกด ^c ที่ Terminal ที่ client ของเราใช้งาน จากนั้น พิมพ์คำสั่ง เพื่อติดตั้ง Vue-Resource ของเราดังนี้

```
npm install --save vue-resource
```

The screenshot shows a terminal window with the following output:

```
client — npm TERM_PROGRAM=Apple_Terminal TERM=xterm-256color SHELL=...  
...book/nv-webblog/server — node src/app.js ...s/lq3mt_sj42l1pnxqrgsjt_1h0000gn/T/  
DONE Compiled successfully in 211ms  
I Your application is running here: http://localhost:8080  
http://localhost:8080/#/user/creat  
^C192:client kornchayapon$  
192:client kornchayapon$ ls  
README.md index.html package.json  
build node_modules src  
config package-lock.json static  
192:client kornchayapon$ npm install --save vue-resource  
(( )) .. fetchMetadata: still resolveWithNewModule got@8.3.2 check
```

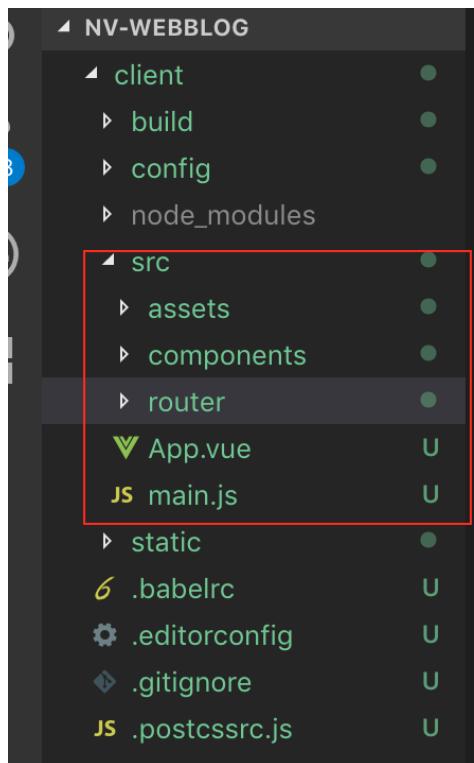
เมื่อทำการติดตั้งแล้ว ไปดูที่ package.json ในส่วนของ client ด้วยนะครับ ว่า ติดตั้งสำเร็จแล้วจริง จะมีบรรทัดเพิ่มขึ้นมาใน "dependencies" ครับ

```
"vue-resource": "^1.5.1",
```

ติดตั้งเสร็จแล้วอย่าลืมรันโปรแกรมฝั่ง client ของเราเหมือนเดิมด้วยนะครับ

การใช้งาน npm package ใน vuejs

ตอนนี้เราติดตั้ง Vue Resource ใน Project ของเราแล้ว เราจะเรียกใช้งานมันได้ เราต้องไปเรียกใช้มันก่อนครับ ทำการเปิดไฟล์ src/main.js ขึ้นมากรับ แล้วทำการเพิ่ม code ดังนี้ครับ



```

2 // (runtime-only or standalone) h
3 import Vue from 'vue'
4 import App from './App'
5 import router from './router'
6 import VueResource from 'vue-reso
7
8 Vue.config.productionTip = false
9
10 Vue.use(VueResource)
11
12 /* eslint-disable no-new */
13 new Vue({
14   el: '#app',
15   router,
16   components: { App },
17   template: '<App/>'
18 })

```

```

import Vue from 'vue'
import App from './App'
import router from './router'
import VueResource from 'vue-resource'

Vue.config.productionTip = false

Vue.use(VueResource)

/* eslint-disable no-new */
new Vue({
  el: '#app',
  router,
  components: { App },
  template: '<App/>'
})

```

ชี้งจริง ๆ ส่วนที่ผิด เพิ่มเข้ามา คือ

```
import VueResource from 'vue-resource'
```

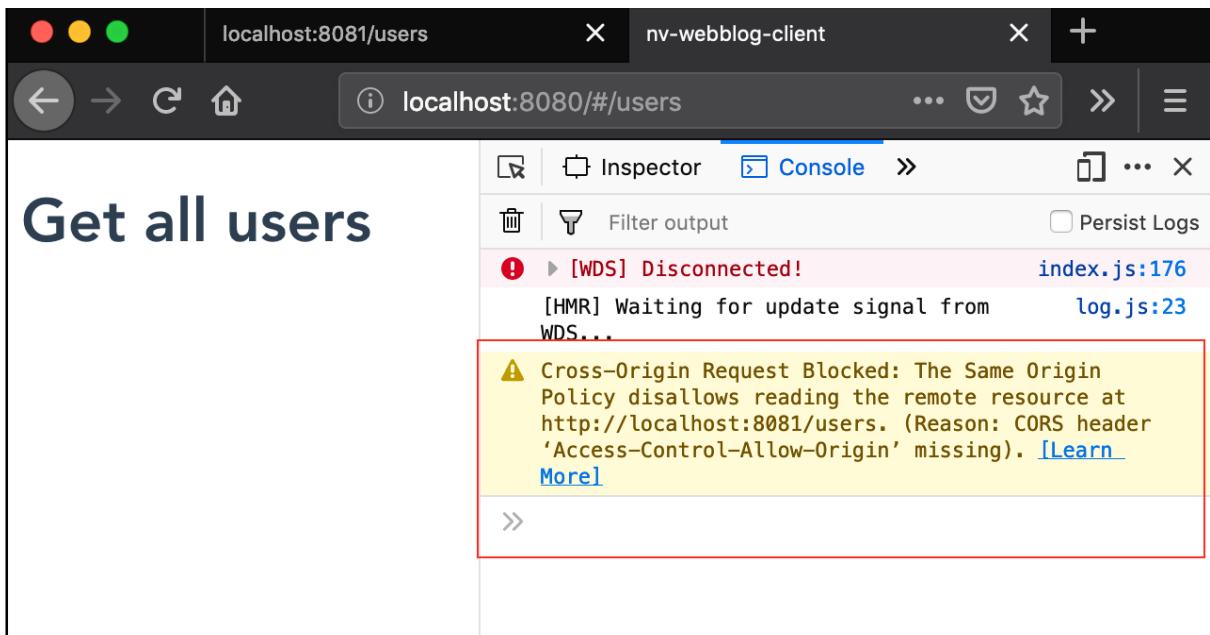
```
Vue.use(VueResource)
```

เพื่อทำให้ vuejs รู้จัก vue-resource ของเราในเองครับ จากนั้นผมจะทำการทดสอบ โดยทำการดึงข้อมูลจาก Server ของเรา กันครับ

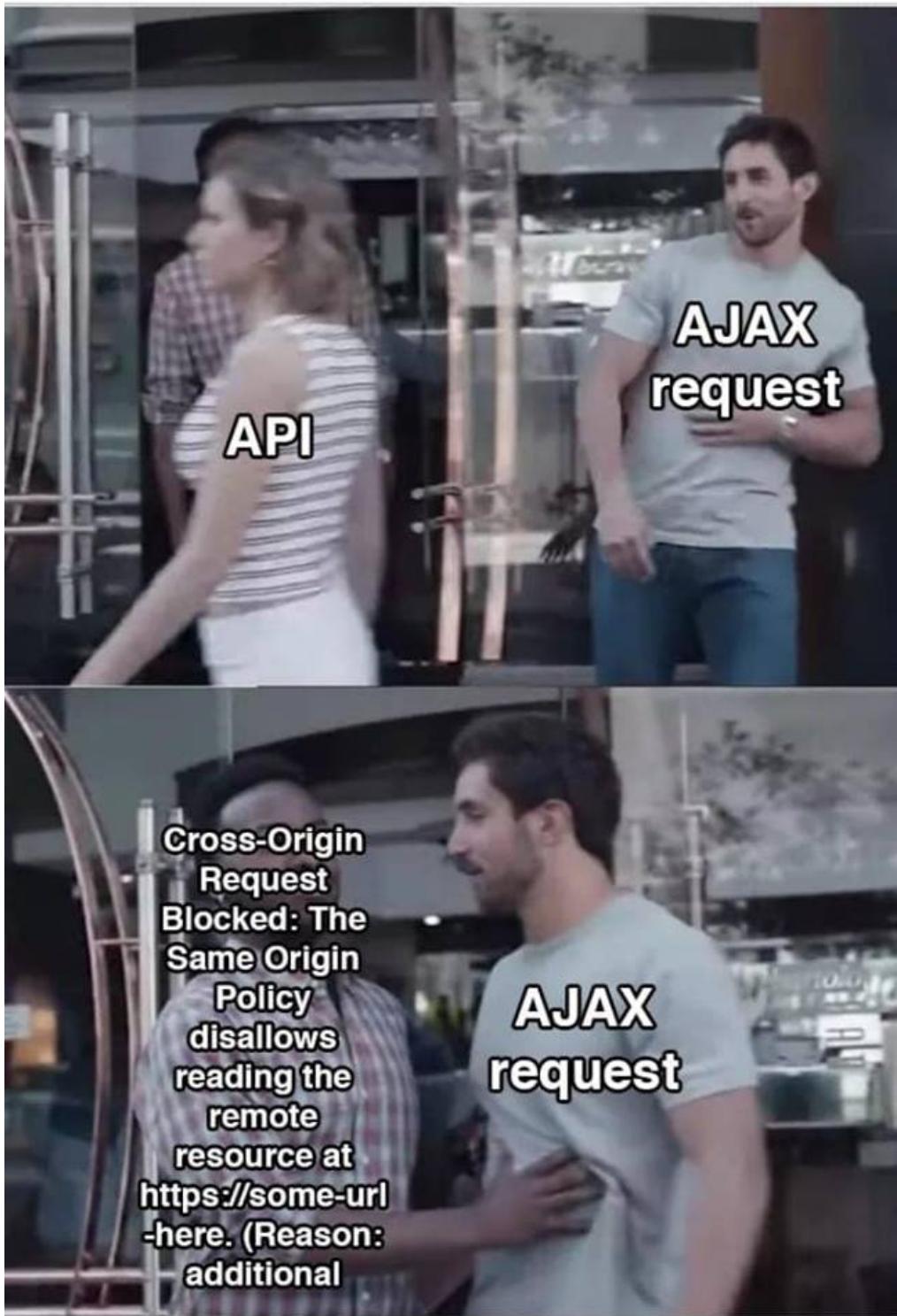
เปิด src/components/Users/Index.vue ขึ้นมาครับ จากนั้นทำการเพิ่ม code นี้ เข้าไปใน tag ของ script ครับ

```
<script>
export default {
  created () {
    this.$http.get('http://localhost:8081/users')
      .then(function (response) {
        console.log(response)
      })
  }
}
</script>
```

จากนั้นทำการเปิด web browser เพื่อดูสิ่งที่เรา log มาจาก server ครับ



จริง ๆ มันต้อง log และแสดงผลลัพธ์ เป็น json ที่เราส่งมาสิครับ ทำไม่ถึงไม่มีอะไร
ออกมาก็มันแจ้งว่า Cross-Origin คือ เราไปดึงข้อมูลจาก server ชาวบ้านมานั่นเอง
ซึ่งเค้าไม่ยอมครับ



นี่ครับ Cross-Origin ทำงานอย่างนี้ครับ

ทำความรู้จัก Cors เพื่อจัดการปัญหาเรื่อง Cross Origin

จากหัวข้อที่ผ่าน server ของเราไม่ยอมให้ข้อมูลตามที่เราต้องขอไป เพราะว่ามันไม่ได้อยู่ server หรือ root domain เดียวกัน เราจะไปปลดล็อกนี้ด้วย npm package ที่ชื่อว่า cors ครับ ทำให้ไครก์สามารถเข้ามาดึงข้อมูลของเราได้หมด พอกดอย่างนี้ นักเรียนผมตกใจกันทุกคน เพราะ server ของเราจะไม่ปลอดภัย

ไม่ต้องกังวลเรื่องนั้น เราจะจัดการการเข้าถึงข้อมูลแต่ละส่วนด้วย Permission ในบทต่อ ๆ ไปครับ

ทำการปิด server ของเรา ก่อน ด้วย ^C ที่ Terminal ของเรา น้อยครับ แล้วทำการติดตั้ง cors ด้วย คำสั่ง

```
npm install --save cors
```

The screenshot shows a terminal window titled "server — -bash — 80x24". The window has three tabs at the top: "server — -bash", "...RM=xterm-256color SHELL=/bin/bash", and "+". The main pane of the terminal displays the following command-line session:

```
~/noderootbook/nv-webblog/server — -bash
...RM=xterm-256color SHELL=/bin/bash
LL, `updatedAt` DATETIME NOT NULL);
Executing (default): PRAGMA INDEX_LIST(`Users')
Server running on 8081
Executing (default): SELECT `id`, `email`, `password`, `name`, `lastname`, `status`, `createdAt`, `updatedAt` FROM `Users` AS `User`;
Executing (default): SELECT `id`, `email`, `password`, `name`, `lastname`, `status`, `createdAt`, `updatedAt` FROM `Users` AS `User`;
Executing (default): SELECT `id`, `email`, `password`, `name`, `lastname`, `status`, `createdAt`, `updatedAt` FROM `Users` AS `User`;
Executing (default): SELECT `id`, `email`, `password`, `name`, `lastname`, `status`, `createdAt`, `updatedAt` FROM `Users` AS `User`;
^C
[192:server kornchayapon$]
[192:server kornchayapon$]
[192:server kornchayapon$]
[192:server kornchayapon$]
[192:server kornchayapon$]
[192:server kornchayapon$]
[192:server kornchayapon$] npm install --save cors
[192:server kornchayapon$] npm WARN nv-webblog-server@1.0.0 No repository field.
+ cors@2.8.5
added 1 package in 1.992s
[192:server kornchayapon$]
```

ผลการติดตั้ง cors package

จากนั้นทำการเปิด src/app.js ใน server ของเราขึ้นมา แล้วทำการเพิ่ม code ดังนี้ เข้าไปครับ

The screenshot shows the VS Code interface. On the left, the Explorer sidebar displays a project structure for 'NV-WEBBLOG'. It includes a 'client' folder, a 'server' folder (which is selected and highlighted with a red border), and a 'src' folder containing 'config', 'controllers', and 'models' subfolders. Inside 'src', there are two JavaScript files: 'app.js' and 'routes.js', both of which are also highlighted with red borders. Other files shown include 'package-lock.json' and 'package.json'. On the right, the main editor tab is titled 'app.js' and contains the following code:

```
1 let express = require('express')
2 let bodyParser = require('body-parser')
3 let cors = require('cors')
4 const {sequelize} = require('../models')
5
6 const app = express()
7
8 app.use(bodyParser.json())
9 app.use(bodyParser.urlencoded({extended: true}))
10 app.use(cors())
11
12
13
14 require('../routes')(app)
15
16 app.get('/status', function (req, res ){
17   res.send('Hello nodejs server')
18 })
19
```

code ที่เราเพิ่มไปใน src/app.js

```
let cors = require('cors')

app.use(cors())
```

```
let express = require('express')
let bodyParser = require('body-parser')
let cors = require('cors')
const {sequelize} = require('./models')

const app = express()

app.use(bodyParser.json())
app.use(bodyParser.urlencoded({extended: true}))
app.use(cors())

require('./routes')(app)

app.get('/status', function (req, res ){
  res.send(['Hello nodejs server'])
})
```

จากนั้นทำการรัน server ของเราอีกครั้งครับ ถ้าไครผลลัพธ์ client ไป ก็ทำการรันด้วยนะครับ จากนั้นเปิด web browser ของเรา เรียกไปที่ url

```
http://localhost:8080/#/users
```

The screenshot shows a browser window with the title "nv-webblog-client" and the URL "localhost:8081/users". The browser's address bar also displays "localhost:8080/#/users". The developer tools are open, specifically the "Console" tab. The log output shows the following:

```
[WDS] Disconnected! index.js:176
[HMR] Waiting for update signal from log.js:23
WDS...
created Index.vue:10
res:[object Object] Index.vue:9
▶ Object { url: "http://localhost:8081/users", ok: true, status: 200, statusText: "OK", headers: {...}, body: (2) [...], bodyText: "[{"id":1,"email": "peter@gmail.com","password": "123456","name": "peter","lastname": "parker"}, {"id":2,"email": "korn.chayapon@gmail.com","password": "123456","name": "korn","lastname": "chayapon"}, {"status": "active"}, {"createdAt": "2019-03-05T13:40:13.802Z"}, {"updatedAt": "2019-03-05T13:40:13.802Z"}], {"id":3,"email": "korn.chayapon@gmail.com","password": "123456","name": "korn","lastname": "chayapon"}, {"status": "active"}, {"createdAt": "2019-03-05T13:43:56.297Z"}, {"updatedAt": "2019-03-05T13:43:56.297Z"}]" }"
```

จาก log ที่เรา log ออกมามาแสดงให้เห็นว่า ตอนนี้เราดึงข้อมูลจาก server ของเราได้แล้วนะครับ

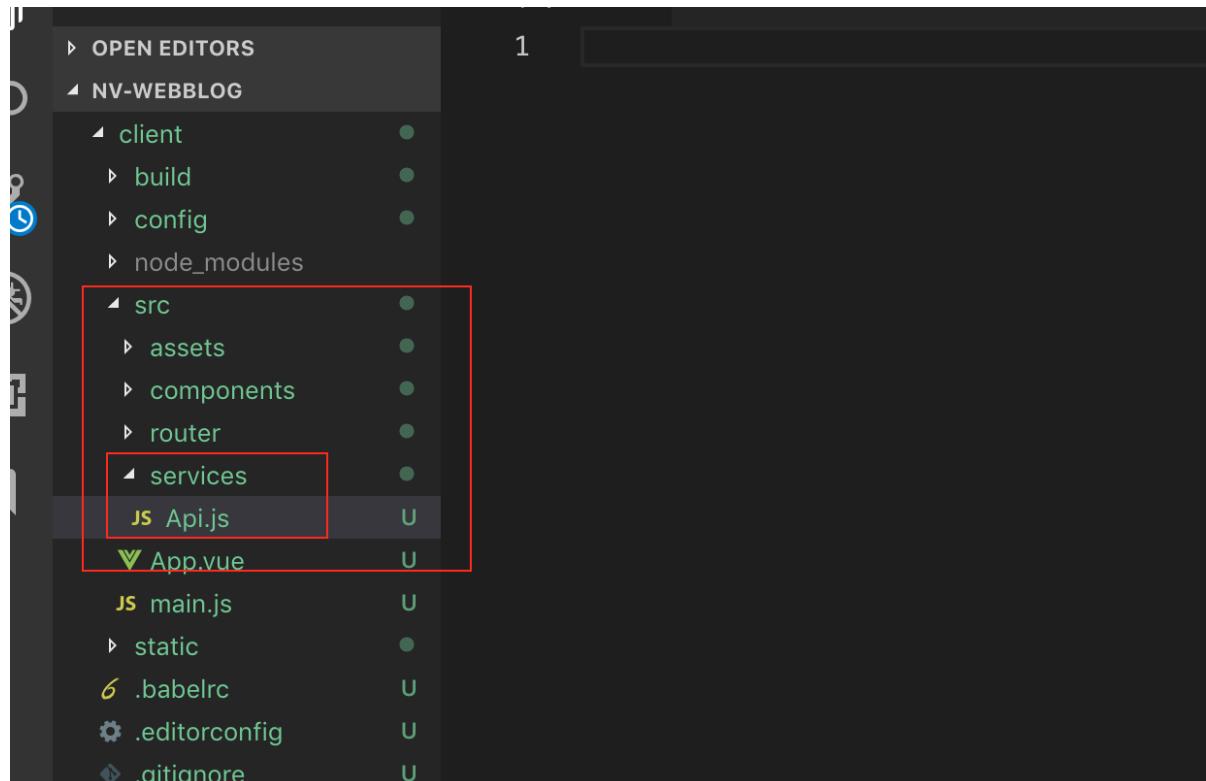
เริ่มต้นทำ Service API ไว้ใช้งาน

จากหัวข้อที่ผ่านมา เราได้ใช้ vue-resource ดึงข้อมูลจาก server ของเราได้แล้ว แต่การใช้งานจริง ที่เรากำลังจะทำนั้น เราจะใช้ package ที่ชื่อ axios มาใช้งาน ครับ เพราะจะทำให้เราเขียน code โปรแกรมในส่วนนี้น้อยลงมาก ช่วยลดความผิดพลาดได้เยอะครับ

ทำการ down client ของเรางอก่อนครับ จากนั้นทำการติดตั้ง axios package ด้วยคำสั่ง

```
npm install --save axios
```

จากนั้นทำการสร้างโฟล์เดอร์ชื่อว่า services ไว้ใน src ของเรา จากทำการสร้างไฟล์ src/services/Api.js ครับ



ทำการเปิดไฟล์ Api.js ขึ้นมา และทำการใส่ code ดังนี้ครับ

```
import axios from 'axios'

export default () => {
  return axios.create({
    baseURL: 'http://localhost:8081/',
  })
}
```

จาก code จะเป็นการ เรียก axios เพื่อใช้งานและ export เพื่อให้ไฟล์โปรแกรมอื่นสามารถเรียกใช้งานด้วยเซ่นกัน โดยเรากำหนด baseURL เป็น URL หลังของ server ของเรา คือ

```
http://localhost:8081/
```

ทำการสร้างไฟล์

```
src/services/UsersService.js
```

จากนั้นทำการใส่ code ดังนี้

```
import Api from '@/services/Api'

export default {
  index (search) {
    return Api().get('users')
  },
  show (userId) {
    return Api().get('user/' + userId)
  },
  post (user) {
    return Api().post('user', user)
  },
  put (user) {
    return Api().put('user/' + user.id, user)
  },
  delete (user) {
    return Api().delete('user/' + user.id, user)
  },
}
```

จาก code ด้านบนผมทำการ import Api.js เพื่อใช้งานผ่าน Api() ซึ่งการใช้งานลักษณะนี้ จะทำการใช้งาน HTTP Method (get, post, put, delete) ได้ง่าย โดยผมทำฟังก์ชันต่าง ๆ แล้วทำการ export เพื่อให้ไฟล์โปรแกรมอื่น ๆ ของผม

สามารถเรียกใช้งานฟังก์ชันต่าง ๆ ได้นั่นเอง เช่นเมื่อเรียกผ่าน function index ของแล้วก็จะทำการ return ค่าที่ได้จาก server ตาม url ที่ส่งไปนั่นเองครับ

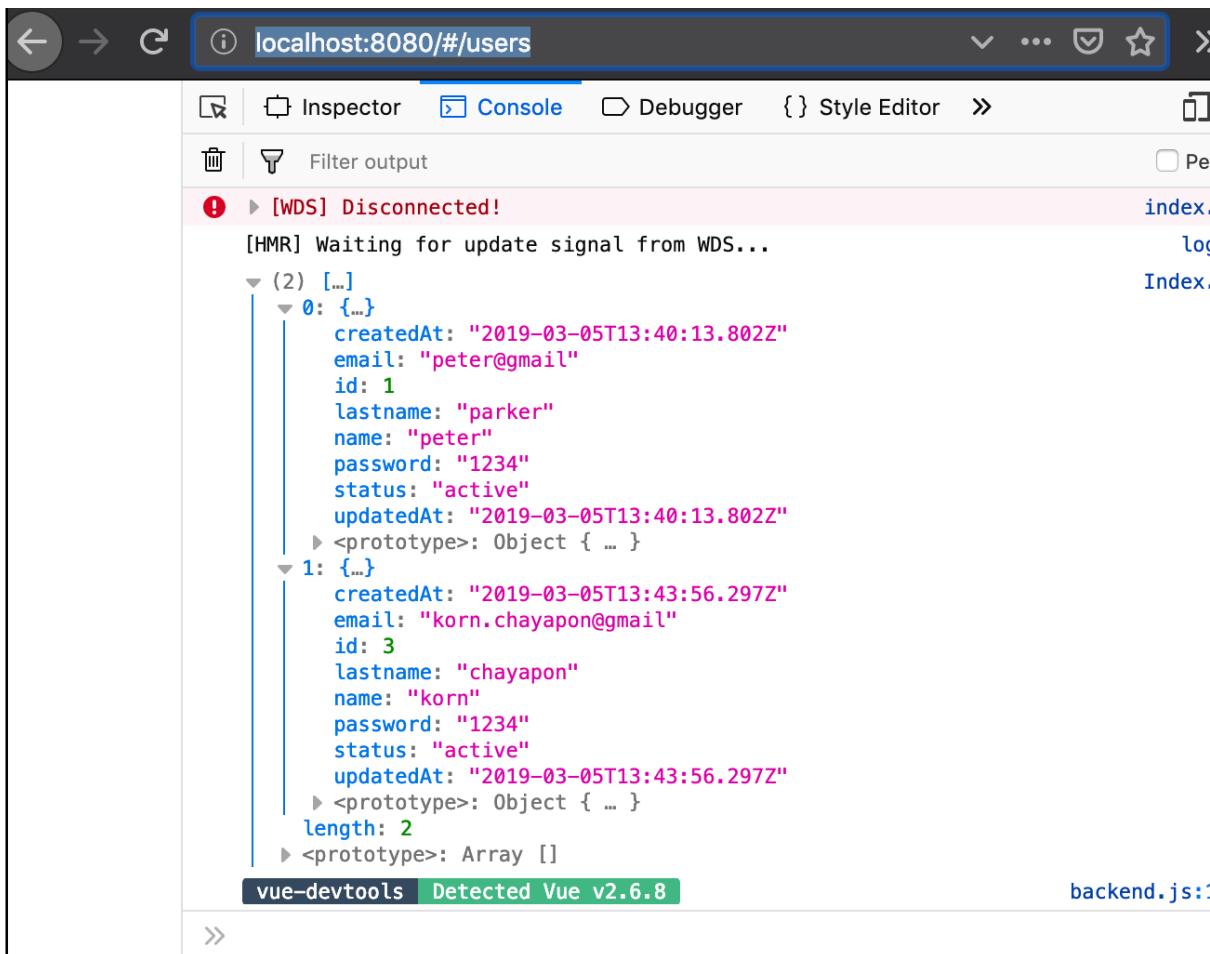
ทำการเรียกใช้งาน Service ผ่าน Vue Component

ทำการเปิด src/components/Users/Index.vue ในส่วนของ client ของเราขึ้นมาแล้วทำการแก้ไข code เป็นดังนี้ครับ

```
<template>
  <div>
    <h1>Get All Users</h1>
  </div>
</template>
<script>
import UserService from '@/services/UsersService'

export default {
  async created () {
    let results = (await UserService.index()).data
    console.log(results)
  }
}
</script>
<style scoped>
</style>
```

เมื่อผมการ log data จากการเรียก url <http://localhost:8080/#/users> ผ่าน Web Brower ของผมจะสังเกตุได้ว่า มีการตีงข้อมูล json ออกมาได้แล้วนั่นเอง



ผลการดึงข้อมูลจาก server ผ่าน service

จาก code โปรแกรมของผมที่ผ่านมา มีจุดที่น่าสนใจคือ await และ async คือเนื่องจากการเขียนโปรแกรมด้วย javascript จะเป็นแบบ async ตามที่เคยกล่าวมา แต่การอ่านข้อมูลจาก server นั้น ใช้เวลาในการอ่าน หรือเขียน ทำให้โปรแกรมของเราต้องหยุดรอ เพื่อให้ได้ข้อมูลครบถ้วน ก่อน จึงไปทำงานส่วนอื่นๆนั่นเอง จำไว้ว่า await ว่าไว้หน้า statement หรือคำสั่งที่รอ ส่วน async วางไว้หน้า function ครับ

ใน code โปรแกรมส่วนนี้จะมีส่วนสำคัญอีกส่วน ที่ผมยังไม่ได้อธิบาย คือ

```
async created () {
```

ตอนนี้รู้ว่า เมื่อ Vue Component ของเราถูกเรียกใช้งาน มันจะทำงานที่ function created ก่อนเสมอครับ မจะไปอธิบายเรื่อง Lifecycle อีกครั้งในหัวข้อต่อๆ ไปครับ

แสดงผลข้อมูลผ่าน Vue Component

มาถึงตอนนี้ ผมจะนำค่าที่อ่านจาก server มาแสดงผลบนหน้า page ของเรานะครับ ผมทำการแก้ code ใน src/components/Users/Index.vue ของเราเป็นดังนี้ครับ

```
<template>
  <div>
    <h1>Get All Users</h1>
    <div v-if="users.length">
      <div>จำนวนผู้ใช้งาน {{ users.length }}</div>
      <div>id: {{ users[0].id }}</div>
      <div>ชื่อ-นามสกุล: {{ users[0].name }} - {{ users[0].lastname }}</div>
      <div>email: {{ users[0].email }}</div>
      <div>password: {{ users[0].password }}</div>
    </div>
  </div>
</template>
<script>
import UserService from '@/services/UserService'

export default {
  data () {
    return {
      users: []
    }
  },
  async created () {
    this.users = (await UserService.index()).data
  }
}
</script>
<style scoped>
</style>
```

ผมจะยกถึง code ที่นำเสนอใจดังนี้ครับ

```
<div v-if="users.length">
```

เริ่มต้นจาก ผมใช้ v-if ซึ่งจะทำงานเหมือน if else ทั่วไปนี่ล่ะครับ เพียงแต่เราเอาไปใส่ใน tag html ต่าง ๆ เพื่อกำหนดว่า tag นี้จะแสดงหรือไม่แสดงตามเงื่อนไขของ if นั้นเอง กรณีนี้ผมบอกว่า ถ้าไม่มีข้อมูลก็ไม่ต้องแสดงครับ

```
{{ users.length }}
```

ส่วนนี้จะเป็นการนำค่าใน data() ที่ชื่อว่า users ออกมาระบุ ผ่าน tag {{}} ที่ต้องจำคือ ถ้า users อยู่ใน tag script เราต้องใช้คำว่า this.users เสมอ แต่ถ้าอยู่ใน tag template หรือ html ของเรายังคงใช้ users ครับ

```
data () {
  return {
    users: []
  }
},
```

data () คือ ที่เก็บตัวแปรของเรานั้นเอง การ return เป็น format ของ vuejs ครับ ต้องใช้ตาม format หรือ syntax นี้เท่านั้น พลิกแพลงอาจทำได้บ้างแต่จะ error ได้ง่ายนะครับ

```
users: []
```

บอกว่าเป็นตัวแบบ array นั้นเอง

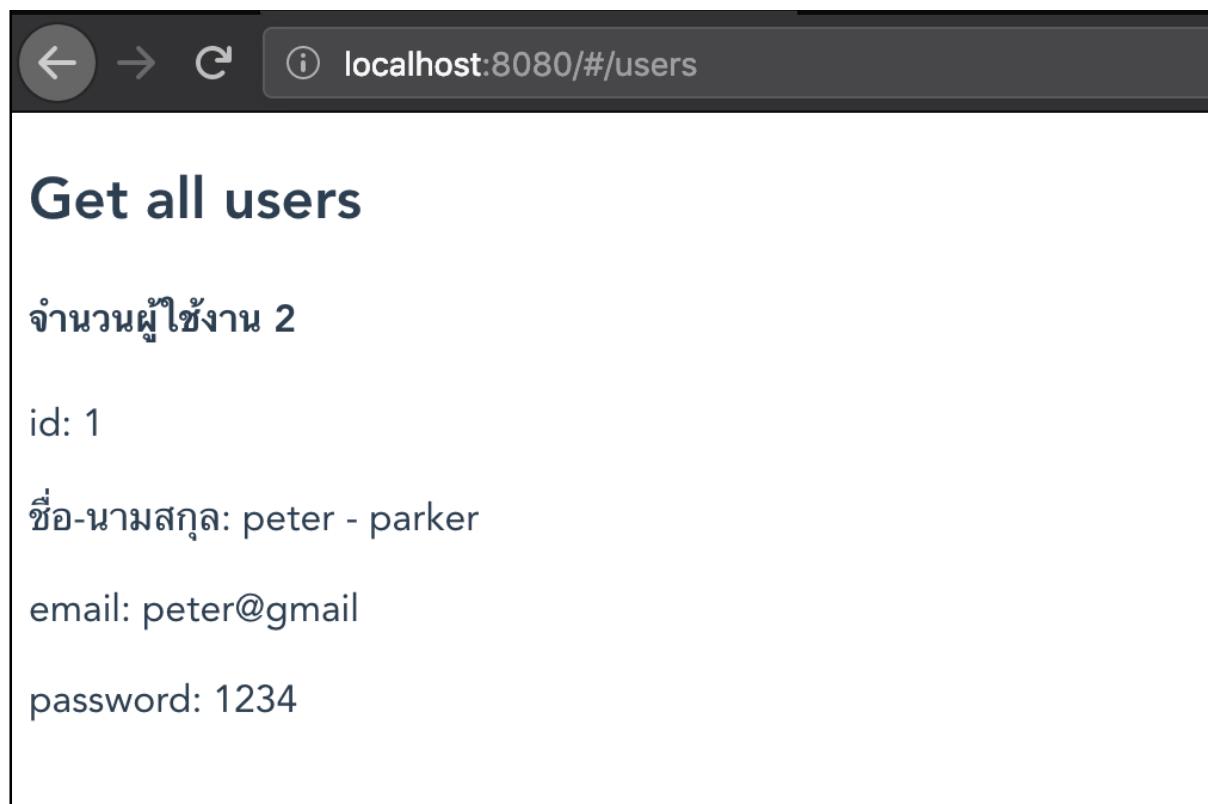
```
async created () {
  this.users = (await UserService.index()).data
  console.log(results)
}
```

อ่านค่าจาก server มาเก็บไว้ใน users ซึ่งเป็นตัวแปรแบบ array ของเรานั้นเอง

```
<div>
  <div>จำนวนผู้ใช้งาน {{ users.length }}</div>
  <div>id: {{ users[0].id }}</div>
  <div>ชื่อ-นามสกุล: {{ users[0].name }} - {{ users[0].lastname }}</div>
  <div>email: {{ users[0].email }}</div>
  <div>password: {{ users[0].password }}</div>
</div>
```

แสดงผล user คนแรกในระบบ หรือ index = 0 นั่นเอง อ่านค่าความยาวของ array ได้โดยตรงจาก users.length ได้เลย

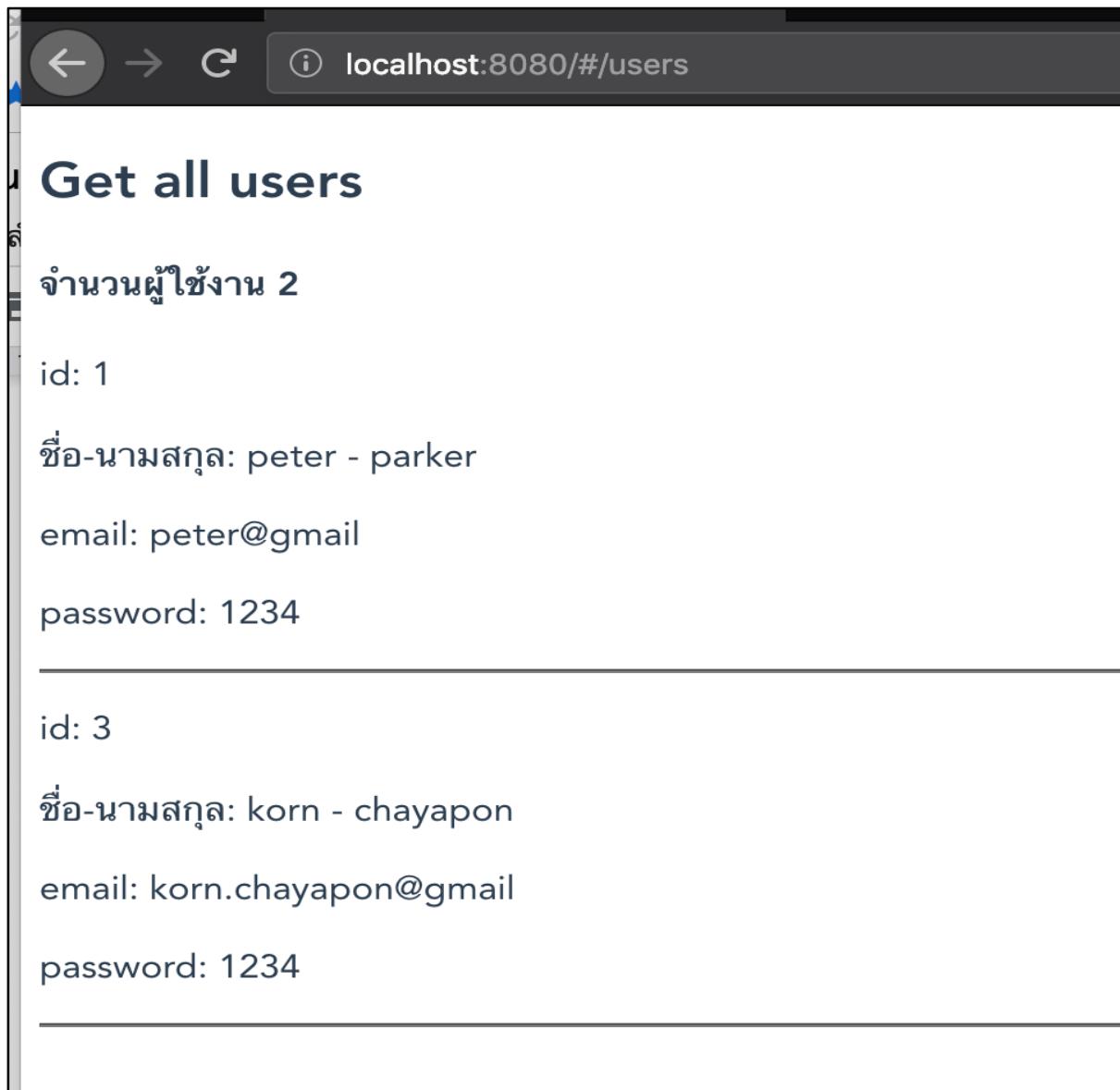
ผลการทำงานจะเป็นดังนี้ครับ



ต่อมาเราจะแก้ code เพิ่มนิดหน่อย เพื่อแสดงผล Users ของเราทั้งหมดด้วย v-for กันครับ เราทำการแก้ code โปรแกรมของเราเป็นดังนี้ครับ

```
<template>
<div>
  <h2>Get all users</h2>
  <h4>จำนวนผู้ใช้งาน {{users.length}}</h4>
  <div v-for="user in users" v-bind:key="user.id">
    <p>id: {{ user.id }}</p>
    <p>ชื่อ-นามสกุล: {{ user.name }} - {{ user.lastname }}</p>
    <p>email: {{ user.email }}</p>
    <p>password: {{ user.password }}</p>
  </div>
</div>
</template>
```

ผลจากโปรแกรมที่เราเขียนจะเป็นดังนี้



ซึ่งเป็นการใช้ v-for ในการแสดงผลข้อมูลผู้ใช้งานของเราทั้งหมดครับ ในตอนนี้ เราสามารถแสดงผลผู้ใช้งานทั้งหมดของเราได้แล้ว โดยการใช้งาน v-for นั้นก็ เหมือนการใช้ loop for ธรรมดานั้นเองครับ

จัดการลีนทาง ด้วย this.\$router

ตอนนี้เราจะเข้าไปดู user แต่ละคนของเรากันครับ ผมจะทำปุ่มดู user ก่อนเลย ครับ โดยเพิ่ม button หนึ่งตัวครับ

```
<div>
  <h2>Get all users</h2>
  <h4>จำนวนผู้ใช้งาน {{users.length}}</h4>
  <div v-for="user in users" v-bind:key="user.id">
    <p>id: {{ user.id }}</p>
    <p>ชื่อ-นามสกุล: {{ user.name }} - {{ user.lastname }}</p>
    <p>email: {{ user.email }}</p>
    <p>password: {{ user.password }}</p>
    <p><button>ดูข้อมูลผู้ใช้</button></p>
    <hr>
  </div>
</div>
```

ตอนนี้ผมทำการบูม เพื่อดูข้อมูลผู้ใช้งานเซิร์ฟแล้ว คือ เพิ่ม code นี้เข้าไปนั้นเอง

```
<p><button>ดูข้อมูลผู้ใช้</button></p>
```

จากนั้นผมจะทำการสร้าง method ให้กับ Vue Component ของเรา กันครับ โดย method นี้จะเป็นการเรียก Vue Component ที่ชื่อว่า components/Users>ShowUser.vue มา Render แทน components/Users/Index.vue กันครับ โดยผมทำการเพิ่ม code เข้าไปใน tag script ใน components/Users/Index.vue ของผมดังนี้

```
<script>
  import UserService from '@/services/UserService'

  export default {
```

```

data () {
  return {
    users: []
  },
  async created () {
    this.users = (await UserService.index()).data
  },
  methods: {
    navigateTo (route) {
      this.$router.push(route)
    },
  }
}
</script>

```

ชี้งส่วนที่ผມเพิ่มเข้าไป คือ

```

methods: {
  navigateTo (route) {
    this.$router.push(route)
  },
}

```

นั่นเองครับ โดย code ชุดนี้ เมื่อเราเรียกใช้ เราจะส่ง path ที่เราต้องการให้วิ่งไป ผ่านตัวแปร route นั่นเอง

โดย `this.$router.push(route)` จะทำการเรียกใช้ได้เลย เพราะ ติดตั้งมากจาก vue-clie ให้เรียบร้อยแล้วครับ จากนั้นผມแก้ไข code ที่ ปุ่มดูข้อมูลผู้ใช้ของผມ เป็น

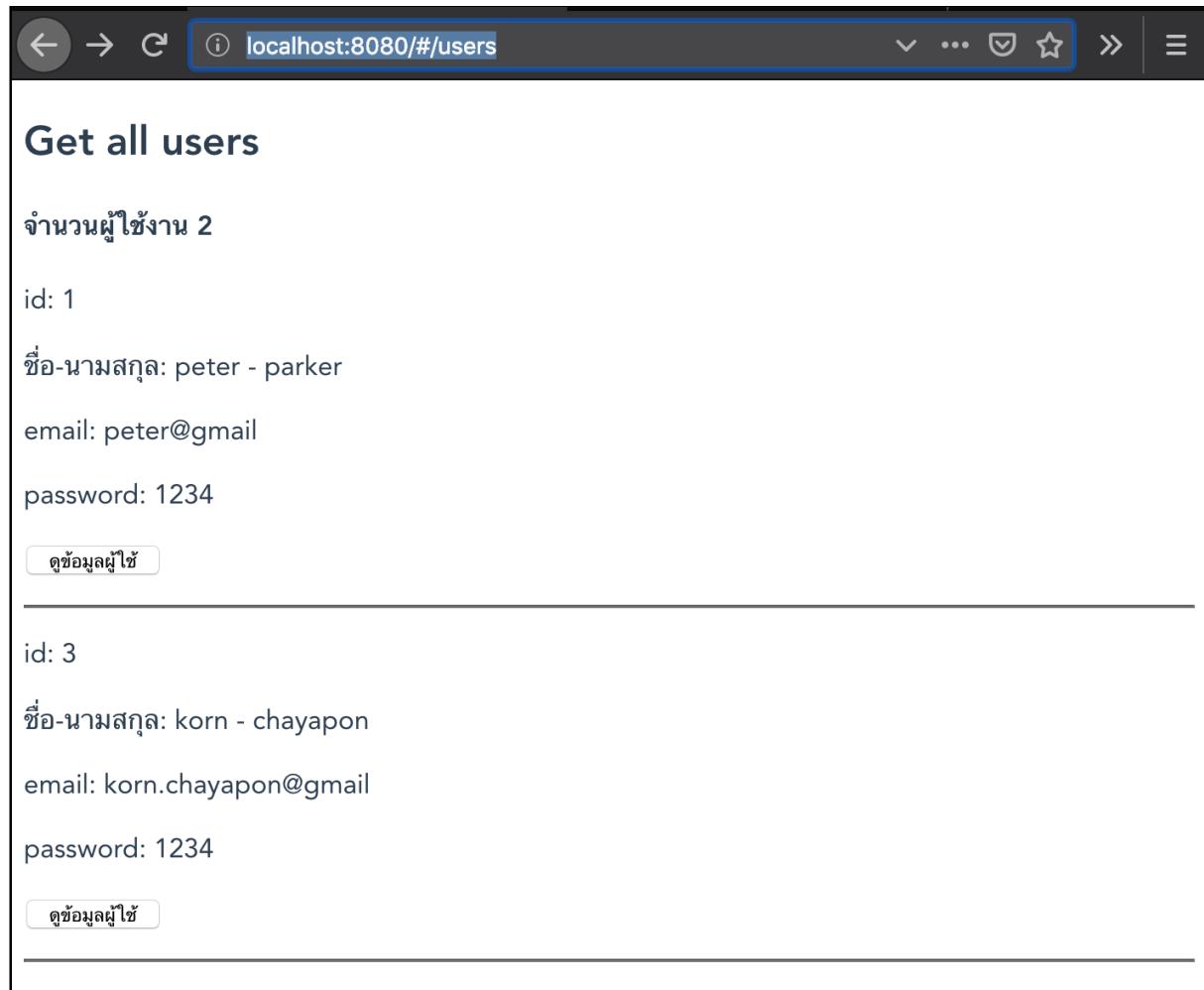
```
<p><button v-on:click="navigateTo('/user/'+user.id)">ดูข้อมูลผู้ใช้</button></p>
```

ชี้งจริง ๆ และ `v-on:click="navigateTo('/user')"` ก็คือ function onclick นั่นเองครึ่งเพียงแต่ว่า แต่ละภาษาเค้าก็จะมี syntax ของเค้า เรา ก็ว่ากันตามเค้าไป เอาเป็นว่าเราเข้าใจใช้งานได้คือจบครับ

จากนั้นเรา ไปที่ Web Browser ของเรา และทำการ เข้าไปที่ url

```
http://localhost:8080/#/users
```

โปรแกรมจะแสดงหน้าตาแบบนี้ครับ



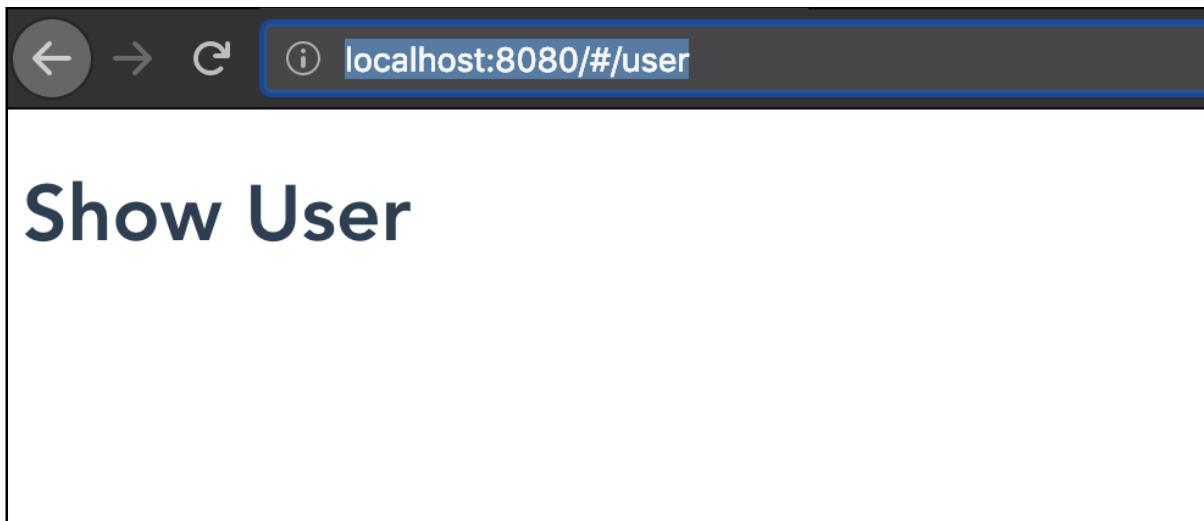
เมื่อเรากดปุ่ม ดูข้อมูลผู้ใช้ จะทำการ Route ไปตามที่กำหนดไว้ โดยคำสั่ง

```
this.$router.push(route)
```

ซึ่ง path ที่เราส่งไป เพื่อทำการ routing ก็คือ /user ซึ่งก็คือ url

```
http://localhost:8080/#/user
```

นั้นเอง ผลการกดปุ่ม ก็ไม่มีอะไรมาก จะ route ไป path ที่ส่งดังภาพครับ



แต่ความจริงแล้วเวลาที่กดดูเข้ามูล user ของแต่ละคน เราต้องทำการดึงข้อมูล user คนนั้นออกมา โดยเรามักจะกำหนดจาก id ดังนั้นเวลาที่เรา กดปุ่มดูข้อมูล ผู้ใช้ เราต้องทำการส่ง id พ่วงมาด้วย เราเริ่มเช็คกันตั้งแต่ router/index.js เลยครับ ทำการเปิด src/router/index.js ขึ้นมาครับ

มองหา code ชุดนี้ครับ

```
{
  path: '/user',
  name: 'user',
  component: UserShow
},
```

จากนั้นเปลี่ยนเป็น

```
{
  path: '/user/:userId',
  name: 'user',
  component: UserShow
},
```

พี่ที่เราจะทำการส่ง userId ของเราเพื่อไปด้วยนั่นเองครับ แน่นอนเมื่อทำการส่งไป ก็ต้องมีการรับ เราไปเขียน code สำหรับ รับค่า userId กัน ที่เรา妄แພนไว้ คือ

```
components/Users/Index.vue → (userId) → components/Users>ShowUser.vue
```

ดังนั้น ตัวที่รับงานที่เราส่งไปจัดการต่อ คือ components/Users>ShowUser.vue นั่นเอง เปิด ShowUser.vue ของเราขึ้นมาครับ จากนั้น ทำการแก้ไข code เป็น ดังนี้ครับ

```
<template>
  <div>
    <h1>Show User</h1>
    <p>User ID: {{ userId }}</p>
  </div>
</template>
<script>
export default {
  data () {
    return {
      userId: 0
    }
  },
  created () {
    this.userId = this.$route.params.userId
  }
}
</script>
<style scoped>

</style>
```

จาก code เหมือนเดิมครับ เราเริ่มทำงานที่ created () ก่อนเหมือนเดิม โดยเรามีตัว แปร หรือ data () ของเราก็คือ userId กำหนดค่าเริ่มต้นไว้ userId = 0

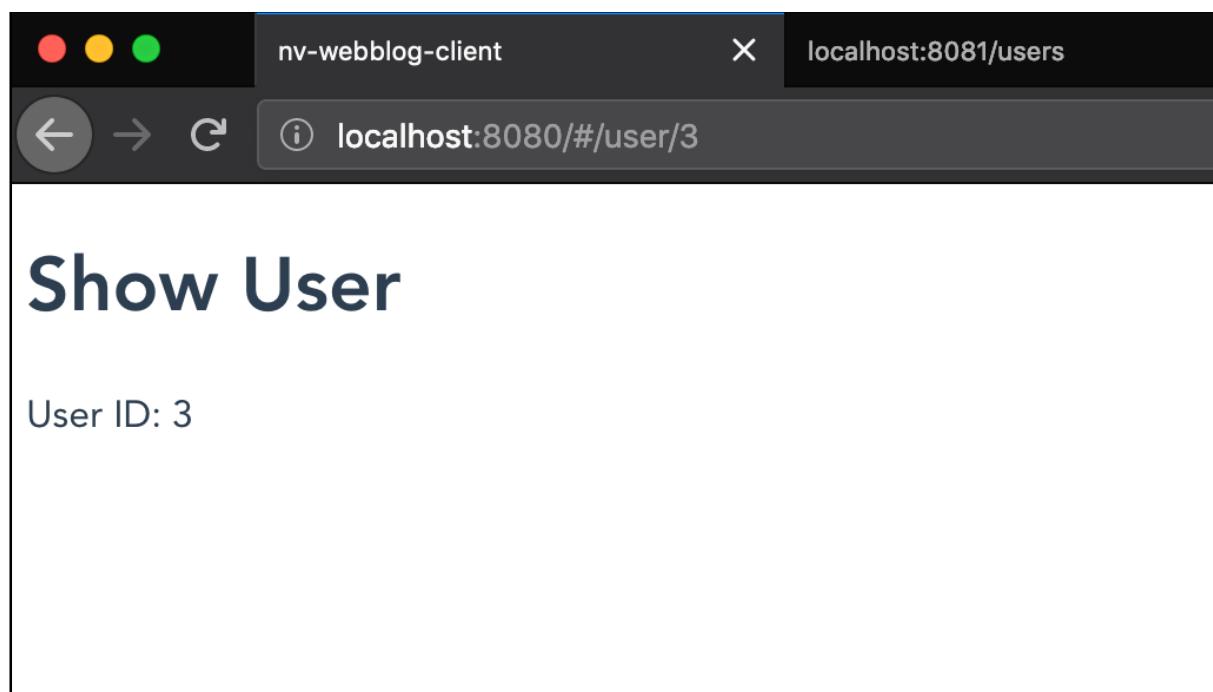
```
this.$route.params.userId
```

มาเก็บไว้ที่ userId ของเรา เอาแล้วไอ้เจ้านี่มายังไง มาตอนเราติดตั้ง vue-cli เลยคับ เค้าติดตั้ง router มาให้ เราเรียกใช้งาน this.\$route ได้เลยครับ จำไว้อย่างจะสั่งงานให้วิ่งไปไหนทำอะไร จะเป็น this.\$router (เร้าท์เตอร์นะครับ) แต่ถ้าอ่านค่าผ่านมัน เป็น this.\$route (รูท) นะครับ router != route นะคับ จำไว้ ๆ ครับ

เรามาทำการทดสอบว่าเราส่งไป เค้าได้รับมั้ย เราทดสอบ แบบ on the fly หรือแบบบ้าน ตามใจฉันเลย ก็พิมพ์ url เข้าไปเลยครับ

```
http://localhost:8080/#/user/3
```

จาก url ที่ผมจะไปเปิดบน web browser ของผม จะเห็นได้ว่าผมส่ง /3 แบบไปด้วยซึ่งคือ userId ของผู้คนนั้นเอง โดย web browser ของผม จะแสดงผลดังนี้ครับ



เพราะ template ของผมเป็นอย่างนี้ครับ

```
<template>
<div>
<h1>Show User</h1>
<p>User ID: {{ userId }}</p>
</div>
</template>
```

เมื่อผู้ทำการกดปุ่ม browser ของผู้ ก็จะวิ่งไปตามที่ผู้กำหนดไว้ พร้อมส่ง userId แบบไปด้วยนั่นเองครับ

ทำการแสดงข้อมูลผู้ใช้แต่ละคน

เมื่อเราทำการรับส่ง userId ได้แล้ว ในตอนนี้เราจะทำการ get user by id เพื่อดึงข้อมูลผู้ใช้ มาแสดงกันครับ ทำการเปิด src/components/Users>ShowUser.vue ขึ้นมาครับ และทำการแก้ code เป็นดังนี้ครับ

```
<template>
  <div>
    <h1>Show User</h1>
    <p>id: {{ user.id }}</p>
    <p>ชื่อ-นามสกุล: {{ user.name }} - {{ user.lastname }}</p>
    <p>email: {{ user.email }}</p>
    <p>password: {{ user.password }}</p>
  </div>
</template>
<script>
import UsersService from '@/services/UsersService'

export default {
  data () {
    return {
      user: null
    }
  },
  async created () {
    let userId = this.$route.params.userId
    this.user = (await UsersService.show(userId)).data
  }
}
</script>
<style scoped>
</style>
```

จาก code ของเรา เริ่มต้นจาก created () แล้ว เมื่อเดิม แต่จากที่เราใช้การ load ข้อมูลจาก api server ที่เราทำไว้ เราต้องทำการใส่ await และ async เมื่อเดิมครับ จากนั้นทำการแสดงผลผ่าน template ที่เราทำไว้นั่นเอง

ในกรณีนี้ถ้าเรา error จากการทำงานต่าง ๆ เราจะไม่รู้เลย เราจะทำการดักมันไว้ ก่อน ด้วย try/catch ในกรณีที่เราโหลดข้อมูลจาก api server และทำงานผิดพลาดครับ จะได้แจ้ง error ให้กับผู้ใช้งานได้ทราบครับ โดยผมจะทำการแก้ไข code ของผมใน created () ของผมเป็นดังนี้ครับ

```
async created () {
  try {
    let userId = this.$route.params.userId
    this.user = (await UserService.show(userId)).data
  } catch (error) {
    console.log (error)
  }
}
```

อย่าลืมตามไปแก้ที่ src/components/Users/Index.vue ของเราด้วยนะครับ จะได้รู้เวลาเมื่อเกิด error ตอนโหลดข้อมูลครับ

ทำการเพิ่ม แก้ไข และ ลบ ข้อมูลผู้ใช้งาน

สร้างผู้ใช้งาน

ในหัวข้อที่ผ่านมาเราได้ทำการ ตั้งข้อมูลผู้ใช้งานทั้งหมด และดูข้อมูลผู้ใช้งานแบบรายบุคคลได้แล้ว ตอนนี้เราจะมาทำการเพิ่มผู้ใช้งานของเรากันครับ ในบทก่อนหน้านี้ เราได้ทำการเพิ่มข้อมูลผู้ใช้งานเข้าไปในระบบของเราได้แล้ว ผ่านโปรแกรม POSTMAN นั้นหมายความว่า ระบบ Back End ของเรา Work แล้ว นั่นเอง

มาถึงตอนนี้เราจะทำการเพิ่มเข้ามูลจาก Vue Component ของเรา กันครับ ทำการเปิด src/components/Users/CreateUser.vue ขึ้นมา และทำการแก้ไข code โปรแกรมของเราเป็นดังนี้ครับ

```

<template>
<div>
  <h1>Create User</h1>
  <form v-on:submit.prevent = "createUser">
    <p>name: <input type="text" v-model="user.name"></p>
    <p>lastname: <input type="text" v-model="user.lastname"></p>
    <p>email: <input type="text" v-model="user.email"></p>
    <p>password: <input type="text" v-model="user.password"></p>
    <p><button type="submit">create user</button></p>
  </form>
</div>
</template>
<script>
import UsersService from '@/services/UsersService'

export default {
  data () {
    return {
      user: {
        name: '',
        lastname: '',
        email: '',
        password: '',
        status: 'active'
      }
    }
  },
  methods: {
    async createUser () {
      try {
        await UsersService.post(this.user)
        this.$router.push({
          name: 'users'
        })
      } catch (err) {
        console.log(err)
      }
    }
  }
}
</script>
<style scoped>

</style>

```

เราจะมาดูส่วนของ template กันก่อนนะครับ

```
<template>
<div>
  <h1>Create User</h1>
  <form v-on:submit.prevent = "createUser">
    <p>name: <input type="text" v-model="user.name"></p>
    <p>lastname: <input type="text" v-model="user.lastname"></p>
    <p>email: <input type="text" v-model="user.email"></p>
    <p>password: <input type="text" v-model="user.password"></p>
    <p><button type="submit">create user</button></p>
  </form>
</div>
</template>
```

โดยส่วน template ของเรา มีตัวที่นำสนใจคือ v-model จะทำหน้าที่ผูกตัวแปรกับ user ใน data () ของเรา เอาไว้ทำการรับค่านั้นเอง

ผมทำการแก้ไข code ในส่วนของ template อีกนิดนึงนะครับจะได้เข้าใจมากขึ้น โดย ทำการแก้ไข code เป็นดังนี้

```
<template>
<div>
  <h1>Create User</h1>
  <form v-on:submit.prevent = "createUser">
    <p>name: <input type="text" v-model="user.name"></p>
    <p>lastname: <input type="text" v-model="user.lastname"></p>
    <p>email: <input type="text" v-model="user.email"></p>
    <p>password: <input type="text" v-model="user.password"></p>
    <p><button type="submit">create user</button></p>
  </form>
  <hr>
  <div>
    <p>name: {{ user.name }}</p>
    <p>lastname: {{ user.lastname }}</p>
    <p>email: {{ user.email }}</p>
    <p>password: {{ user.password }}</p>
  </div>
</div>
```

```

</template>
<script>
import UserService from '@/services/UserService'

export default {
  data () {
    return {
      user: {
        name: '',
        lastname: '',
        email: '',
        password: '',
        status: 'active'
      }
    }
  },
  methods: {
    async createUser () {
      try {
        await UserService.post(this.user)
        this.$router.push({
          name: 'users'
        })
      } catch (err) {
        console.log(err)
      }
    }
  }
}
</script>
<style scoped>
</style>

```

เมื่อเราทำการเรียกเข้าไป url

```
http://localhost:8080/#/user/create
```

ผ่าน web browser ของเรา และทำการแก้ฟิล์ดต่าง ๆ ของเรา ส่วนของการแสดงผลด้านล่างของเราก็จะเปลี่ยนแปลงด้วยนั่นเอง จากนั้นให้สังเกตุส่วนของ form นะครับ

```
<form v-on:submit.prevent = "createUser">
  <p>name: <input type="text" v-model="user.name"></p>
  <p>lastname: <input type="text" v-model="user.lastname"></p>
  <p>email: <input type="text" v-model="user.email"></p>
  <p>password: <input type="text" v-model="user.password"></p>
  <p><button type="submit">create user</button></p>
</form>
```

ส่วนของ form ของเรานอกจาก v-model แล้ว เราไม่ code สำหรับ submit ไปที่ methods ที่เราเขียนรองรับไว้ คือ createUser นั่นเอง โดย

```
<form v-on:submit.prevent = "createUser">
```

นั้น จะทำการ submit ไปที่ methods ที่เราเขียนรับไว้ โดยไม่ redirect นั่นเองไปไหนนั่นเองครับ

มาดู code ในส่วนของการ submit ที่ methods create user กันครับ

```
methods: {
  async createUser () {
    try {
      await UsersService.post(this.user)
      this.$router.push({
        name: 'users'
      })
    } catch (err) {
      console.log(err)
    }
  }
}
```

การทำงานของ code ส่วนนี้ คือใช้งาน Services ที่เราเตรียมไว้ และ import เข้ามา โดย post user ของเราไป จากนั้นร่องน้ำเส้นจะกระวนการ และทำการ redirect ที่หน้า

```
http://localhost:8080/#/users
```

ผลจากการทดสอบเมื่อทำการกรอกข้อมูล ที่ create user ของผมแล้ว หากทำงานได้ถูกต้อง user ที่ผมสร้างขึ้นจะมาแสดงที่หน้านี้ของผมครับ

localhost:8080/#/user/create

Create User

name: tony

lastname: stark

email: tony.stark@gmail.com

password: 12345678

Console Network Performance Memory Application Vue

[HMR] Waiting for update signal from WDS... log.js?42

ดูข้อมูลผู้ใช้

id: 3

ชื่อ-นามสกุล: korn - chayapon

email: korn.chayapon@gmail

password: 1234

id: 5

ชื่อ-นามสกุล: tony - stark

email: tony.stark@gmail.com

password: 12345678

เมื่อทำการเปิด database ของเราจะเห็นว่ามีการเพิ่ม tony stark เข้ามาในระบบ
ของเราแล้วครับ

	id	email	password	name	lastname	status	create
1	1	peter@gmail	1234	peter	parker	active	2019-01-01 00:00:00
2	3	korn.chayapon@gmail	1234	korn	chayapon	active	2019-01-01 00:00:00
3	5	tony.stark@gmail.com	12345678	tony	stark	NULL	2019-01-01 00:00:00

ตอนนี้เรารสามารถ ทำการเพิ่มผู้ใช้งานของเราได้แล้วนะครับ

แก้ไขข้อมูลผู้ใช้งาน

ก่อนจะทำการแก้ไขข้อมูลเราต้องโหลดข้อมูลผู้ใช้งานมาก่อน แต่เหมือนผมจะจำ
ได้ว่า route ใน <http://localhost:8080/user/edit> ของผมยังไม่ได้ส่งอะไรมาให้
เราไปเปิดไฟล์ router/index.js ขึ้นมา มองหา path

```
/user/edit
```

แล้วทำการแก้ไข เพื่อให้ล็อก parameter userId และไปด้วยดังนี้

```
{
  path: '/user/edit/:userId',
  name: 'user-edit',
  component: UserEdit
},
```

ตอนนี้ edit path ของเราได้ส่ง userId มาให้เราแล้วครับ เราไปทำการแก้ code ส่วน Edit User กันเลยครับ เปิด src/components/Users/EditUser.vue ขึ้นมาทำการแก้ไข code ของเราเป็นดังนี้ครับ

```
<template>
<div>
  <h1>Edit User</h1>
  <form v-on:submit.prevent = "editUser">
    <p>name: <input type="text" v-model="user.name"></p>
    <p>lastname: <input type="text" v-model="user.lastname"></p>
    <p>email: <input type="text" v-model="user.email"></p>
    <p>password: <input type="text" v-model="user.password"></p>
    <p><button type="submit">edit user</button></p>
  </form>
  <hr>
  <div>
    <p>name: {{ user.name }}</p>
    <p>lastname: {{ user.lastname }}</p>
    <p>email: {{ user.email }}</p>
    <p>password: {{ user.password }}</p>
    <p></p>
  </div>
</div>
</template>
<script>
import UsersService from '@/services/UsersService'

export default {
  data () {
    return {
      user: {
        name: '',
        lastname: '',
        email: '',
        password: '',
        status: 'active'
      }
    }
  },
  methods: {
    async editUser () {
      try {
        await UsersService.put(this.user)
```

```
        this.$router.push({
          name: 'users'
        })
      } catch (err) {
        console.log(err)
      }
    },
  },
  async created () {
    try {
      let userId = this.$route.params.userId
      this.user = (await UserService.show(userId)).data
    } catch (error) {
      console.log(error)
    }
  }
}
</script>
<style scoped>
</style>
```

จากนั้นเปิด web browser ของเราไปที่ (ตรวจสอบ id ของเราด้วยว่ามีอยู่หรือไม่)

```
http://localhost:8080/#/user/edit/1
```

จะแสดงผลดังภาพ จากนั้นทำการแก้ไข ข้อมูลและบันทึกครับ

Edit User

name:

lastname:

email:

password:

name: user2

lastname: user2lastname

email: user2@gmail.com

password: 12345678



password: 12345678

จำนวนผู้ใช้งาน 5

id: 5

ชื่อ-นามสกุล: user2xxx - user2lastnamexxx

email: user2xxx@gmail.com

password: 12345678

จำนวนผู้ใช้งาน 5

id: 6

ชื่อ-นามสกุล: user3 - user3lastname

email: user3@gmail.com

password: 12345678

การทำงานเหมือนกับโปรแกรมสองส่วนรวมกัน คือ ส่วนໂ Holdenข้อมูล สามารถดูจาก ShowUser.vue ประกอบหรือ กลับไปอ่านบททวนดูครับ อีกส่วนคือจาก CreateUser.vue ซึ่งการส่งข้อมูลไปเหมือนกัน เพียงแต่ผนเปลี่ยน HTTP Method จาก Post คือสร้าง เป็น Put คือ อัพเดทนั่นเองครับ

Post เพื่อสร้าง

```
await UsersService.post(this.user)
```

Put เพื่ออัพเดท

```
await UsersService.put(this.user)
```

ตอนนี้เรารสามารถดูข้อมูลผู้ใช้ทั้งหมด และแบบรายบุคคล รวมถึงสร้างและแก้ไขได้แล้วนะครับ

ทำการเชื่อมโยงส่วนต่าง ๆ ด้วยปุ่มที่ หน้าของผู้ใช้งานของเรากันก่อนครับ จะได้สร้างและแก้ไขผ่าน หน้าหลักในส่วนของ user หรือ <http://localhost:8080/#/users> ของเราได้

จะทำการเพิ่มปุ่มแก้ไขที่หน้าหลักของเราดังนี้นะครับ

```
<p><button v-on:click="navigateTo('/user/' + user.id)">ดูข้อมูลผู้ใช้</button>
<button v-on:click="navigateTo('/user/edit/' + user.id)">แก้ไขข้อมูล
</button></p>
```

และทำการเพิ่มปุ่มให้สร้างผู้ใช้งานได้ดังนี้ครับ

```
<h4>จำนวนผู้ใช้งาน {{users.length}}</h4>
<p><button v-on:click="navigateTo('/user/create')">สร้างผู้ใช้งาน</button></p>
<div v-for="user in users" v-bind:key="user.id">
```

ลบข้อมูลผู้ใช้งาน

หลังจากเรารอ่าน สร้าง และแก้ไข ข้อมูลผู้ใช้งานได้แล้ว เราจะมาทำการลบผู้ใช้งานของเรากันนะครับ แต่การลบข้อมูลผู้ใช้นี้เราไม่ต้อง route ไปไหน เราอยากรบเราก็ลับเลยครับ นั่นคือจะเขียน methods การลบที่หน้าหลักของ user ของเราครับ ก่อนอื่นจะลบเราต้องทำการเพิ่มปุ่มลบก่อนครับ โดยผมแก้ไข code ชุดนี้ครับ

```
<p>
  <button v-on:click="navigateTo('/user/' + user.id)">ดูข้อมูลผู้ใช้</button>
  <button v-on:click="navigateTo('/user/edit/' + user.id)">แก้ไขข้อมูล
</button>
  <button v-on:click="deleteUser(user)">ลบข้อมูล</button>
</p>
```

เมื่อผมเพิ่มปุ่มลบผู้ใช้งานของเราแล้ว และกำหนดให้ทำงานที่ deleteUser เราต้องไปทำการเพิ่ม methods ให้มันทำงานครับ โดยเราเพิ่ม code ที่ methods ของเราดังนี้ครับ

```
methods: {
  navigateTo (route) {
    this.$router.push(route)
  },
  async deleteUser (user) {
    let result = confirm("Want to delete?")
    if (result) {
      try {
        await UsersService.delete(user)
      } catch (err) {
        console.log(err)
      }
    }
  }
}
```

เมื่อเราเรียกไปที่ <http://localhost:8080/#/users> จะมีปุ่มลบข้อมูลเพิ่มขึ้นมา เมื่อเราทำการกดปุ่มลบ จะทำการ confirm delete ก่อน ถ้าตอบใช่ จะทำการลบข้อมูล

ใน database ของเรานั้นที่ การทดสอบ ทดสอบโดยการกดลบข้อมูล หนึ่งครั้ง ครั้งเดียวจะครับ แล้วกด refresh สังเกตว่า ผู้ใช้ที่เราลบจะหายไป แต่มันคงหน้าเป็นมาก ที่เราลบแต่ละครั้ง ต้องการค่อยกด refresh เราแก้ไขได้โดยการเพิ่ม code เข้าไปดังนี้ครับ

```
methods: {
  navigateTo (route) {
    this.$router.push(route)
  },
  async deleteUser (user) {
    try {
      await UsersService.delete(user)
      this.refreshData()
    } catch (err) {
      console.log(err)
    }
  },
  async refreshData() {
    this.users = (await UsersService.index()).data
  }
}
```

มาถึงตอนนี้ เราจะลบผู้ใช้งานปุ๊บ ก็จะหายไปปีบเลยครับ ตอนนี้เราจะจัดการ CRUD ผ่านหน้าบ้านด้วย vuejs ของเราได้แล้วนะครับ เราลืม ๆ เรื่องความสวยงามและ action ต่างไปก่อนนะครับ เราทำระบบหลัก ๆ ให้เรียบร้อยกันก่อน อยากรวยเราก็ปรับ css เข้าไป อยากรีบ action เทพ ๆ เราค่อยไปเล่นกับ json ตอนท้าย ๆ กันครับ

บทที่ 9 จัดการ Permission

ในบรรดาทุกบท ผมว่าบทนี้ยกสุดครับ ผมจะพยายามอธิบายอย่างง่าย ๆ นะครับ ค่อยเป็น ๆ ค่อย ๆ ไปครับ

ทำความรู้จักกับ Token

ถ้าใครเคยเขียนเว็บไซต์ด้วย PHP หรือ Server Side Script อื่น จะรู้จัก Cookie หรือ Session ซึ่ง Token ที่เราจะกล่าวถึงก็จะคล้าย ๆ กัน คือ เอาไว้เก็บข้อมูลที่ Server ส่งมาให้กับ Client แล้วเอาไว้ตรวจสอบการเข้าถึงข้อมูลระหว่างกันนั่นเอง

แล้วมันเกิดขึ้นเมื่อไหร่ ยังไง? ปกติแล้วเราจะกระบวนการจะเริ่มตั้งแต่เรา login ผ่านแล้ว Server หรือ Back End ของเราจะสร้าง Token แล้วส่งกลับมาให้ Client หรือ Front End ของเรา Client ของเราจะเก็บไว้ใน Local Storage หรือในเครื่องที่เราใช้เปิดเว็บไซต์นั่นเอง ต่อไปเมื่อ Client จะเข้าถึง Server ในส่วนที่มีการป้องกัน Client ก็จะส่ง Token ให้ตรวจสอบเพื่อขอเข้าใช้งานระบบ ถ้าใครเคยเข้าผับ และปั่นแซน เอาไว้เข้ารอบต่อไปโดยไม่ต้องตรวจบัตร การ login เมื่อนอกันแสดงบัตร ส่วน Token สัญญาลักษณ์ที่ปั๊ม บนแซนเราครับ

ติดตั้ง package ที่จำเป็น

ในส่วนนี้เราจะใช้ package หลายตัวกันเลย ผมมีเกียจนานั่ง Install ทีละตัวด้วยคำสั่ง npm ผมทำการใส่ code เพิ่มเติมดังนี้ เปิด package.json ในโฟล์เดอร์ server ของเรารีบามาครับ จากนั้นทำการแก้ไข code ใน dependencies ของเรา เป็นดังนี้ครับ

```
"bcrypt-nodejs": "^0.0.3",
"bluebird": "^3.5.3",
"joi": "^13.7.0",
"jsonwebtoken": "^8.4.0",
"passport": "^0.4.0",
"passport-jwt": "^4.0.0"
```

```
{} package.json ●
1  {
2      "name": "nv-webblog-server",
3      "version": "1.0.0",
4      "description": "nodejs vuejs webblog server (back end)",
5      "main": "src/app.js",
6      "scripts": {
7          "test": "echo \\\"Error: no test specified\\\" && exit 1"
8      },
9      "author": "chayapon chandra",
10     "license": "ISC",
11     "dependencies": [
12         "body-parser": "^1.18.3",
13         "cors": "^2.8.5",
14         "express": "^4.16.4",
15         "sequelize": "^4.43.0",
16         "sqlite3": "^4.0.6",
17         "bcrypt-nodejs": "^0.0.3",
18         "bluebird": "^3.5.3",
19         "joi": "^13.7.0",
20         "jsonwebtoken": "^8.4.0",
21         "passport": "^0.4.0",
22         "passport-jwt": "^4.0.0"
23     ]
24 }
25
```

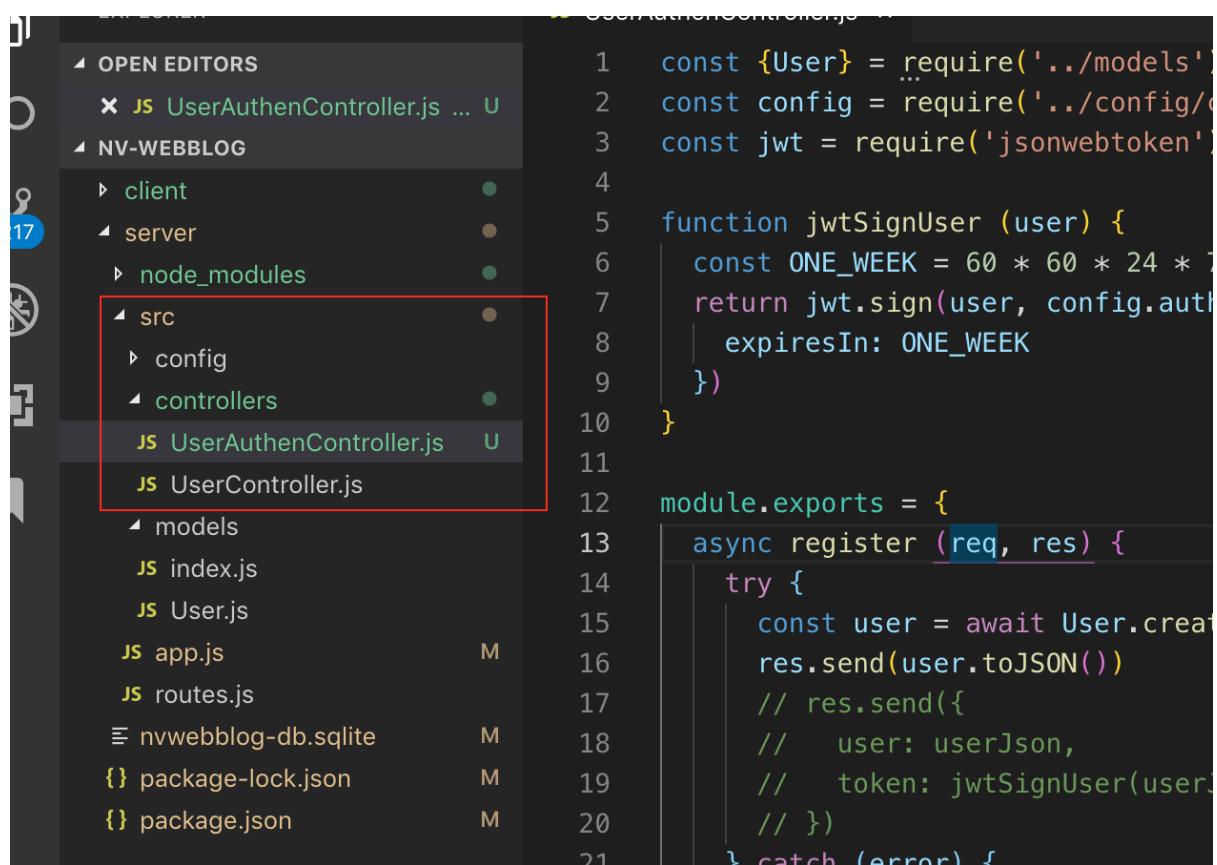
จากนั้นทำการ หยุดการทำงาน Server ของเรา ที่ Terminal ของแล้วทำการ พิมพ์คำสั่ง

```
npm install
```

คำสั่งนี้จะทำการติดตั้ง package ที่เรายังไม่มีให้ทั้งหมดจาก dependencies ของเราครับ

เริ่มต้นสร้าง Token บน Server กัน

เราจะเริ่มการสร้าง Token จาก Server หรือ Back End ส่งไปให้ Front End กันครับ โดยเริ่มจาก ผู้จะสร้าง UserAuthenController.js ขึ้นมาจัดการเรื่องนี้โดยเฉพาะครับ ผู้สร้างไว้ที่นี่ครับ



The screenshot shows the VS Code interface with the Explorer sidebar on the left and the Editor pane on the right. The Explorer sidebar displays the project structure of 'NV-WEBBLOG' with a red box highlighting the 'src' folder under 'server'. Inside 'src', there are 'controllers' and 'UserAuthenController.js'. The Editor pane shows the code for 'UserAuthenController.js'.

```
const {User} = require('../models')
const config = require('../config/config')
const jwt = require('jsonwebtoken')

function jwtSignUser (user) {
  const ONE_WEEK = 60 * 60 * 24 * 7
  return jwt.sign(user, config.auth.secretOrKey,
    { expiresIn: ONE_WEEK })
}

module.exports = {
  async register (req, res) {
    try {
      const user = await User.create(req.body)
      res.send(user.toJSON())
      // res.send({
      //   user: userJson,
      //   token: jwtSignUser(user)
      // })
    } catch (error) {
      res.status(400).send(error.message)
    }
  }
}
```

เมื่อเราสร้างไฟล์ UserAuthenController.js เสร็จแล้วเราจะมาเขียน code ตามนี้กันนะครับ โดยผู้จะค่อย ๆ เพิ่มทีละส่วนและ อธิบายนะครับ จะได้ไม่งง เริ่มต้นจากส่วนนี้ครับ

```
const {User} = require('../models')
const config = require('../config/config')
const jwt = require('jsonwebtoken')
```

'jsonwebtoken' คือ package ที่เราติดตั้งไว้ตอนเราเพิ่มใน dependencies เอาไว้จัดการกับ token ของเรารับ

จากนั้นผมเพิ่ม export function ของผมเพื่อให้คนอื่นเรียกใช้ได้

```
const {User} = require('../models')
const config = require('../config/config')
const jwt = require('jsonwebtoken')

function jwtSignUser (user) {
  const ONE_WEEK = 60 * 60 * 24 * 7
  return jwt.sign(user, config.authentication.jwtSecret, {
    expiresIn: ONE_WEEK
  })
}

module.exports = {
  async register (req, res) {
    try {
      const user = await User.create(req.body)
      res.send(user.toJSON())
    } catch (error) {
      res.status(400).send({
        error: 'The content information was incorrect'
      })
    }
  },
  async login (req, res) {
    try {
      const {email, password} = req.body
      const user = await User.findOne({
        where: {
          email: email
        }
      })

      if(!user) {
        return res.status(403).send({
          error: 'Email or password is incorrect'
        })
      }

      const token = jwtSignUser(user)

      res.cookie('token', token, {
        maxAge: ONE_WEEK
      }).status(200).send({
        user: user.toJSON(),
        token: token
      })
    } catch (error) {
      res.status(400).send({
        error: 'The content information was incorrect'
      })
    }
  }
}
```

```

        error: 'User/Password not correct'
    })
}

const isPasswordValid = await user.comparePassword(password)
if (!isPasswordValid) {
    return res.status(403).send({
        error: 'User/Password not correct'
    })
}

const userJSON = user.toJSON()
res.send(userJSON)

} catch (error) {
    res.status(500).send({
        error: 'Error! from get user'
    })
}
}
}

```

ซึ่งเราจะโฟกัสไปที่ function ในการ login ของผู้คนก่อนนะครับ โดยถ้าไม่ตาม code นั้นสื่อความหมายได้ชัดเจนอยู่แล้วว่า ถ้าค้น email ไม่เจอให้ error ถ้าเจอให้ไปตรวจสอบว่า password ตรงหรือป่าว แต่เอ๊ะ แล้วบรรทัดนี้มันยังไงกัน

```
const isPasswordValid = await user.comparePassword(password)
```

นั้นสิ มันยังไม่มีครับ เราจำลังจะไปเขียนกันครับ เปิด src/models/User.js ของเราขึ้นมาครับ แล้วทำการแก้ไข code ของเราเป็นดังนี้ครับ

```

module.exports = (sequelize, DataTypes) => {
    const User = sequelize.define('User', {
        email: DataTypes.STRING,
        password: DataTypes.STRING,
        name: DataTypes.STRING,
        lastname: DataTypes.STRING,
        status: DataTypes.STRING
    })

    User.prototype.comparePassword = function (password) {
        if (password == this.password) {

```

```
        return true
    }
    return false
}

User.associate = function (models) {}

return User
}
```

code ที่เพิ่มขึ้นมาคือส่วนนี้ครับ

```
User.prototype.comparePassword = function (password) {
  if (password == this.password) {
    return true
  }
  return false
}

User.associate = function (models) {}
```

พออธิบายคร่าว ๆ ว่า เป็นการเพิ่ม comparePassword Function เข้าไปใน model ของเรานั้นเองครับ พอเราเพิ่มแล้ว เราจะสามารถเรียกใช้ function นี้ผ่าน user ที่เรารับค่าจาก User Model ของเรานั้นเอง

พอทำการทดสอบการ login ของผู้ด้วย POSTMAN จะได้ผลดังนี้ครับ เพราะผมสั่งให้ทำการส่ง user ออกมากำหนดทำการ Authenticate ได้สำเร็จ

The screenshot shows a POST request to `http://localhost:8081/login`. The request body is a JSON object with fields `email` and `password`. The response status is `200 OK`, and the response body is a user object with various properties like `id`, `email`, `password`, `name`, `lastname`, `status`, `createdAt`, and `updatedAt`.

```

1 {
2   "email": "peterxx@gmail.com",
3   "password": "1234xxx"
4 }
    
```

```

1 {
2   "id": 1,
3   "email": "peterxx@gmail.com",
4   "password": "1234xxx",
5   "name": "peterxxx",
6   "lastname": "parkerxxx",
7   "status": "active",
8   "createdAt": "2019-03-05T13:40:13.802Z",
9   "updatedAt": "2019-03-10T06:59:20.206Z"
10 }
    
```

จากนั้นจะมาเพิ่มเติม code อีกนิดหน่อยครับในส่วน UserAuthenController.js ของผมให้ทำการสร้าง Token แล้วส่งไปด้วยเลย เวลาที่ใครทำการ login สำเร็จ โดยผมทำการแก้ไข code ทั้งหมดของผมดังนี้ครับ

```

const {User} = require('../models')
const config = require('../config/config')
const jwt = require('jsonwebtoken')

function jwtSignUser (user) {
  const ONE_WEEK = 60 * 60 * 24 * 7
  return jwt.sign(user, config.authentication.jwtSecret, {
    expiresIn: ONE_WEEK
  })
}

module.exports = {
  async register (req, res) {
    try {
      const user = await User.create(req.body)
      res.send(user.toJSON())
    } catch (error) {
    }
  }
}
    
```

```
res.status(400).send({
  error: 'The content information was incorrect'
})
},
}

async login (req, res) {
try {
  const {email, password} = req.body
  const user = await User.findOne({
    where: {
      email: email
    }
  })

  if(!user) {
    return res.status(403).send({
      error: 'User/Password not correct'
    })
  }

  const isPasswordValid = await user.comparePassword(password)
  if (!isPasswordValid) {
    return res.status(403).send({
      error: 'User/Password not correct'
    })
  }

  res.send({
    user: userJSON,
    token: jwtSignUser(userJSON)
  })
}

} catch (error) {
  res.status(500).send({
    error: 'Error! from get user'
  })
}
}
}
```

จากนั้นทำการเพิ่ม route ไปที่ login ของเรา โดยเปิด src/routes.js ใน server ของเราขึ้น แล้วทำการใส่ code เพิ่มเข้าไป โดยเพิ่มทั้ง controller ที่เรียกใช้งาน และ route mapping ด้วย

```
const UserAuthenController = require('../controllers/UserAuthenController')
```

```
app.post('/login',
  UserAuthenController.login
)
```

แล้วเข้าเปิด config/config.js ในโฟล์เดอร์ server ของเราครับ แล้วเพิ่ม code เข้าไปตามนี้ครับ โดย 'secret' ของเราจะเปลี่ยนเป็น string อะไรก็ได้ครับ เป็นคำที่จะนำมาทำการเข้ารหัสแล้วทำการสร้าง token ของเราเลย ๆ ครับ

จากนั้นผมทดสอบส่วนของ login ของผมด้วย postman จะเห็นว่ามีการส่ง Token กลับมาให้ Front End ของเราด้วย เราจะใช้ Token ตัวนี้ล็อครับในการขอเข้าผับครั้งต่อๆ ไปแทนการแสดงบัตรประชาชนกันครับ



```
app.js          JS isAuthenticatedController.js      JS routes.js      ●      JS config.js      X      ⌂
```

nv-weblog > server > src > config > **JS config.js** > [?] <unknown> > 🔒 authentication >

```
1  module.exports = {
2    port: 8081,
3    db: {
4      database: process.env.DB_NAME || 'nvWebblogDb',
5      user: process.env.DB_User || 'root',
6      password: process.env.DB_PASS || '',
7      options: {
8        dialect: process.env.DIALECT || 'sqlite',
9        storage: './nvwebblog-db.sqlite'
10       },
11     },
12     authentication: {
13       jwtSecret: "test"
14     }
15 }
```

POST http://localhost:8081/login

Params Authorization Headers (1) Body Pre-request Script Tests Cookies Code

Body (raw JSON)

```

1 {
2   "email": "peterxx@gmail.com",
3   "password": "1234xxx"
4 }

```

Status: 200 OK Time: 55 ms Size: 836 B

Pretty Raw Preview JSON

```

1 {
2   "user": {
3     "id": 1,
4     "email": "peterxx@gmail.com",
5     "password": "1234xxx",
6     "name": "peterxxx",
7     "lastname": "parkerxxx",
8     "status": "active",
9     "createdAt": "2019-03-05T13:40:13.802Z",
10    "updatedAt": "2019-03-10T06:59:20.206Z"
11  },
12  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwiZW1haWwiOiJwZXRLcnh4QGdtYWlsIiwicGFzc3dvcmQiOiIxMjM0eHh4IiwibmFtZSI6InBldGVyeHh4IhbWUiOiJwYXJrZXJ4eHgiLCJzdGF0dXMiOiJhY3RpdmUiLCJjcmVhdGVkQXQiOiIyMDE5LTazLTA1VDEz0jQw0jEzLjgwMhdGVkQXQiOiIyMDE5LTazLTEwVDA20jU50jIwljIwnlojLCJpYXQiOjE1NTIyMDc5MTUsImV4cCI6MTU1MjgxMjcxNX0.JEw0KupK1LT9ibZhhQqTQa6ch2ojV1XY28Foml65-LM"

```

เมื่อได้ token มาแล้วเราจะทำการสร้าง Form Login จากหน้า Front End ของเราเพื่อทำการเก็บ Token และ ข้อมูลของเราลงใน Local Storage กันครับ

ไปที่โฟล์เดอร์ client ของเรา กันครับ จากนั้นเราจะทำการสร้างส่วนของการ login ตามลำดับต่อไปนี้ จำ workflow นี้ไว้เลยก็ได้นะครับ เวลาเราจะทำการสร้าง Vue Component เราจะได้พลาดครับ

- มี Service ที่จะใช้งานหรือยัง ถ้ายังให้สร้าง service ก่อน
- สร้าง Vue Component (Login Component)
- เชื่อมโยง Vue Component ด้วย Route

ผมจะทำการเชื่อมโยง Vue Component ของผมผ่าน route ใหม่บน Server ผมสร้าง Service ก่อนเลยครับ โดยผมตั้งชื่อว่า

services/AuthenService.js จากนั้นทำการเพิ่ม code ดังนี้ครับ

```
import Api from '@/services/Api'

export default {
  register (credentials) {
    return Api().post('register', credentials)
  },
  login (credentials) {
    return Api().post('login', credentials)
  }
}
```

ลักษณะจะคล้ายกับ User Service ที่เราเคยสร้าง โดยผมสร้าง function login และ register เพื่อ Post Data ไปที่ Server หรือ Back End ของเราเองครับ

จันน์ทำการสร้าง Login Vue Component กันก่อนเลยครับ ผมสร้างไว้ที่ src/components/Login.vue เลยนะครับ

จากนั้นทำการเปิด Login.vue ของเราขึ้นมา แล้วทำการใส่ code ดังนี้ครับ

```
<template>
  <div>
    <h1>User Login</h1>
    <form v-on:submit.prevent="onLogin">
      <p>Username: <input type="text" v-model="email" /></p>
      <p>Password: <input type="password" v-model="password" /></p>
      <p><button type="submit">Login</button></p>
    </form>
  </div>
</template>
<script>

  import AuthenService from '@/services/AuthenService'
```

```

export default {
  data () {
    return {
      email: '',
      password: ''
    }
  },
  methods: {
    async onLogin () {
      try {
        const response = await AuthenService.login({
          email: this.email,
          password: this.password
        })
        console.log(response)

      } catch (error) {
        console.log(error)
      }
    }
  }
}
</script>

```

จาก code ที่เราเพิ่มขึ้นมาจะเป็นการเรียกใช้ AuthenService เพื่อทำการ Login ไปที่ Server นั้นเอง ซึ่งจะเป็นการ Post Data ไปเหมือนตอนเราใช้จัดการกับผู้ใช้งานของเรา

จากนั้นทำการเพิ่ม route เพื่อให้สามารถเรียกไปที่ url ของเราเพื่อทำการ login ได้โดย เปิดไฟล์ router/index.js ใน client ของเราขึ้นมา แล้วทำการเพิ่ม code เข้าไป

ทำการใส่ Login Component

```

// Authen
import Login from '@/components/Login'

```

```
// authen
{
  path: '/login',
  name: 'login',
  component: Login
},
```

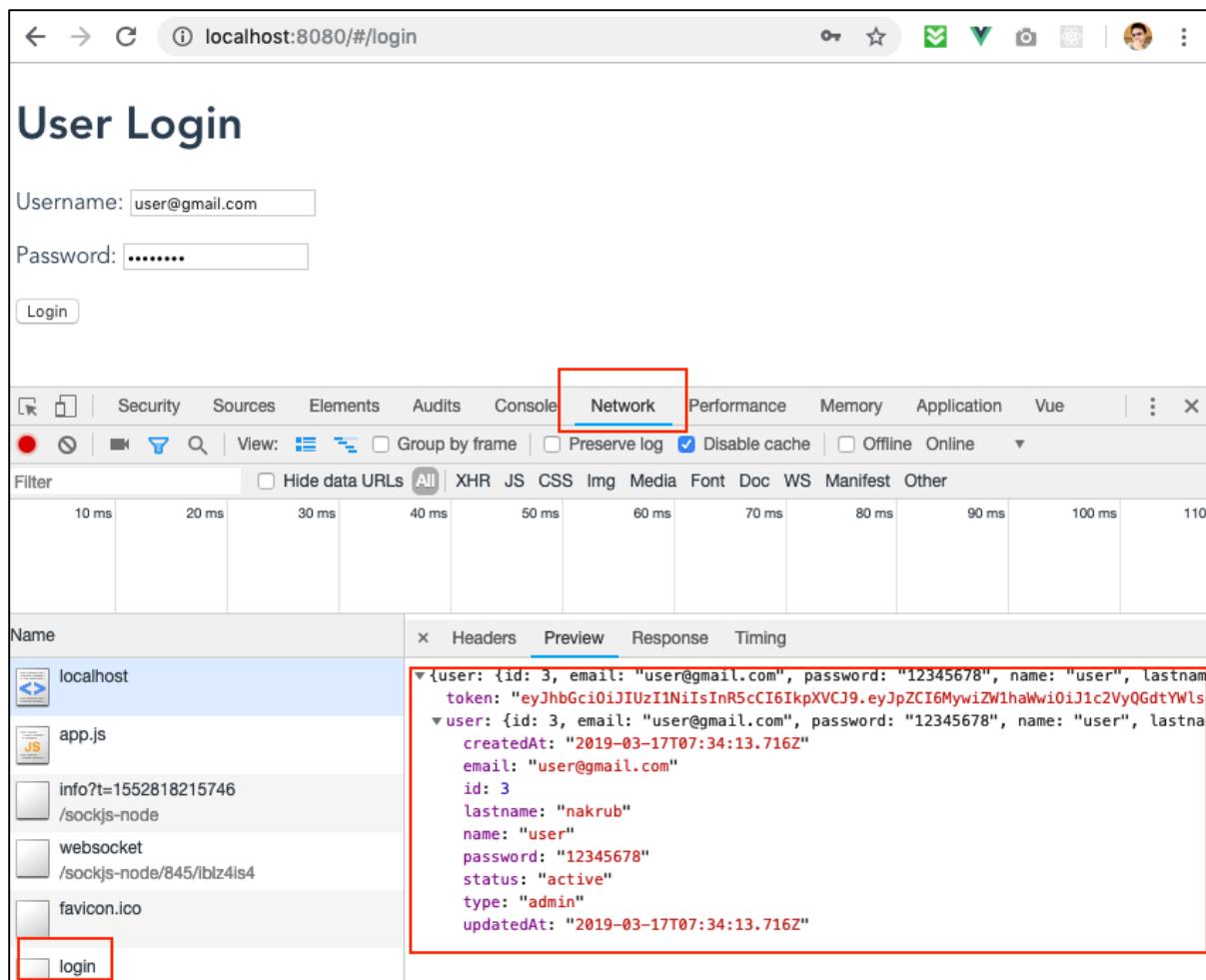
ตอนนี้ส่วนการ login ของเราพร้อมใช้งานแล้ว ให้ทำการเปิด web browser และ เรียกไปที่

```
http://localhost:8080/#/Login
```

แล้วทำการใส่ email และ password ที่เราเคยสร้างไว้ เพื่อทำการทดสอบ login หากจำไม่ได้ ให้เปิด web browser อีกแท็บหนึ่ง แล้วเข้าไปที่

```
http://localhost:8080/#/users
```

เพื่อดูก่อน เมื่อทำการกดปุ่ม login หน้าเพจของเราจะไม่ redirect ไปไหน เพราะยัง ไม่ได้ใส่ code ให้มันวิ่งไปไหนนั่นเองครับ จากนั้นเปิด Dev Tool ใน web browser ที่เราใช้ ตามตัวอย่างจะเห็นว่า Server ได้ส่ง token กลับมาให้เราใช้แล้ว นั่นเอง



เก็บ Token ลง Local Storage

จากที่เราได้ Token มาจาก Server ของเราแล้วมาถึงตอนนี้เราจะทำการเก็บของมูล Token ของเราลง Local Storage เพื่อใช้งานร่วมกันในส่วนของ Front End รวมถึงส่งกลับที่ Server เพื่อขอใช้งานในส่วนที่ต้อง Authen ด้วยครับ

ในส่วนของ Front End หรือ Client ของเรา เราจะใช้ package เพิ่มเติม เมื่อตอนเดิมครับ ผู้เปิด package.json ในโฟล์เดอร์ client ของผู้เขียนมา จากนั้นทำการ

เพิ่มเติม dependencies ที่ผู้ต้องการใช้งานเข้าไป คือ

```

"vuex": "^3.0.1",
"vuex-persistedstate": "^2.5.4",
"vuex-router-sync": "^5.0.0"

```

```
"start": "npm run dev",
"build": "node build/build.js"
},
"dependencies": {
  "axios": "^0.18.0",
  "vue": "^2.5.2",
  "vue-resource": "^1.5.1",
  "vue-router": "^3.0.1",
  "vuex": "^3.0.1",
  "vuex-persistedstate": "^2.5.4",
  "vuex-router-sync": "^5.0.0"
},
"devDependencies": {
  "autoprefixer": "^7.1.2",
  "babel-core": "^6.22.1",
  "babel-helper-vue-jsx-merge-props": "^2.0.3",
  "css-loader": "^0.28.0",
  "eslint": "^4.12.1",
  "eslint-plugin-vue": "^3.5.0",
  "file-loader": "^1.1.11",
  "html-loader": "^0.5.5",
  "html-webpack-plugin": "^3.2.0",
  "node-sass": "^4.9.0",
  "sass-loader": "^7.0.1",
  "style-loader": "^0.23.1",
  "url-loader": "^1.0.1",
  "vue-template-compiler": "^2.5.2"
}
```

จากนั้นทำการหยุด การทำงานในส่วนของ client ที่เรารันไว้ที่ Terminal ของเรา ก่อนแล้วทำการติดตั้ง package ที่เราเพิ่มไว้ ด้วยคำสั่ง

```
npm install
```

ติดตั้งเสร็จก็รัน client ของเราเหมือนเดิมนะคับ

Vuex มาแล้ว

เห็นมั้ย vuex พระเอกของเราแล้ว พอพูดถึง vuex คนเขียนօธินายยาเหยียด จัดการ data flow ครับ ให้ข้อมูลใหม่ไปในทางเดียว จัดการ state โว้โห .. ผ่อนผัน เลยครับ หรือผ่อนจะ iq น้อยเกินไปที่จะเข้าใจคำศัพท์ ยกเว้นพวงนี้ได้กันนะ

แต่ผ่อนจะบอกว่า vuex เนี่ยมันช่วยให้เราง่ายขึ้น ง่ายยังไงกันล่ะ ?

เขียน code กันก่อนดีกว่าครับ พอเปิด code เล้ามาอธินายจะเข้าใจง่ายกว่าครับ ทำการสร้างไฟล์ src/store.js กันก่อนเลยครับ จากนั้นทำการเพิ่ม code ดังนี้ครับ

```

import Vue from 'vue'
import Vuex from 'vuex'
import createPersistedState from 'vuex-persistedstate'

Vue.use(Vuex)

export default new Vuex.Store({
  strict: true,
  plugins: [
    createPersistedState()
  ],
  state: {
    token: null,
    user: null,
    isUserLoggedIn: false,
  },
  mutations: {
    setToken (state, token) {
      state.token = token
      state.isUserLoggedIn = !!token
    },
    setUser (state, user) {
      state.user = user
    }
  },
  actions: {
    setToken ({commit}, token) {
      commit('setToken', token)
    },
    setUser ({commit}, user) {
      commit('setUser', user)
    }
  }
})

```

สังเกตว่าผมใช้ vuex การที่ผมใช้ vuex นี้ล่ะจะทำให้ผมสามารถเข้าถึงโปรแกรม และ action ของ code ชุดนี้ได้ทุกที่ ทุกเวลาที่ผมต้อง แบบว่าผมจะเรียกใช้ตอนไหนก็ได้ โดย vuex ของผมจะจัดการให้นั่นเอง เป็นไปครับ เจ้มั้ยครับ ซึ่งตัวแปรที่ผมพูดถึงว่าผมเข้าถึงได้ คือ

```
state: {  
    token: null,  
    user: null,  
    isLoggedIn: false,  
},
```

นั่นเอง นี่ในส่วนที่เค้าเรียกว่าจัดการ state จริงก็คือตัวแปรของเรานั้นล่าครับ แต่ยังมีอะไรมากกว่านี้ ค่อยๆ ตามไปครับ ไม่ต้องไปหนานิยามอะไรกามายเดียวงานไม่เสร็จครับ

ใจไว้ว่า คือ mutations กระบวนการภายใน การเรียกใช้จะเรียกใช้ ผ่าน action ด้วยคำสั่ง dispatch ซึ่งเราจะเรียกผ่านจากภายนอกไฟล์ จริงๆ Vuex ทำอะไรได้อีกมาก แต่ผมมักจะเอาจัดการเรื่องการเก็บข้อมูลลง Local Storage เป็นหลัก เพราะง่ายต่อการบันทึกและอ่านค่า เช่น ข้อมูลผู้ใช้งาน หรือ สินค้าในตะกร้าสินค้า เพราะ Local Storage นี้เวลาเรา Refresh มันจะไม่หายไปไหนนั่นเองครับ ปิดและเปิดกลับมาใหม่มันก็จะยังอยู่ตามที่เรากำหนด expire ไว้ครับ เมื่อตอน cookies เลยครับ

เรียกใช้ Vuex.Store

เราสร้างไว้แล้วแต่ไม่ได้ใช้ก็เท่านั้น เราทำการเรียกใช้ store.js ของเราใน src/main.js ได้เลยครับ

ทำการแก้ code src/main.js ของเราเป็นดังนี้ครับ

```
// The Vue build version to load with the `import` command  
// (runtime-only or standalone) has been set in webpack.base.conf with  
// an alias.  
import Vue from 'vue'  
import App from './App'  
import router from './router'  
import { sync } from 'vuex-router-sync'  
import store from './store'  
import VueResource from 'vue-resource'
```

```
Vue.config.productionTip = false

Vue.use(VueResource)

sync(store, router)

/* eslint-disable no-new */
new Vue({
  el: '#app',
  router,
  store,
  components: { App },
  template: '<App/>'
})
```

โดยจาก code จะเห็นว่าผมทำการ

```
import { sync } from 'vuex-router-sync'
import store from './store'
```

เพื่อจะใช้งานร่วมกันโดยผมต้องการให้ store ของผมสามารถเรียกใช้งานได้ทุก url route โดยการ

```
sync(store, router)
```

แล้วก็เพิ่ม data ที่ชื่อว่า store ของเราใน Vue Object ของเราเพื่อให้ vue application ของเรารู้จัก this.\$store ของเราทุก Component นั่นเอง

```
new Vue({
  el: '#app',
  router,
  store,
  components: { App },
```

```
template: '<App/>'  
})
```

มากครับมาลองทดสอบกันดูว่า ที่เรารอ กอกแบบໄว้มันจะ work มั้ย ทำการเปิด src/components/Login.vue ของเราขึ้นมาครับ แล้วใส่ code เพิ่มเข้าไปดังนี้ครับ

```
methods: {  
  async onLogin () {  
    try {  
      const response = await AuthenService.login({  
        email: this.email,  
        password: this.password  
      })  
  
      this.$store.dispatch('setToken', response.data.token)  
      this.$store.dispatch('setUser', response.data.user)  
  
      console.log(response)  
  
    } catch (error) {  
      console.log(error)  
    }  
  }  
}
```

เห็นมั้ยผมเรียกใช้ this.\$store.dispatch ได้โดยไม่ต้องประกาศอะไรเลยครับ มันก็จะเก็บลง Local Storage ผ่าน store.js หรือ vuex.store ของเรานั้นเองครับ ทำการเปิด web browser ของเราขึ้นมาครับ แล้วไปที่

```
http://localhost:8080/#/Login
```

จากนั้นทำการ ใส่ email และ password จากนั้นทำการ login แล้วตรวจสอบจาก Dev Tool ของ browser ของเราที่ Application Local Storage นั้นใน มิงโก้ !!! token ของเราที่ส่งมาจาก Server ถูกเก็บไว้ใน Local Storage เรียบร้อยครับ

Key	Value
loglevel:webpack-dev-server	WARN
vuex	{ "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwiZW1haWwiOiJwZXRLcnh4QGdtYWlsIiwicGFzc3dvcnQiOiI..."}

ลองเปิดไปที่หน้าอื่น เช่น

<http://localhost:8080/#/users>

แล้วทำการ refresh สักสอง หรือสามครั้ง จะสังเกตว่า Token และ User ที่เราเก็บไว้ ยังอยู่ไม่หายไปครับ แล้วมันอยู่นานแค่ล่ะ ขึ้นอยู่กับ Server ตอนที่เราสร้าง jwt token ผ่านแอบไม่บอกเอาไว้ให้สงสัย จะได้ไปลองไล่ดูครับ

The screenshot shows a browser window with multiple tabs open. The active tab is at `localhost:8080/#/users`, displaying the heading "Get all users" and a message "จำนวนผู้ใช้งาน 1". Below this, there is a button labeled "สร้างผู้ใช้งาน" (Create User). The user details shown are: id: 1, ชื่อ-นามสกุล: peterxxx - parkerxxx, and email: peterxx@gmail.com.

Below the browser, the Chrome DevTools Application tab is visible. It shows the storage state for the domain `http://localhost:8080`. In Local Storage, there is a key "vuex" with a value containing a token. In the Vuex store, there is a state object with properties: `isUserLoggedIn: true`, `route: {name: "users", path: "/users", hash: "", query: {}, params: {}, fullPath: "http://localhost:8080/users"}`, `token: "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwiZW1haWwiOiJwZXRlcnh4QGc...`, and `user: {id: 1, email: "peterxx@gmail.com", password: "1234xxx", name: "peterxxx", lastLogin: null}`.

เหมือนจะจบแต่ยังไม่จบนะครับ ขอบดีใจไปแล้ว .. เรายังไม่ได้ส่งค่า Token ของเราไปหา Server กันเลย ถ้าเราต้องมานั่งส่งทุก ๆ vue component ที่เราสร้างขึ้นคงไม่สนุก ผມทำการเปิด `src/Services/Api.js` ของผມขึ้นมาครับ จากนั้นทำการแก้ไข code เป็นดังนี้ครับ

```
import axios from 'axios'
import store from '@/store'

export default () => {
  return axios.create({
    baseURL: 'http://localhost:8081/',
    headers: {
      Authorization: `Bearer ${store.state.token}`
    }
})
```

อ้าว !!! เดี๋ยวก่อนครับ ทำไมต้องมา import store.js อีกแล้วครับ ไหนว่า ที่เดียว ใช้ได้หมดทุกที่ ใจครับ ที่ได้ทุกที่ คือ Vue Component ของเรารับไม่ใช่ .js ครับ ดังนั้นจะใช้ต้อง import ก่อนครับ

จากนั้นแก้ code โดย แนบ header token ไปหา server ทุกครั้งที่มีการเรียกใช้ ทุก service ของเราเลยครับ จะได้ไม่ต้องกังวลว่าจะลืมครับ นี่ล่ะครับเหตุผลที่ผม ใช้ axios ทำ service api เพื่อติดต่อกับ server ครับ

ทำความรู้จัก vue devtools

vue devtools คือ plugin ตัวนึงเราใช้ในการดูการทำงานต่าง ๆ ของ vue ไม่ว่าจะ เป็น data () ต่าง ๆ ของ vuejs application ที่เราเขียน ในแต่ละ components ของเรา เราจะไปใช้งานจริงตอนทดสอบกันนะจะดีกว่าครับ ตอนนี้ติดตั้งก่อน

สำหรับ chrome ไปที่ link นี้ได้เลยครับ

<https://chrome.google.com/webstore/detail/vuejs-devtools/nhdogjmejiglipccpnnnanhbledajbpd>

จากนั้นทำการกดติดตั้ง restart browser ของเราถ้าขึ้น logo vue เรียwa ๆ ที่มุมบน ขวาแสดงว่าพร้อมใช้งานแล้ว เสร็จแล้วกดที่ menu vue ด้านล่าง ใน browser dev tool ของเราเพื่อทำการใช้งานครับ

The screenshot shows a browser window at `localhost:8080/#/login`. The main content is a "User Login" form with fields for "Username" (set to `user1@gmail.com`) and "Password" (redacted). A "Login" button is below the fields. Above the form is a toolbar with icons for back, forward, search, and other browser functions. The "Vue" tab in the developer tools panel is highlighted with a red box.

Vue DevTools Panel:

- mutations:**
 - Base State (17:34:14)
 - setToken (17:38:32)
 - setUser (17:38:32) - This row is selected and highlighted in green.
- state:**

```

isUserLoggedIn: true
route: Object
token: "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6M...  

user: Object
  createdAt: "2019-03-17T07:35:05.241Z"
  email: "user1@gmail.com"
  id: 4
  lastname: "nakrub"
  name: "user"
  password: "12345678"
  status: "active"
  type: "admin"
  updatedAt: "2019-03-17T07:35:05.241Z"

```
- mutation:** (List of mutation types)

ตรวจสอบ Authen บน Server จาก Token ของ Client

ถึงตอนนี้เราได้ Token มาจาก Server และนำมารับ กำลังสนุกเลยครับ เรามาทำ Authen Permission บน Server กันครับ เพื่อกำหนดการเข้าถึงข้อมูลบน Server ของเราครับ

เปิดที่โฟล์เดอร์ในส่วน `Server` ของเรากันครับ ทำการสร้างไฟล์ `src/userPassport.js` ขึ้นมาครับ สร้างไว้ตามแน่งดังรูปเลยนะครับ จะได้ require ได้ง่าย

The screenshot shows the VS Code interface. On the left is the Explorer sidebar with icons for files, folders, and search. The tree view shows a project structure under 'NV-WEBBLOG': 'client' (expanded), 'server' (selected and highlighted with a red border), 'node_modules', 'src' (expanded), 'config', 'controllers', 'models', 'app.js', 'routes.js', and 'userPassport.js'. Below these are 'nvwebblog-db.sqlite', 'package-lock.json', and 'package.json'. On the right is the code editor window titled 'userPassport.js'. The code is a Node.js module using Passport.js and JWTStrategy. It includes imports for passport, User, JwtStrategy, ExtractJwt, and config. It sets up a 'user' strategy using the JwtStrategy with jwtFromRequest set to ExtractJwt.fromAuthHeaderAsBearerToken() and secretOrKey set to config.authentication.jwtSecret. It then defines an asynchronous function to handle jwtPayload, which finds a User by email and returns done(true) if found or done(new Error(), false) if not.

```
const passport = require('passport')
const {User} = require('./models')

const JwtStrategy = require('passport-jwt').Strategy
const ExtractJwt = require('passport-jwt').ExtractJwt

const config = require('./config/config')

passport.use('user',
  new JwtStrategy({
    jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
    secretOrKey: config.authentication.jwtSecret
  }, async function (jwtPayload, done) {
    try {
      const user = await User.findOne({
        where: {
          email: jwtPayload.email
        }
      })

      if (!user) {
        return done(new Error(), false)
      }

      done(null, user)
    } catch (err) {
      done(err)
    }
  })
)
```

จากนั้นทำการเพิ่ม code ดังนี้ก่อนครับ ผู้จะอธิบายในลำดับต่อไปครับ

```
const passport = require('passport')
const {User} = require('./models')

const JwtStrategy = require('passport-jwt').Strategy
const ExtractJwt = require('passport-jwt').ExtractJwt

const config = require('./config/config')

passport.use('user',
  new JwtStrategy({
    jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
    secretOrKey: config.authentication.jwtSecret
  }, async function (jwtPayload, done) {
    try {
      const user = await User.findOne({
        where: {
          email: jwtPayload.email
        }
      })

      if (!user) {
        return done(new Error(), false)
      }

      done(null, user)
    } catch (err) {
      done(err)
    }
  })
)
```

```
        }
        return done(null, user)
    } catch (err) {
        return done(new Error(), false)
    }
}

module.exports = null
```

code ส่วนนี้จะเป็นการเรียกใช้งาน passport เพื่อทำการตรวจสอบการ login ผ่าน passport-jwt ครับ โดยทำการเอาค่า Token ที่รับมาผ่าน jwt และถอดรหัสแล้วไปค้นหาใน User Model ของเรา จาก email ถ้าพบก็จะทำการ return user อกมา อกมาครับ โดยเราใช้งานมันผ่าน app.js เลยครับ

เปิด app.js ขึ้นมา เพิ่ม code ตามนี้เลยครับ เอาไว้ก่อน require routes ของเราจะครับ

```
require('./userPassport')

require('./routes')(app)
```

จากที่เราเขียน ๆ มาในหัวข้อนี้ตอนนี้ เครื่องของเราจะทำการปรีบเทียน user ของเรา กับ Token ที่ client ส่งมาได้แล้ว ผู้จะทำการสร้าง Controller อีกด้วยนึง เพราะใช้งานในส่วนนี้กันครับ

ทำการสร้างโฟล์เดอร์ และ ไฟล์ ชื่อว่า src/authen/isAuthenController.js กันครับ

```

.
+-- client
  +-- server
    +-- node_modules
      +-- src
        +-- authen
          +-- isAuthenController.js
        +-- config
        +-- controllers
        +-- models
      +-- app.js
      +-- routes.js
      +-- userPassport.js
      +-- nvwebblog-db.sqlite
    { package-lock.json
    { package.json
  
```

```

  4   /*console.log('*** user data ***')
  5   console.log(res)*/
  6   passport.authenticate('user','jwt', function (err, user) {
  7     //if (err || !user) {
  8     if (err || !user) {
  9       res.status(403).send({
 10         error: 'you do not have access to this resource'
 11       })
 12     } else {
 13       req.user = user
 14       next()
 15     }
 16   })(req, res, next)
 17 }
  
```

จากนั้นทำการเพิ่ม code ชุดนี้เข้าไปครับ

```

const passport = require('passport')

module.exports = function (req, res, next) {
/*console.log('*** user data ***')
console.log(res)*/
  passport.authenticate('user','jwt', function (err, user) {
    //if (err || !user) {
    if (err || !user) {
      res.status(403).send({
        error: 'you do not have access to this resource'
      })
    } else {
      req.user = user
      next()
    }
  })(req, res, next)
}
  
```

code ในส่วนนี้จะเป็นการตรวจสอบการ login ด้วย passport.js ซึ่งเป็น package ที่เราติดตั้งไว้ในตอนต้นนั่นเอง โดยจะทำงานร่วมกับ userPassport.js ของเรา เพื่อ ตรวจสอบ Token ที่ส่งมาจาก Client ว่าใช่ของเราและ หาผู้ใช้ในระบบได้มั้ย รวมถึงวันหมดอายุด้วยครับ

จากนั้นมาเรียกใช้งาน isAuthenController.js กันครับ เราไปเรียกใช้งานมันที่ router.js ครับ มองหา api หรือ path '/users' ของเรารับ จากนั้น แก้ไข code เป็นดังนี้ครับ

ทำการ import controller ของเราเพื่อใช้งาน

```
const isAuthenController = require('./controllers/isAuthenController')
```

จากนั้นทำการ mapping route ของเรา

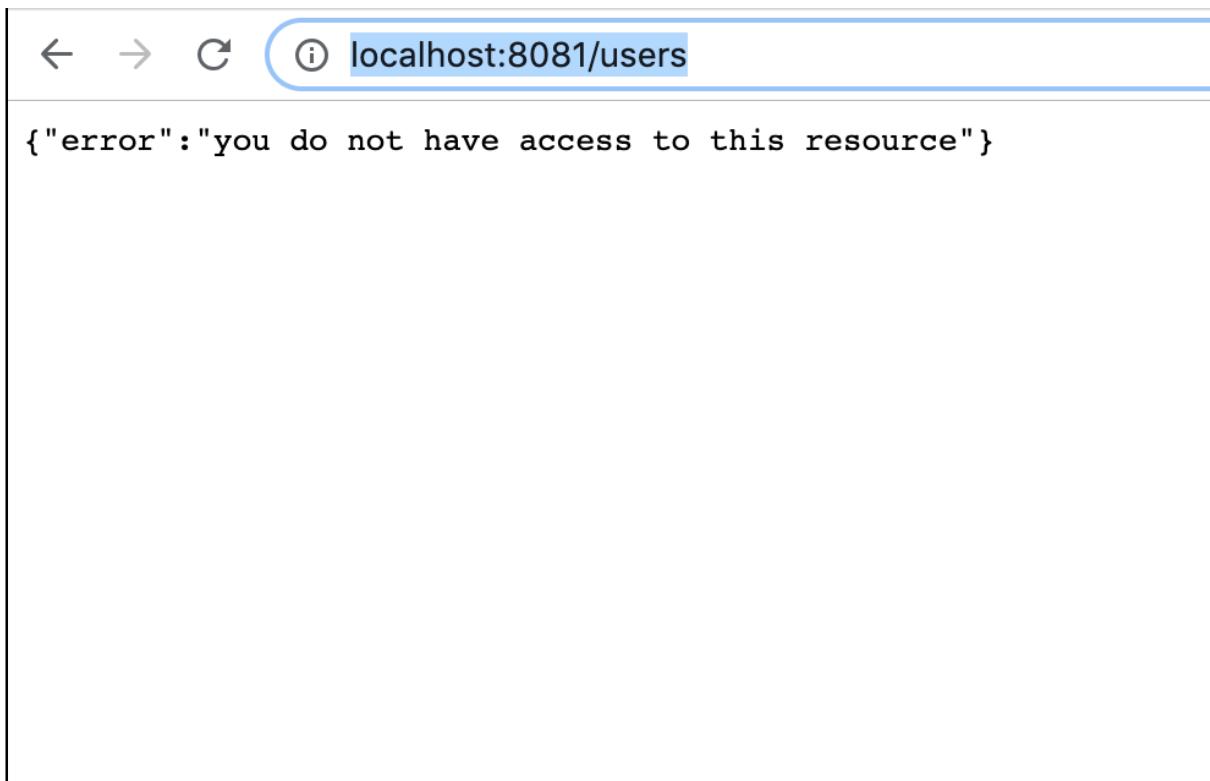
```
// get all user
app.get('/users',
  isAuthenController,
  UserController.index
)
```

โดยก่อนที่ผ่านไปหา UserController.index ของเรา ต้องผ่านการตรวจสอบ Token ก่อนนั่นเอง

เปิด web browser ของเราขึ้นมาครับ จากนั้นเรียกที่ url api ของเราโดยตรงเลย ครับ อย่าลืมนะครับ 8081 คือ port server นะครับ

```
http://localhost:8081/users
```

เราจะไม่มี permission ใน การเข้าถึงข้อมูลนะครับ เพราะว่าเราไม่ได้ส่ง Token ของเราไปด้วยนั่นเองครับ



A screenshot of a web browser window. The address bar at the top shows the URL `localhost:8081/users`. Below the address bar, the main content area displays the JSON response: `{"error": "you do not have access to this resource"}`. The browser interface includes standard navigation buttons (back, forward, refresh) and a search/address bar.

มาทดสอบกันต่อครับ เราเปิด web browser ของเราไปที่

`http://localhost:8080/#/users`

แล้วทำการลบข้อมูลใน Local Storage ของเรา กันครับ

localhost:8080/#/users

Get all users

จำนวนผู้ใช้งาน 1

สร้างผู้ใช้งาน

id: 1

ชื่อ-นามสกุล: peterxxx - parkerxxx

email: peterxx@gmail.com

password: 1234xxx

Application

Storage

Local Storage

Session Storage

IndexedDB

Web SQL

Cookies

vuex

loglevel:webpack-dev-server

WARN

จากนั้นกด refresh ครับ จะเห็นว่า Server ว่าจะไม่ส่งข้อมูลให้เราครับ แต่จะส่ง error response ตามที่เราเขียนไว้ อกมาแทนครับ

จำนวนผู้ใช้งาน 0

สร้างผู้ใช้งาน

Network

All

localhost

users

Preview

{error: "you do not have access to this resource"}
error: "you do not have access to this resource"

เลือก "localhost" หรือ "users"

ผมจะทำการเพิ่ม code ในส่วน login ในฝั่ง client โดยให้ login เสร็จแล้วถ้าไม่ติด error ได ๆ ให้ทำการวิงกลับไปที่หน้าหลักของการจัดการผู้ใช้งานของเรา คือ

<http://localhost:8080/#/users>

ผมทำการเปิด src/components/Login.vue ขึ้นมาแล้วเพิ่ม code ดังนี้ครับ

```
methods: {
  async onLogin () {
    try {
      const response = await AuthenService.login({
        email: this.email,
        password: this.password
      })

      this.$store.dispatch('setToken', response.data.token)
      this.$store.dispatch('setUser', response.data.user)
    }
  }
}
```

```

        this.$router.push({
          name: 'users'
        })

      } catch (error) {
        console.log(error)
      }
    }
}

```

เมื่อผู้ทำการ Login ถูกต้องแล้วผู้มีจะตีงข้อมูลจาก Server ของผู้ได้โดยไม่ติด Permission นั่นเองครับ หากคนอื่นมาตีงข้อมูลจากผู้มีจะตีงข้อมูลไม่ได้นั่นเองครับ

Login ได้แล้ว มาทำ ปุ่ม Logout กัน

หลังจากเรา Login และจัดการเรื่อง Permission ในเว็บไซต์ของเรา ก็ได้แล้ว เพื่อความสะดวกเราจะทำ ปุ่ม Logout กันจะได้ไม่ต้องไปค่อยลบ Local Storage กันให้เสียเวลา

ผู้เปิด src/components/Users/Index.vue ขึ้นมาแล้ว เพิ่มปุ่ม Logout เข้าไปดังนี้ครับ

```
<p><button v-on:click="logout">Logout</button></p>
```

จากนั้นผู้ไปเพิ่ม method logout ให้กับ component ของผู้ดังนี้ครับ

```

logout () {
  this.$store.dispatch('setToken', null)
  this.$store.dispatch('setUser', null)
  this.$router.push({
    name: 'login'
  })
},

```

ซึ่งจริงแล้ว methods นี้ไม่มีอะไรมาก่อน นอกจาก dispatch ค่า null ไปให้ store ของคุณแล้ว redirect ไปที่หน้า <http://localhost:8080/#/login> ของคุณเท่านั้นเอง

Login ไม่ผ่านแสดงผลกันหน่อย

จากที่ผ่านมาเราได้ตอบสนอง การ Login เฉพาะ Login ผ่าน แต่ถ้าไม่ผ่านตอนนี้ จะค้างอยู่ User ก็จะไม่รู้ว่าเกิดอะไรขึ้น เราไปเปิด Code ในส่วนของ api ที่เราเขียนไว้ บน Server ของเรา ก่อนครับว่า ถ้า Login และไม่ผ่าน เราส่งอะไรจาก Server มาให้ Client กัน โดยเปิดไฟล์ src/controllers/UserAuthenController.js ในส่วนของ server ขึ้นมา แล้วมองหาส่วนของ Login กันครับ จะเห็นว่ามี error ออกร่วมกับ

```
error: 'User/Password not correct'
```

อกมาทั้งสองกรณี ไม่ว่าจะ email หรือ password ผิดก็ตาม เพื่อป้องกัน random data แล้วมาเก็บ result ไปได้ประมาณนึง ซึ่งจริงต้องทำอีกนิดหน่อย ถึงจะ perfect แต่ใน class นี้เราจะเรียกกันแค่นี้ก่อนครับ

ตอนนี้เรารู้แล้วว่า Server ส่งอะไรมา ต่อมาเราจะเปิด src/components/Login.vue ในส่วนของ client ขึ้นมาครับ

พอทำการเพิ่ม ตัวแปร error เข้าไปที่ data ดังนี้ครับ

```
data () {
  return {
    email: '',
    password: '',
    error: null
  }
},
```

และทำการเพิ่ม code ในส่วนของ methods เวลาเกิด login error ของเราดังนี้ครับ

```
async onLogin () {
  try {
    const response = await AuthService.login({
      email: this.email,
      password: this.password
    })

    this.$store.dispatch('setToken', response.data.token)
    this.$store.dispatch('setUser', response.data.user)

    this.$router.push({
      name: 'users'
    })
  } catch (error) {
    console.log(error)
    this.error = error.response.data.error
    this.email = ''
    this.password = ''
  }
}
```

จะเห็นว่า client ของเราทำการเก็บ error.response ที่ server ส่งมาไว้ในตัว
แปร error ใน data ของเราเอง

จากนั้นเราทำการแสดงผลให้หน้า login ของเราเวลาเกิด error ดังนี้ครับ

```
<div>
  <h1>User Login</h1>
  <form v-on:submit.prevent="onLogin">
    <p>Username: <input type="text" v-model="email" /></p>
    <p>Password: <input type="password" v-model="password" /></p>
    <p><button type="submit">Login</button></p>
```

```
<div class="error" v-if="error">{{error}}</div>
</form>
</div>
```

จะเห็นว่า ผู้เพิ่ม class ให้มันเพื่อให้แสดงออกมาเป็น สีแดง จะได้เห็นชัด ๆ ตอน มัน error ครับ นั่นหมายความว่าผู้ต้องไปเพิ่ม class style ให้มันด้วยนั่นเอง ผู้ เพิ่มได้ดังนี้ครับ

```
<style scoped>
  .error {
    color:red;
  }
</style>
```

จากนั้นผู้ทำการทดสอบโดยการใส่ login ผิดเข้าไปผลจะเป็นดังนี้ครับ

ผู้เพิ่มส่วนในการเมื่อ login สำเร็จแล้ว ให้ทำการ redirect ไปที่หน้า User Index ของเราโดยการเพิ่ม code ดังนี้ เข้าไปที่ function login

```
this.$router.push({
  name: 'users'
})
```

← → ⌛ i localhost:8080/#/login

User Login

Username:

Password:

Login

User/Password not correct

ปลอดภัยขึ้นอีกนิดโดยการ Encrypt Password

ที่ผ่านมาเราได้ทำการ Authen ได้แล้วแต่ความปลอดภัยของเราอยังไม่ดีนั่น เราจะยกระดับความปลอดภัยของเราให้มากขึ้นด้วยการ Hash หรือเข้ารหัส password ของเรา กันครับ

ในบทที่ผ่าน ๆ มา เราได้ทำการติดตั้ง package ที่จำเป็นไว้หมดแล้ว ในส่วนนี้เราทำการเข้ารหัสส่วนของ password กันครับ หรือหากใครอยากให้ปลอดภัยมากกว่านั้นจะทำการเข้ารหัสในฟล็อก อีก 1 ด้วยก็สามารถทำได้ แต่ในหนังสือเล่มนี้ จะทำการเข้ารหัสเพียง password เท่านั้นเพิ่มตัวอย่างในการใช้งานเบื้องต้นให้ดูครับ

ทำการเปิดไฟล์ src/models/User.js หรือ User Model ของเราระบบมา แล้วทำการแก้ไข code ดังนี้

```

const Promise = require('bluebird')
const bcrypt = Promise.promisifyAll(require('bcrypt-nodejs'))

function hashPassword (user, options) {
  const SALT_FACTOR = 8

  if (!user.changed('password')) {
    return
  }

  return bcrypt
    .genSaltAsync(SALT_FACTOR)
    .then(salt => bcrypt.hashAsync(user.password, salt, null))
    .then(hash => {
      user.setDataValue('password', hash)
    })
}

module.exports = (sequelize, DataTypes) => {
  const User = sequelize.define('User', {
    email: DataTypes.STRING,
    password: DataTypes.STRING,
    name: DataTypes.STRING,
    lastname: DataTypes.STRING,
    status: DataTypes.STRING
  }, {
    hooks: {
      beforeCreate: hashPassword,
      beforeUpdate: hashPassword
    }
  })

  User.prototype.comparePassword = function (password) {
    return bcrypt.compareSync(password, this.password)
  }

  User.associate = function (models) {}

  return User
}

```

จาก code ที่เราเขียนเราได้เรียกใช้งาน pacakge เพิ่มขึ้นสองตัว คือ

```
const Promise = require('bluebird')
const bcrypt = Promise.promisifyAll(require('bcrypt-nodejs'))
```

โดย bluebird ของผมจะช่วยให้ bcrypt-nodejs ของผมทำงานแบบ sync หรือพูดให้ง่ายขึ้นกว่านั้น ตอนที่ผม hash หรือเข้ารหัส password นั้นมันจะรอมเข้ารหัสเสร็จก่อน เมื่อตอน async และ await ที่เราเคยใช้กันแล้ว แต่ผมผูกมันกับ bcrypt-nodejs ไว้ก่อนแล้ว .then() ของผมเลยการทำงานแบบ synchronize ไปโดยปริยาย ไม่ว่าจะในไปก่อนในตอนที่ผมเข้ารหัส

ใน sequelize ของเราสามารถทำการ hooks ได้เลย ถ้าใครไม่เข้าใจ คือกำหนดให้งานอัตโนมัติการเงื่อนไขที่เกิดขึ้น คือ

```
beforeCreate: hashPassword,
beforeUpdate: hashPassword
```

ซึ่งก่อนจะ created ให้ว่างไปทำ function hashPassword นั้นเอง ในรายละเอียด function ผมไม่ขออธิบายนะครับ ໄล ๆ ดูน่าจะเข้าใจ ใช้งานตามรูปแบบที่ผมเขียนได้เลยคับ

แต่ส่วนของ beforeUpdate มันไม่ทำงานนะครับ มันเป็น bug ของ version ใหม่ทาง sequelize บอกว่าจะแก้ ก็ไม่แก้ซะที ผมก็รอมเห็นอยแต่ไม่เป็นไร เดียวเราไปเขียนดักไว้เองได้ครับ

ตอนนี้เราจะทดลองสร้าง User กันครับ โดย ไปที่

```
http://localhost:8080/#/user/create
```

จากนั้นทำการกรอกข้อมูลเพื่อ create user กัน ครับเมื่อเราสร้าง user เสร็จแล้ว
เราจะเห็นว่า Password ของเราถูกเข้ารหัสไว้เรียบร้อยครับ

<p>id: 7</p> <p>ชื่อ-นามสกุล: admin - 12345678</p> <p>email: admin@gmail.com</p> <p>password: \$2a\$08\$7TMrtM2GrF7ghFA1RXn8euR2wjc/dXI9mWcMDTPbnukhC9TpMguQO</p> <p><input type="button" value="ดูข้อมูลผู้ใช้"/> <input type="button" value="แก้ไขข้อมูล"/> <input type="button" value="ลบข้อมูล"/></p>
<p>id: 8</p> <p>ชื่อ-นามสกุล: user1 - lastnameuser1</p> <p>email: user1@gmail.com</p> <p>password: \$2a\$08\$eT7e96tzcpduHC5sLMcFUOEPfRPSIolon0m9rWKD/eWV156ETIWIC</p> <p><input type="button" value="ดูข้อมูลผู้ใช้"/> <input type="button" value="แก้ไขข้อมูล"/> <input type="button" value="ลบข้อมูล"/></p>

ตอนนี้เราได้ทำการเข้ารหัส Password ของเราไว้เรียบร้อยแล้ว ถึงตอนนี้ ระบบ
ของเราปลอดภัยขึ้นมาอีกระดับล่ะครับ

ในเมื่อเราทำการสร้าง User พร้อมทั้งเข้ารหัสแล้ว เราจะทำการ Login ดูนะครับ
ถ้าสามารถ login ได้ ก็เป็นอันเสร็จพิธีครับ โดย code ที่ตรวจสอบ password ของ
เราคือ code ชุดนี้ นั่นเอง ซึ่งใน package ที่เราติดตั้งจะมี function ในการ
เปรียบเทียบมาให้เราใช้เลยครับ

```
User.prototype.comparePassword = function (password) {  
    return bcrypt.compareSync(password, this.password)  
}
```

บทที่ 10 เริ่มจัดระเบียบ Front End ให้เป็นลั๊ดส่วนกัน

ก่อนอื่นเลย เราจะมาจัดการ url ต่างของเราให้เป็นระบบ และมี layout ใกล้เคียงกับที่เราจะใช้งาน จริงก็เป็น layout ง่ายล่าครับ ผมจะเริ่มทำ navbar ก่อน

navbar ในส่วนของ Back Office ซึ่งอยู่ในส่วน Front End ของเรา ผมจะทำให้มี เมนูหลัก 4 เมนู คือ

1. Blogs
2. Users
3. Comments
4. Login
5. Logout

ในส่วน Front Office หรือส่วนที่ผู้อ่านจะเข้ามาอ่าน Blog ของเรา จะอยู่ในส่วนของ Front End เมื่อกัน โดยเราจะทำเมนูเดียว คือ blog โดย จะแสดง list ของ blog ที่เราเขียน และให้ user ทำการอ่านพร้อมทั้ง comment ได้ โดยหนังสือเล่มนี้ จะเปิดให้เป็น public comment ให้สามารถ comment ได้เลย และจะมี แบบฝึกหัดให้ทำ และมาทำการเฉลย ด้วย code ตัวอย่างภายหลัง

เมื่อเรารอกแบบ navbar ของเราแล้วเรามาเริ่มทำ navbar ในส่วนของ Back Office กันก่อนเลยครับ

ทำการสร้างไฟล์ src/components/Header.vue ขึ้นมา

```

2
3 </template>
4 <script>
5 export default {
6
7 }
8 </script>
9 <style scoped>
10
11 </style>
12
13
14

```

สร้างไว้ที่นี่เลยนะครับ จากนั้นทำการแก้ไข code ของเราดังนี้

```

<template>
  <div>
    <ul>
      <li><router-link :to="{name: 'blogs'}" >Blogs</router-link></li>
      <li><router-link :to="{name: 'users'}" >Users</router-link></li>
      <li><router-link :to="{name: 'comments'}" >Comments</router-link></li>
      <li><router-link :to="{name: 'login'}" >Login</router-link></li>
      <li><router-link :to="{name: 'blogs'}" >Blogs</router-link></li>
    </ul>
  </div>
</template>
<script>
export default {
}
</script>
<style scoped>
</style>

```

จะเห็นว่าใน tag ของเราจะไม่ใช้ tag <a> เมื่อตอน html ทั่วไป แต่เราจะใช้

```
<router-link :to="{name: 'users'}" >Users</router-link>
```

มาจัดการ โดยส่ง ชื่อ route ของ path ของเราไปเพื่อทำการ Vue Component ของเราเอง ลองดู code ชุดนี้นะครับ ให้ดูเปรียบเทียบเลย ๆ นะครับ ไม่ต้องไปใส่ที่ไหนนะครับ

```
{
  path: '/users',
  name: 'users',
  component: UserIndex
},
```

สังเกตว่า path ของมันคือ '/users' และ name ของมันคือ 'users' ในการใช้งาน tag <router-link></router-link> นั้น เราจะทำการส่ง name ที่ตรงกันไปครับ

ตอนนี้เราสร้าง Vue Component Header ของเราขึ้นมาแล้ว ต่อมาเราจะทำการเรียกใช้กันครับ ก่อนอื่นเราเปิด src/main.js และทำการเพิ่ม code ดังนี้ครับ

```
import BackHeader from '@/components/Header.vue'

Vue.component('back-header', BackHeader)
```

ซึ่งคือ เรา import Vue Component Header ของเราเข้ามา และบอก Vue ของเราว่า อันนี้คือ component ของเราจะ โดยบอกถึง ชื่อ tag ที่เราอ้างอิงด้วย คือ back-header นั้นเอง จากนั้นก็จะสามารถเรียกใช้ tag <back-header> ที่ไหนใน Vue Component อีน ๆ ได้ทันที

เปิด src/components/App.vue ของเราขึ้นมา แล้วทำการแก้ไข code ในส่วน template ของเราเป็นดังนี้ครับ

```
<template>
<div id="app">
  <back-header />
  <router-view/>
</div>
</template>
```

เห็นมั้ยครับ เราทำการเรียกใช้ tag <back-header /> ของเราได้เลย ไม่ต้องไป import อะไรมาเลยครับ ที่เราแปะ navbar ของเราไว้ที่ App.vue เพราะว่าทุก Vue Component ของเราจะมา Render ที่นี่ครับ

เมื่อเราเปิด web browser แล้วเรียกไปที่

```
http://localhost:8080/#/users
```

จะเห็นว่า navbar ของเราขึ้นมาแล้วนั่นเองครับ

The screenshot shows a web browser window with the URL `localhost:8080/#/users`. The page content includes:

- A navigation bar with the following links:
 - [Blogs](#)
 - [Users](#)
 - [Comments](#)
 - [Login](#)
 - [Blogs](#)
- A main section with the heading **Get all users**.
- A button labeled **Logout**.
- The text **จำนวนผู้ใช้งาน 3**.

ตอนนี้ navbar ในส่วนของ back office ของเราแล้วนะครับ แต่เห็นแล้วอาจน่าหงุดหงิดไปหน่อย แต่เราจะปล่อยไปก่อนครับ ให้ engine เราทำงานได้ก่อน มีเวลาจะเก็บให้สวยงามแค่ไหนก็ได้ครับ

จากที่เราทำ navbar ตอนนี้ link ของเรา work แค่เพียง users และ login เท่านั้นเอง เราจะทำการสร้าง Vue Component สำหรับ Blogs และ Comments ทำเหมือนตอนเราสร้าง Users Vue Component เลยครับ

สร้างโฟล์เดอร์ Blogs ใน src/components และสร้าง

```
src/components/Blogs/Index.vue  
src/components/Blogs/CreateBlog.vue  
src/components/Blogs/EditBlog.vue  
src/components/Blogs>ShowBlog.vue
```

แล้วเราก็สร้าง template ง่ายโดยบอก ว่าหน้านี้จะใช้ในไฟล์

```
src/components/Blogs/Index.vue
```

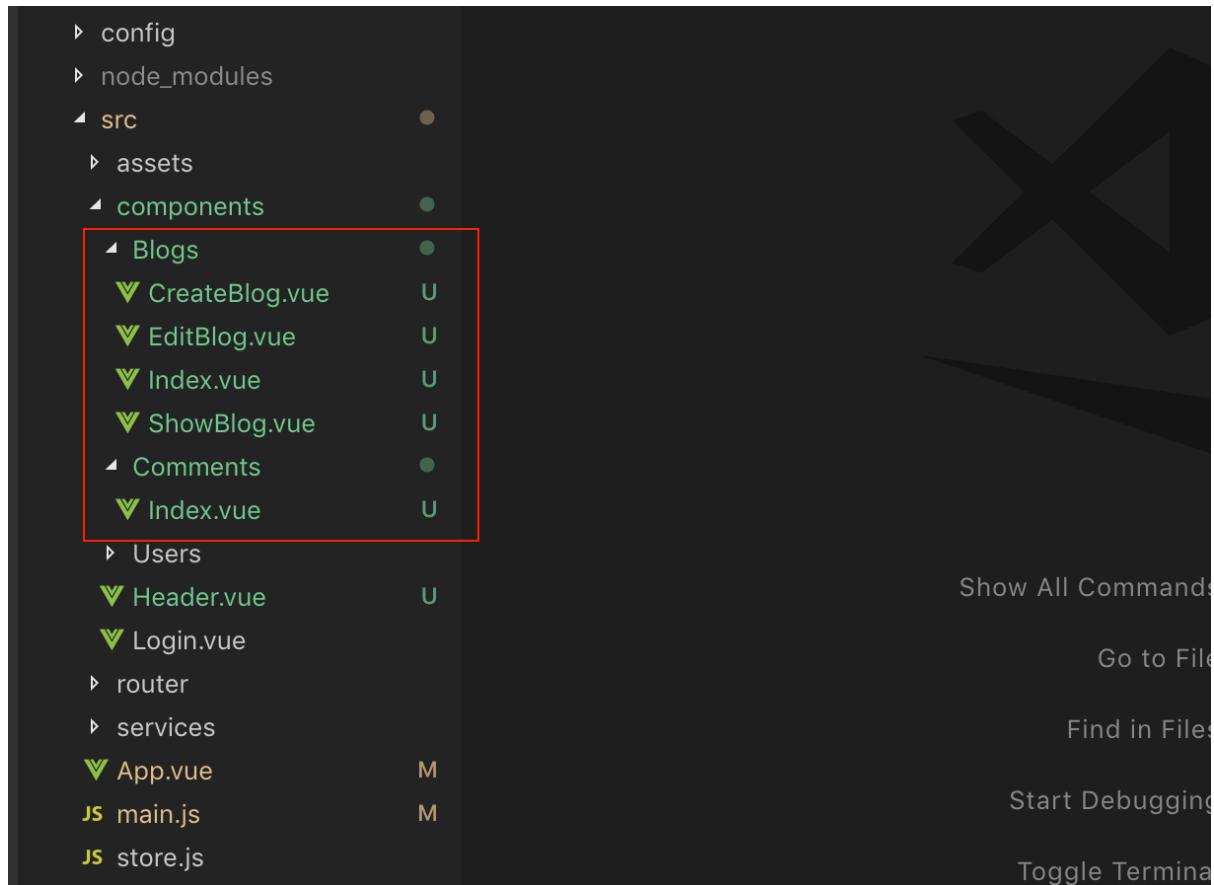
```
<template>  
  <div>  
    <h1>Blog Index</h1>  
  </div>  
</template>  
<script>  
export default {  
}  
</script>  
<style scoped>  
</style>
```

ไฟล์อีนกิทำการแก้ไข

ที่

```
<h1>Blog Index</h1>
```

ให้บอกถึงสถานะพ่อว่ามาถึงนี่แล้วนะ อะไรประมาณนั้น ใน Comments ก็ทำ
เหมือนกันครับ แต่ผมจะไม่ทำให้ดู แต่ดูจากภาพด้านล่างนะครับ



จากนั้น เราทำการแก้ไข src/router.js ใน client ของเรา กันนะครับ ให้ทำการวิ่งไป
ที่ component ที่เรากำหนดไว้ เปิด src/router.js ขึ้มมาแล้ว แล้วทำการเพิ่ม Vue
Component ที่เราสร้างไว้ดังนี้ครับ

```
// Comments
```

```
import CommentIndex from '@/components/Comments/Index'

// Blogs
import BlogIndex from '@/components/Blogs/Index'
import BlogCreate from '@/components/Blogs/CreateBlog'
import BlogEdit from '@/components/Blogs/EditBlog'
import BlogShow from '@/components/Blogs>ShowBlog'
```

ต่อมาเราจะทำการเพิ่ม route path ให้เว็บไซต์ของเรา วิ่งไปตามที่เรากำหนดครับ
ทำการแก้ไข code โดยเพิ่มเติม route path ของผู้ ไว้ใน routes: [] ดังนี้ครับ

```
// blogs
{
  path: '/blogs',
  name: 'blogs',
  component: BlogIndex
},
{
  path: '/blog/create',
  name: 'blogs-edit',
  component: BlogCreate
},
{
  path: '/blog/edit/:blogId',
  name: 'blog-edit',
  component: BlogEdit
},
{
  path: '/blog/:blogId',
  name: 'blog',
  component: BlogShow
},

// comments
{
  path: '/comments',
  name: 'comments',
  component: CommentIndex
},
```

จากนั้นทำการทดสอบ route path ต่าง ๆ ที่เราสร้างขึ้น โดย เปิด web browser และทำการเรียกไปที่ url ต่าง ๆ ของเราดังนี้ครับ

```
http://localhost:8080/#/blogs  
http://localhost:8080/#/blog/1  
http://localhost:8080/#/blog/edit/1  
http://localhost:8080/#/blog/create  
  
http://localhost:8080/#/comments
```

ถ้าเราทำอะไรไม่ผิด จะต้องแสดงหน้าเว็บไซต์ตามที่เราเขียนกันไว้ในตอนที่ผ่านมาครับ ถ้าผิดไม่ต้องกังวล เช็คจาก code ของ pm ในส่วนนี้ได้เลยครับ

จากนั้นเราทดสอบโดยการ กด navbar ของเรา ที่เราสร้างไว้กันครับ สังเกตว่าจะเข้าได้หมดแล้วไม่ error และวนะครับ

เอา # ใน url ของเราออก

ตอนนี้ url ของเราจะมี # อยู่ใน url ด้วย ซึ่งเค้าเรียกว่าการ route แบบ hash mode ซึ่งเค้าจะเอาไว้แสดง เพื่อขึ้นระหว่าง baseURL ของเรา กับ route path ที่เราเพิ่มเติมเข้าไป แต่ในการทำงานจริงเราจะเอาออกตอนท้าย ซึ่งเป็น url ปกติของเรามิ่ง # นั่นเอง ซึ่งเค้าเรียกว่า history mode โดยเราเพิ่ม code เข้าไปใน router/index.js ของเราดังนี้ครับ

```
export default new Router({  
  mode: 'history',
```

จากนั้นเปิด web browser ของเราขึ้นมา และเข้าที่ url ดังนี้

```
http://localhost:8080/users
```

จะได้ผลเหมือนกับการ # ก่อนหน้านี้ เรียนร้อยครั้ง history mode ของเรา work แล้วครับ หายหุด Heidi เลี้ยวนะครับ

ปรับ style ของ navbar ของเรา

เรามาปรับ style ของ navbar กันหน่อยครับ เพื่อที่จะได้ดูดีขึ้นมาอีกหน่อยครับ เรา เปิด src/components/Header.vue ขึ้นมาครับ และทำการแก้ไข code ใน template ของเราดังนี้ครับ

```
<template>
  <div>
    <div class="nv-navbar">
      <ul class="nav">
        <li><router-link :to="{name: 'blogs'}" >Blogs</router-link></li>
        <li><router-link :to="{name: 'users'}" >Users</router-link></li>
        <li><router-link :to="{name: 'comments'}" >Comments</router-link></li>
        <li><router-link :to="{name: 'login'}" >Login</router-link></li>
        <li><router-link :to="{name: 'blogs'}" >Blogs</router-link></li>
      </ul>
      <div class="clearfix"></div>
    </div>
  </template>
```

แล้วทำการเพิ่ม code ใน tag <style> ของเราดังนี้ครับ

```
<style scoped>
  .nv-navbar {
    background-color:palegoldenrod;
    width: 100%;
    padding:10px 0px 10px 0px;
```

```

}

.nv-navbar .nav {
    list-style: none;
    margin:0;
    padding:0;
    float:left;
}

.nv-navbar .nav li {
    float:left;
}

.nv-navbar .nav li a {
    padding: 10px;
    text-decoration: none;
    color:gray;
    font-weight: bold;
}

.nv-navbar .nav li a:hover {
    padding: 10px;
    text-decoration: none;
    color:darkslategrey;
}

.nv-navbar .nav li a.router-link-active {
    background-color:gold;
    color:darkslategrey;
}

.clearfix {
    clear: left;
}

</style>

```

ตอนนี้เริ่มใส่ style หรือ css ให้กับหน้าเว็บไซต์ของเรากันบ้างแล้ว ผู้จะค่อยๆ อธิบาย ส่วนที่สำคัญนะครับ ส่วนพื้นฐานด้าน css สามารถศึกษาได้จากที่นี่ครับ

<https://www.w3schools.com/css/default.asp>

ในส่วนของ navbar ของเราจะมีส่วนที่นำส่งไป คือ ส่วนที่เราใช้ router-link ใน การทำ link ต่าง ๆ ในส่วน navbar ของเรา แทน tag <a> ซึ่งจริงแล้วมันก็ไปทำการสร้าง tag <a> ให้เราเองนั่นล่ะ เปิด Dev Tool ของ Web Browser ของเราขึ้นมาครับ จิ้มไปที่ navbar ของเรา และดูที่ html element จะเห็นว่ามันคือ tag <a> นั่นเอง

ซึ่งเวลาเราเลือก link ไหน router-link ของเราจะสร้าง css class .router-link-active ให้เราอัตโนมัติ เราใช้ class นี้ลากรับมาจัดการ active ของ navbar ของเราครับ

Blogs Users Comments Login Logout

Get all users

Logout

จำนวนผู้ใช้งาน 3

สร้างผู้ใช้งาน

id: 1

ชื่อ-นามสกุล: peterxxx - parkerxxx

email: peterxx@gmail.com

password: 1234xxx

ดูข้อมูลผู้ใช้ แก้ไขข้อมูล ลบข้อมูล

ตอนนี้ navbar ของเราก็จะดูดีขึ้น แต่เราทำใช้ตอนพัฒนาเนย ๆ เครับไม่ต้องไปกังวล เราจะไปใช้งานจริงเรื่องหน้าตาเว็บไซต์ของเราตอนท้าย ๆ ด้วย bootstrap กันครับ

บทที่ 11 เริ่มต้นทำส่วน Blog

ในส่วนนี้เราต้องการให้สามารถสร้าง blog ของเราระบบให้ผู้ใช้งานเข้ามาอ่านได้ นั่นเองครับ เราสามารถนำความรู้จาก การทำส่วนของ user หรือผู้ใช้งานของเรา มาสร้างได้เลย แบบจะเหมือนกันเลย ง่ายกว่าด้วยซ้ำครับ ผมเลยจะอธิบายไม่ ละเอียดมากนักในส่วนนี้ เพื่อเปิดโอกาสให้ผู้อ่านได้ลงมือทดสอบเขียนเองบ้าง หากติดขัดตรงไหน ส่วนที่พูดได้เหมือนเดิมครับ

ทำส่วน Back End สำหรับ Blog

ทุกครั้งที่ผมทำงานถ้าเป็นการทำเว็บไซต์คนเดียวเดียวหรือว่า full stack web development นั้นผมจะเริ่มทำจาก api ของเราระบบ ก่อน โดยผมจะมี work flow ดังนี้ครับ

1. สร้าง Model
2. สร้าง Controller
3. เชื่อมต่อกับ Route
4. ทดสอบ

ตอนนี้เรามาทำการสร้าง Blog Model กันก่อนเลยครับ โดยสร้างไฟล์ใน server

ของเราว่า src/models/Blog.js และทำการเพิ่ม code ดังนี้ครับ

```
module.exports = (sequelize, DataTypes) => {
  const Blog = sequelize.define('Blog', {
    title: DataTypes.STRING,
    thumbnail: DataTypes.STRING,
    pictures: DataTypes.STRING,
    content: DataTypes.TEXT,
    category: DataTypes.STRING,
    status: DataTypes.STRING,
  })

  return Blog
}
```

จากนั้นผมทำการสร้าง src/controllers/BlogController.js ใน server ของผมแล้วทำการเขียน code ดังนี้ครับ

```
const {Blog} = require('../models')

module.exports = {
  // get all blog
  async index (req, res) {
    try {
      const blogs = await Blog.findAll()
      res.send(blogs)
    } catch (err){
      res.status(500).send({
        error: 'The blogs information was incorrect'
      })
    }
  },

  // create blog
  async create (req, res) {
    // res.send(JSON.stringify(req.body))
    try {
      const blog = await Blog.create(req.body)
      res.send(blog.toJSON())
    } catch (err) {
      res.status(500).send({
        error: 'Create blog incorrect'
      })
    }
  },

  // edit blog, suspend, active
  async put (req, res) {
    try {
      await Blog.update(req.body, {
        where: {
          id: req.params.blogId
        }
      })
      res.send(req.body)
    } catch (err) {
```

```

        req.status(500).send({
            error: 'Update blog incorrect'
        })
    },
}

// delete blog
async remove (req, res) {
    try {
        const blog = await Blog.findOne({
            where: {
                id: req.params.blogId
            }
        })

        if(!blog){
            return res.status(403).send({
                error: 'The blog information was incorrect'
            })
        }

        await blog.destroy()
        res.send(blog)
    } catch (err) {
        req.status(500).send({
            error: 'The blog information was incorrect'
        })
    }
},
}

// get blog by id
async show (req, res) {
    try {
        const blog = await Blog.findById(req.params.blogId)
        res.send(blog)
    } catch (err) {
        req.status(500).send({
            error: 'The blog information was incorrect'
        })
    }
}
}

```

จากนั้นเพิ่มทำการ แก้ src/route.js ใน server ของ pm โดยการเพิ่ม Controller ของ pm เข้าไป

```
const BlogController = require('./controllers/BlogController')
```

แล้วทำการ เพิ่ม path สำหรับ route ของ pm เข้าไปอีกดังนี้ครับ

```
// blog route
// create blog
app.post('/blog',
  BlogController.create
)

// edit blog, suspend, active
app.put('/blog/:blogId',
  BlogController.put
)

// delete blog
app.delete('/blog/:blogId',
  BlogController.remove
)

// get blog by id
app.get('/blog/:blogId',
  BlogController.show
)

// get all blog
app.get('/blogs',
  BlogController.index
)
```

เมื่อผมทำ api ในส่วนของ blog ทั้งหมดเสร็จแล้ว ผมจะทำการทดสอบ ด้วย POSTMAN ของผม ได้ผลเป็นที่เรียบร้อยดังนี้ครับ

POST http://localhost:8081/blog

No Environment

http://localhost:8081/blog

POST http://localhost:8081/blog Send

Params Authorization Headers (1) Body Pre-request Script Tests Cookies Code Com

Body (1) none form-data x-www-form-urlencoded raw binary JSON (application/json)

```

1 {
2   "title": "blog title",
3   "thumbnail": "null",
4   "pictures": "null",
5   "content": "content",
6   "category": "category",
7   "status": "status"
8 }

```

Body Cookies Headers (7) Test Results Status: 200 OK Time: 63 ms Size: 449 B

Pretty Raw Preview JSON

```

1 {
2   "id": 1,
3   "title": "blog title",
4   "thumbnail": "null",
5   "pictures": "null",
6   "content": "content",
7   "category": "category",
8   "status": "status",
9   "updatedAt": "2019-03-11T04:28:58.615Z",
10  "createdAt": "2019-03-11T04:28:58.615Z"
11 }

```

PUT http://localhost:8081/blog/1 Send

Params Authorization Headers (1) Body Pre-request Script Tests Cookies Code Com

Body (1) none form-data x-www-form-urlencoded raw binary JSON (application/json)

```

1 {
2   "title": "blog title put",
3   "thumbnail": "null put",
4   "pictures": "null put",
5   "content": "content put",
6   "category": "category put",
7   "status": "status put"
8 }

```

Body Cookies Headers (7) Test Results Status: 200 OK Time: 25 ms Size: 388 B

Pretty Raw Preview JSON

```

1 {
2   "title": "blog title put",
3   "thumbnail": "null put",
4   "pictures": "null put",
5   "content": "content put",
6   "category": "category put",
7   "status": "status put"
8 }

```

GET http://localhost:8081/blogs

Params Authorization Headers (1) Body Pre-request Script Tests

none form-data x-www-form-urlencoded raw binary JSON (application/json)

```
1 {  
2   "title": "blog title put",  
3   "thumbnail": "null put",  
4   "pictures": "null put",  
5   "content": "content put",  
6   "category": "category put",  
7   "status": "status put"  
8 }
```

Body Cookies Headers (7) Test Results Status: 204

Pretty Raw Preview JSON ↻

```
1 [  
2 {  
3   "id": 1,  
4   "title": "blog title put",  
5   "thumbnail": "null put",  
6   "pictures": "null put",  
7   "content": "content put",  
8   "category": "category put",  
9   "status": "status put",  
10  "createdAt": "2019-03-11T04:28:58.615Z",  
11  "updatedAt": "2019-03-11T04:29:42.851Z"  
12 }  
13 ]
```

GET http://localhost:8081/blog/1

Params Authorization Headers (1) Body Pre-request Script Tests

none form-data x-www-form-urlencoded raw binary JSON (application)

```
1 {  
2   "title": "blog title put",  
3   "thumbnail": "null put",  
4   "pictures": "null put",  
5   "content": "content put",  
6   "category": "category put",  
7   "status": "status put"  
8 }
```

Body Cookies Headers (7) Test Results Status:

Pretty Raw Preview JSON ↻

```
1 {  
2   "id": 1,  
3   "title": "blog title put",  
4   "thumbnail": "null put",  
5   "pictures": "null put",  
6   "content": "content put",  
7   "category": "category put",  
8   "status": "status put",  
9   "createdAt": "2019-03-11T04:28:58.615Z",  
10  "updatedAt": "2019-03-11T04:29:42.851Z"  
11 }
```

ทำหน้าบ้าน (Front End) ในส่วน Blog

ก่อนหน้านี้เราได้ทำ api ในส่วนของ Back End ไว้เสร็จแล้ว และทดสอบด้วย POSTMAN แล้วว่า Work ตอนนี้ เราจะมาทำในส่วน Front End ของ Blog ของเรากันครับ โดย Work Flow ในส่วน Front End ของผมจะมีดังนี้ครับ

1. สร้าง Service
2. สร้าง Vue Component
3. เชื่อมโยงด้วย Route

ทำการสร้าง Blog Service

ตอนนี้เราทำงานในส่วน client นะครับ อย่าลืม จะงง ๆ nidnengนะครับ เพราะทำทั้งสองส่วนเลย

เราเริ่มจากทำการสร้าง src/services/BlogsService.js และเขียน code โปรแกรมของเราดังนี้ครับ

```
import Api from '@/services/Api'

export default {
  index () {
    return Api().get('blogs')
  },
  show (blogId) {
    return Api().get('blog/'+blogId)
  },
  post (blog) {
    return Api().post('blog', blog)
  },
  put (blog) {
    return Api().put('blog/'+blog.id, blog)
  },
  delete (blog) {
    return Api().delete('blog/'+blog.id, blog)
  },
}
```

จากนั้นผมทำการแก้ไข code ใน Vue Component ของผมดังนี้

src/components/Blogs/Index.vue

```
<template>
<div>
  <h2>Get all blogs</h2>
  <p><button v-on:click="logout">Logout</button></p>
  <h4>ចាំនាម blog {{blogs.length}}</h4>
  <p><button v-on:click="navigateTo('/blog/create')">សរុប
blog</button></p>
  <div v-for="blog in blogs" v-bind:key="blog.id">
    <p>id: {{ blog.id }}</p>
    <p>title: {{ blog.title }}</p>
    <p>content: {{ blog.content }}</p>
    <p>category: {{ blog.category }}</p>
    <p>status: {{ blog.status }}</p>
    <p>
      <button v-on:click="navigateTo('/blog/'+ blog.id)">ទួរលើ
blog</button>
      <button v-on:click="navigateTo('/blog/edit/'+ blog.id)">កែតាំង
blog</button>
      <button v-on:click="deleteBlog(blog)">លើប្លើមូល</button>
    </p>
    <hr>
  </div>
</div>
</template>
<script>
import BlogsService from '@/services/BlogsService'

export default {
  data () {
    return {
      blogs: []
    }
  },
  async created () {
    this.blogs = (await BlogsService.index()).data
  },
}
```

```

methods: {
  logout () {
    this.$store.dispatch('setToken', null)
    this.$store.dispatch('setBlog', null)
    this.$router.push({
      name: 'login'
    })
  },
  navigateTo (route) {
    this.$router.push(route)
  },
  async deleteBlog (blog) {
    let result = confirm("Want to delete?")
    if (result) {
      try {
        await BlogsService.delete(blog)
        this.refreshData()
      } catch (err) {
        console.log(err)
      }
    }
  },
  async refreshData() {
    this.blogs = (await BlogsService.index()).data
  }
}
</script>
<style scoped>
</style>

```

src/components/Blogs/CreateBlog.Vue

```

<template>
  <div>
    <h1>Create Blog</h1>
    <form v-on:submit.prevent = "createBlog">
      <p>title: <input type="text" v-model="blog.title"></p>

```

```

<p>content: <input type="text" v-model="blog.content"></p>
<p>category: <input type="text" v-model="blog.category"></p>
<p>status: <input type="text" v-model="blog.status"></p>
<p><button type="submit">create blog</button></p>
</form>
</div>
</template>
<script>
import BlogsService from '@/services/BlogsService'

export default {
  data () {
    return {
      blog: {
        title: '',
        thumbnail: 'null',
        pictures: 'null',
        content: '',
        category: '',
        status: 'saved'
      }
    }
  },
  methods: {
    async createBlog () {
      try {
        await BlogsService.post(this.blog)
        this.$router.push({
          name: 'blogs'
        })
      } catch (err) {
        console.log(err)
      }
    }
  }
}
</script>
<style scoped>
</style>

```

src/components/Blogs/EditBlog.Vue

```

<template>
<div>
  <h1>Edit Blog</h1>
  <form v-on:submit.prevent = "editBlog">
    <p>title: <input type="text" v-model="blog.title"></p>
    <p>content: <input type="text" v-model="blog.content"></p>
    <p>category: <input type="text" v-model="blog.category"></p>
    <p>status: <input type="text" v-model="blog.status"></p>
    <p>
      <button type="submit">update blog</button>
      <button v-on:click="navigateTo('/blogs')">ກລົມ</button>
    </p>
  </form>
</div>
</template>
<script>
import BlogsService from '@/services/BlogsService'

export default {
  data () {
    return {
      blog: {
        title: '',
        thumbnail: 'null',
        pictures: 'null',
        content: '',
        category: '',
        status: ''
      }
    }
  },
  methods: {
    async editBlog () {
      try {
        await BlogsService.put(this.blog)
        this.$router.push({
          name: 'blogs'
        })
      } catch (err) {
        console.log(err)
      }
    }
  }
}

```

```

        },
        async created () {
            try {
                let blogId = this.$route.params.blogId
                this.blog = (await BlogsService.show(blogId)).data
            } catch (error) {
                console.log (error)
            }
        }
    }
</script>
<style scoped>
</style>

```

src/components/Blogs/ShowBlog.vue

```

<template>
    <div>
        <h1>Show Blog</h1>
        <p>id: {{ blog.id }}</p>
        <p>title: {{ blog.title }}</p>
        <p>content: {{ blog.content }}</p>
        <p>category: {{ blog.category }}</p>
        <p>status: {{ blog.status }}</p>
        <p>
            <button v-on:click="navigateTo('/blog/edit/' + blog.id)">ແກ້ໄຂ
            blog</button>
            <button v-on:click="navigateTo('/blogs')">ກລົມ</button>
        </p>
    </div>
</template>
<script>
import BlogsService from '@/services/BlogsService'

export default {
    data () {
        return {
            blog: null
        }
    }
}

```

```
},
async created () {
  try {
    let blogId = this.$route.params.blogId
    this.blog = (await BlogsService.show(blogId)).data
  } catch (error) {
    console.log (error)
  }
},
methods : {
  navigateTo (route) {
    this.$router.push(route)
  },
}
}
</script>
<style scoped>
</style>
```

ปกติถ้าผมเขียน code ในกรณีที่ไม่ได้สอนเขียนโปรแกรมอย่างนี้ ผมจะทำการ work ที่กล่าวมา แต่ในกรณีนี้เราได้ทำการ route blog component ของเราได้ หมดแล้ว ดังนั้นตอนนี้ ระบบของเราพร้อมจะเขียน และจัดการ blog ของเรา แบบ บ้าน ๆ ได้แล้วนั่นเองครับ

Get all blogs

[Logout](#)

จำนวน blog 2

[สร้าง blog](#)

id: 1

title: blog title put xx -

content: content put xx

category: category put xx

status: status put xx

[ดู blog](#)

[แก้ไข blog](#)

[ลบข้อมูล](#)

id: 2

Create Blog

title:

content:

category:

status:

[create blog](#)

Blogs Users Comments Login Logout

Edit Blog

title:

content:

category:

status:

ทำสร้าง User Comments

work flow ของผมจะทำ api ในส่วนของ Back End ก่อนเช่นเคยครับ เราเริ่มต้นทำ api ของเรา โดยทำ Comment Model กันก่อนครับ อย่าลืมนะครับ api ในส่วนของ Back End ของเราจะอยู่ใน Server นะครับ

โดยเราทำการสร้างไฟล์ src/models/Comment.js ขึ้นมาแล้วทำการเขียน code ในส่วนของ model ของเราดังนี้ครับ

```
module.exports = (sequelize, DataTypes) => {
  const Comment = sequelize.define('Comment', {
    blogId: DataTypes.STRING,
    comment: DataTypes.TEXT,
    userId: DataTypes.STRING
  })

  return Comment
}
```

จากนั้นเราไปทำการสร้าง ส่วนของ Coment Controller กัน โดยทำการสร้างไฟล์

src/controllers/CommentController.js และทำการเขียน code ดังนี้ครับ

```
const {Comment} = require('../models')

module.exports = {
  // get all comment
  async index (req, res) {
    try {
      const comments = await Comment.findAll()
      res.send(comments)
    } catch (err){
      res.status(500).send({
        error: 'The comments information was incorrect'
      })
    }
  },
  // create comment
  async create (req, res) {
    // res.send(JSON.stringify(req.body))
    try {
      const comment = await Comment.create(req.body)
      res.send(comment.toJSON())
    } catch (err) {
      res.status(500).send({
        error: 'Create comment incorrect'
      })
    }
  },
  // edit comment, suspend, active
  async put (req, res) {
    try {
      await Comment.update(req.body, {
        where: {
          id: req.params.commentId
        }
      })
      res.send(req.body)
    } catch (err) {
```

```

        req.status(500).send({
            error: 'Update comment incorrect'
        })
    },
}

// delete comment
async remove (req, res) {
    try {
        const comment = await Comment.findOne({
            where: {
                id: req.params.commentId
            }
        })

        if(!comment){
            return res.status(403).send({
                error: 'The comment information was incorrect'
            })
        }

        await comment.destroy()
        res.send(comment)
    } catch (err) {
        req.status(500).send({
            error: 'The comment information was incorrect'
        })
    }
},
}

// get comment by id
async show (req, res) {
    try {
        const comment = await Comment.findById(req.params.commentId)
        res.send(comment)
    } catch (err) {
        req.status(500).send({
            error: 'The comment information was incorrect'
        })
    }
}
}

```

เมื่อเราสร้างทั้ง Model และ Controller เสร็จแล้ว เราต้องทำการเชื่อมโยงกันด้วยการ routing เปิด src/routes.js ของเรามา แล้วทำการ import controller ของเราเข้ามา

```
const CommentController = require('./controllers/CommentController')
```

จากนั้นทำการ ใส่ path เพื่อเรียกใช้งาน Controller ของเราครับ

```
// comment route
// create comment
app.post('/comment',
  CommentController.create
)

// edit comment, suspend, active
app.put('/comment/:commentId',
  CommentController.put
)

// delete comment
app.delete('/comment/:commentId',
  CommentController.remove
)

// get comment by id
app.get('/comment/:commentId',
  CommentController.show
)

// get all comment
app.get('/comments',
  CommentController.index
)
```

เพราทำ api ในส่วนของ Back End ของเราเสร็จแล้ว เราต้องทำการทดสอบกัน
ครับ ผมทำการทดสอบด้วย POSTMAN เมื่อเดิมครับ

POST http://localhost:8081/user POST http://localhost:8081/comment + ... No Environment

http://localhost:8081/comment

POST http://localhost:8081/comment Send

Params Authorization Headers (1) Body ● Pre-request Script Tests Cookies Code

Body Type: JSON (application/json)

```

1 {
2   "blogId": "10",
3   "comment": "comment testing"
4 }
```

Body Cookies Headers (7) Test Results Status: 200 OK Time: 59 ms Size: 294 B

Pretty Raw Preview JSON ↻

```

1 {
2   "id": 1,
3   "blogId": "10",
4   "updatedAt": "2019-03-11T06:38:37.405Z",
5   "createdAt": "2019-03-11T06:38:37.405Z"
6 }
```

http://localhost:8081/comment/1

PUT http://localhost:8081/comment/1 Send

Params Authorization Headers (1) Body ● Pre-request Script Tests Cookies Code

Body Type: JSON (application/json)

```

1 {
2   "blogId": "20",
3   "comment": "comment testing update"
4 }
```

Body Cookies Headers (7) Test Results Status: 200 OK Time: 23 ms Size: 294 B

Pretty Raw Preview JSON ↻

```

1 {
2   "blogId": "20",
3   "comment": "comment testing update"
4 }
```

http://localhost:8081/comments

GET http://localhost:8081/comments Send

Params Authorization Headers (1) Body Pre-request Script Tests Cookies Code C

Body (1) none form-data x-www-form-urlencoded raw binary JSON (application/json)

```
1 {  
2   "blogId": "20",  
3   "comment": "comment testing update"  
4 }
```

Body Cookies Headers (7) Test Results Status: 200 OK Time: 18 ms Size: 363 B

Pretty Raw Preview JSON ↻

```
1 [  
2   {  
3     "id": 1,  
4     "blogId": "20",  
5     "comments": null,  
6     "createdAt": "2019-03-11T06:38:37.405Z",  
7     "updatedAt": "2019-03-11T06:39:00.431Z"  
8   }  
9 ]
```

http://localhost:8081/comment/1

GET http://localhost:8081/comment/1 Send

Params Authorization Headers (1) Body Pre-request Script Tests Cookies Code C

Body (1) none form-data x-www-form-urlencoded raw binary JSON (application/json)

```
1 {  
2   "blogId": "20",  
3   "comment": "comment testing update"  
4 }
```

Body Cookies Headers (7) Test Results Status: 200 OK Time: 10 ms Size: 361 B

Pretty Raw Preview JSON ↻

```
1 {  
2   "id": 1,  
3   "blogId": "20",  
4   "comments": null,  
5   "createdAt": "2019-03-11T06:38:37.405Z",  
6   "updatedAt": "2019-03-11T06:39:00.431Z"  
7 }
```

ทำส่วน Comments ใน Back Office (Front End)

ในส่วนนี้ผมจะมาทำ Back Office ในส่วนของ comments ที่ให้ผู้ใช้ comment เข้ามายกันนะครับ โดยในระบบ หลังบ้านเราจะทำอะไรไม่ได้ นอกจาก แสดง list comment ที่ ผู้อ่านทำการ comment เข้ามาและลบ comment ได้ อย่างเดียว แก้ไขไม่ได้ หรือใครอยากรедакแก้ไขได้ ผมว่าตอนนี้คงเขียนเองได้แล้วล่ะครับ

work flow ในการทำงานของเราจะเหมือนเดิมเลยนะครับ ผมเริ่มต้นทำ service ก่อนครับ โดยสร้างไฟล์ src/services/CommentsService.js ใน client ของเรา และทำการเขียน code ของเราดังนี้ครับ

```
import Api from '@/services/Api'

export default {
  index () {
    return Api().get('comments')
  },
  show (commentId) {
    return Api().get('comment/' + commentId)
  },
  post (comment) {
    return Api().post('comment', comment)
  },
  put (comment) {
    return Api().put('comment/' + comment.id, comment)
  },
  delete (comment) {
    return Api().delete('comment/' + comment.id, comment)
  }
}
```

แล้วทำการแก้ไขไฟล์ใน src/components/Comments/Index.vue ดังนี้

```
<template>
<div>
  <h2>Get all comments</h2>
  <p><button v-on:click="logout">Logout</button></p>
  <h4>จำนวน comment {{comments.length}}</h4>
  <div v-for="comment in comments" v-bind:key="comment.id">
    <p>id: {{ comment.id }}</p>
```

```

<p>blog id: {{ comment.blogId }}</p>
<p>comment: {{ comment.comment }}</p>
<p>
    <button v-on:click="navigateTo('/comment/' + comment.id)">comment</button>
    <button v-on:click="deleteComment(comment)">ลบข้อความ</button>
</p>
<hr>
</div>
</div>
</template>
<script>
import CommentsService from '@/services/CommentsService'

export default {
  data () {
    return {
      comments: []
    }
  },
  async created () {
    this.comments = (await CommentsService.index()).data
  },
  methods: {
    logout () {
      this.$store.dispatch('setToken', null)
      this.$store.dispatch('setComment', null)
      this.$router.push({
        name: 'login'
      })
    },
    navigateTo (route) {
      this.$router.push(route)
    },
    async deleteComment (comment) {
      try {
        await CommentsService.delete(comment)
        this.refreshData()
      } catch (err) {
        console.log(err)
      }
    },
    async refreshData() {
      this.comments = (await CommentsService.index()).data
    }
  }
}

```

```
        }
    }
}
</script>
<style scoped>
</style>
```

จากนั้นเปิด web browser ของเราไปที่ <http://localhost:8080/comments>

บีบิ้งโก้ ส่วนจัดการ Comment ในส่วนของ Back office ของเราพร้อมใช้งานแล้ว
ครับ

The screenshot shows a web browser window with the URL `localhost:8080/comments` in the address bar. The page has a yellow header bar with navigation links: `Blogs`, `Users`, `Comments` (which is highlighted in yellow), `Login`, and `Logout`. Below the header, the main content area displays the heading `Get all comments`. Underneath it, there is a button labeled `Logout`. The main content area contains the text `จำนวน comment 1`. Below this, there is a button labeled `สร้าง comment`. Further down, the text `id: 1` and `blog id: 20` are displayed. At the bottom of the content area, there is a text input field followed by three buttons: `ดู comment`, `แก้ไข comment`, and `ลบข้อมูล`.

ทำการ Logout จาก navbar

เปิดไฟล์

```
src/components/Users/Index.vue  
src/components/Blogs/Index.vue  
src/components/Comments/Index.vue
```

ตามลำดับ แล้วมองหา code ปุ่ม logout และ method logout เพื่อทำการลบ code ออกครับ

code ปุ่ม logout จะมีหน้าตาอย่างนี้ครับ เจอแล้วลบออกครับ

```
<p><button v-on:click="logout">Logout</button></p>
```

code ในส่วนของ method จะมีดังนี้ครับ ค้นหาและทำการลบครับ

```
logout () {  
    this.$store.dispatch('setToken', null)  
    this.$store.dispatch('setComment', null)  
    this.$router.push({  
        name: 'login'  
    })  
},
```

จากนั้นทำการเปิด src/components/Header.vue ขึ้นมาครับ ทำการแก้ code เพื่อเมนู Logout ของเราทำงานได้ที่นี่เลยครับ โดยทำการแก้ link สำหรับ logout สำหรับ logout เพื่อเรียกใช้ function logout ของเราใน template

```
<li><a v-on:click.prevent="logout" href="#">Logout</a></li>
```

จากนั้นทำการเพิ่ม function ในการ logout ใน methods ของเราดังนี้

```
logout () {
    this.$store.dispatch('setToken', null)
    this.$store.dispatch('setUser', null)
    this.$router.push({
        name: 'login'
    })
}
```

เราจับส่วนนี้ไว้เพียงเท่านี้ก่อน เรื่อง login/logout เรา ลืม ๆ มันไปก่อน เดี๋ยวเราจะมาใส่ปิด permission กันตอนท้าย ๆ เราจะทำงานได้ง่ายกว่า ไม่ต้องมานั่ง loging/logout ให้เสียเวลา ตอนนี้เพียงแต่ทำการบันทึกก่อนครับ

บทที่ 12 ทำส่วนเขียน Blog และ Upload File รูปภาพ

ก่อนหน้านี้เราทำส่วนเขียน blog กันแล้ว แต่จริง ๆ แล้วควรจะนокกว่าเป็น ส่วนเพิ่มข้อมูล blog กันมากกว่า เพราะถ้าเขียนจริง จะต้องจัดการรูปแบบของ blog เราได้เหมือน Microsoft Word เล็ก ๆ เพื่อเขียนบทความของเรารับ

เริ่มต้นใช้งาน CKEditor

ในส่วนนี้เราจะใช้งาน CKEditor มาทำ Editor เพื่อเอาไว้เขียนและแก้ไข Blog ของเรากัน โดยก่อนอื่นเราต้องทำการติดตั้ง package vue-ckeditor2 กันก่อน เลยด้วยคำสั่ง

```
npm install --save vue-ckeditor2
```

หลังจากนั้นใน Code ที่ main.js ให้เพิ่มโค้ดดังนี้

→ ในที่ Client

```
import Vue from 'vue'
import VueEditor from 'vue-ckeditor2'
Vue.use(VueEditor)
```

จากนั้นทำการ

download

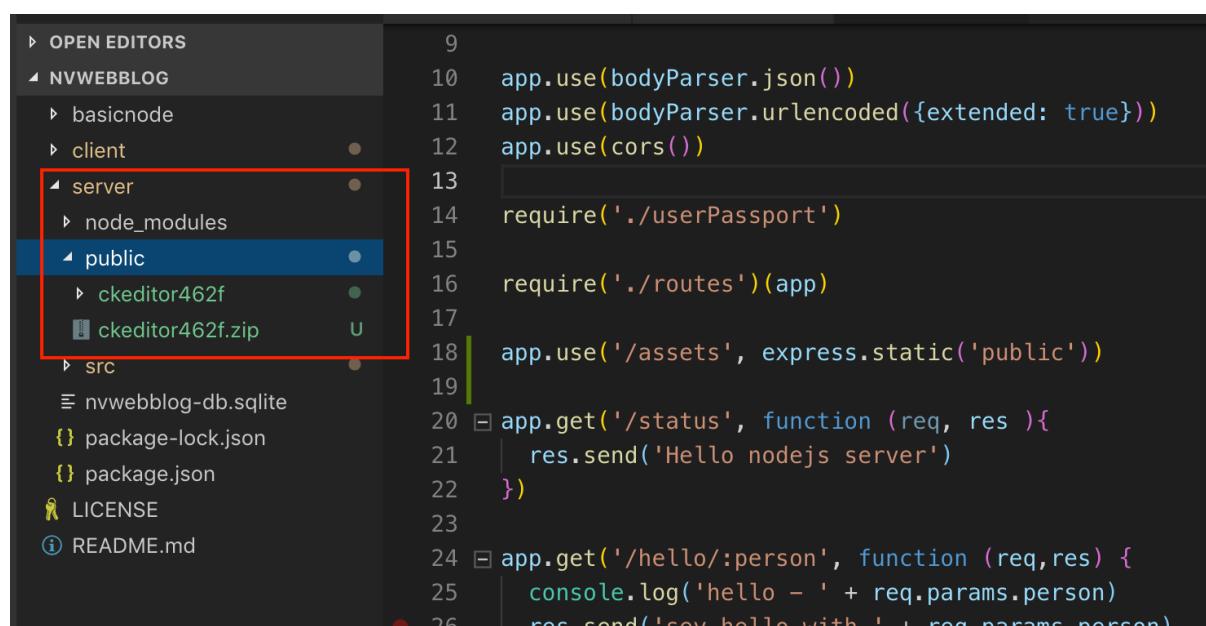
ไฟล์

https://drive.google.com/open?id=1sVtrYU3k2lcg6qr_1SOM11Z5iS4F0dl

M

↙ In Server

จากนั้นทำการสร้างโฟล์เดอร์ public จากนั้น extract และวาง CKEdit ของเราไว้ดังภาพ



```
OPEN EDITORS
NVWEBBLOG
basicnode
client
server
node_modules
public
ckeditor462f
ckeditor462f.zip
src
nvwebblog-db.sqlite
package-lock.json
package.json
LICENSE
README.md

9
10 app.use(bodyParser.json())
11 app.use(bodyParser.urlencoded({extended: true}))
12 app.use(cors())
13
14 require('./userPassport')
15
16 require('./routes')(app)
17
18 app.use('/assets', express.static('public'))
19
20 app.get('/status', function (req, res) {
21   res.send('Hello nodejs server')
22 })
23
24 app.get('/hello/:person', function (req, res) {
25   console.log('hello - ' + req.params.person)
26   res.send('say hello with ' + req.params.person)
```

จากนั้นเปิด src/app.js ใน server ของเราขึ้นมา แล้วใส่ code ดังนี้เข้าไป

```
app.use('/assets', express.static('public'))
```

โดย code ชุดนี้จะทำให้ public ของเรากลายเป็น static ไฟล์ หรือเข้าถึงได้แบบ public นั่นเอง เนื่องจาก ปกติแล้วใน nodejs server ของเราจะถูกป้องกันไม่ให้เข้าถึงจากภายนอก server ได้

จากนั้นทำการโหลด ckeditor เพื่อใช้งานโดยเพิ่ม script ที่ client/index.html ดังนี้

```
<script  
src="http://localhost:8081/assets/ckeditor462f/ckeditor.js"></script>
```

อันที่จริงแล้วเราสามารถเรียกใช้งาน CKEditor ผ่าน CDN ที่อยู่ก็สามารถทำได้แต่เนื่องจาก เราเป็นโปรแกรมเมอร์ บทความที่เราเขียนนั้นต้องใช้ Toolbar หลายตัว ผມเลยทำการ Build เป็น Library ไว้แล้วทำการเรียกใช้ผ่าน Server ของเรา นั่นเอง

จากนั้นทำการเปิด src/components/Blogs/CreateBlog.vue ของเราขึ้นมา

จากนั้นเริ่มเขียน code เพื่อใช้งาน ckeditor ของเราดังนี้ครับ เราเริ่มจากการแก้ไข code ใน template ของเราในส่วนของ content ของเราเป็นดังนี้

```
<p><strong>content: </strong></p>  
<p><vue-ckeditor v-model.lazy="blog.content" :config="config" @blur="onBlur($event)" @focus="onFocus($event)" /></p>
```

ตอนนี้เรามี code เรียกใช้งานใน HTML Template ของเราแล้ว ต่อมาเราทำการเขียน script เพื่อใช้งาน vue-ckeditor2 โดยการ

```
import VueCkeditor from "vue-ckeditor2"
```

จากนั้นทำการเพิ่ม code ใน data ของเรา ดังนี้ครับ

```
data () {  
    return {
```

↓ หาต้อง npm install --save
vue-ckeditor2
ที่ Client ต้อง import ยังไง

```
blog: {
    title: '',
    thumbnail: 'null',
    pictures: 'null',
    content: '',
    category: '',
    status: ''
},
config: {
    toolbar: [
        ["Bold", "Italic", "Underline", "Strike", "Subscript",
        "Superscript"]
    ],
    height: 300
},
},
},
```

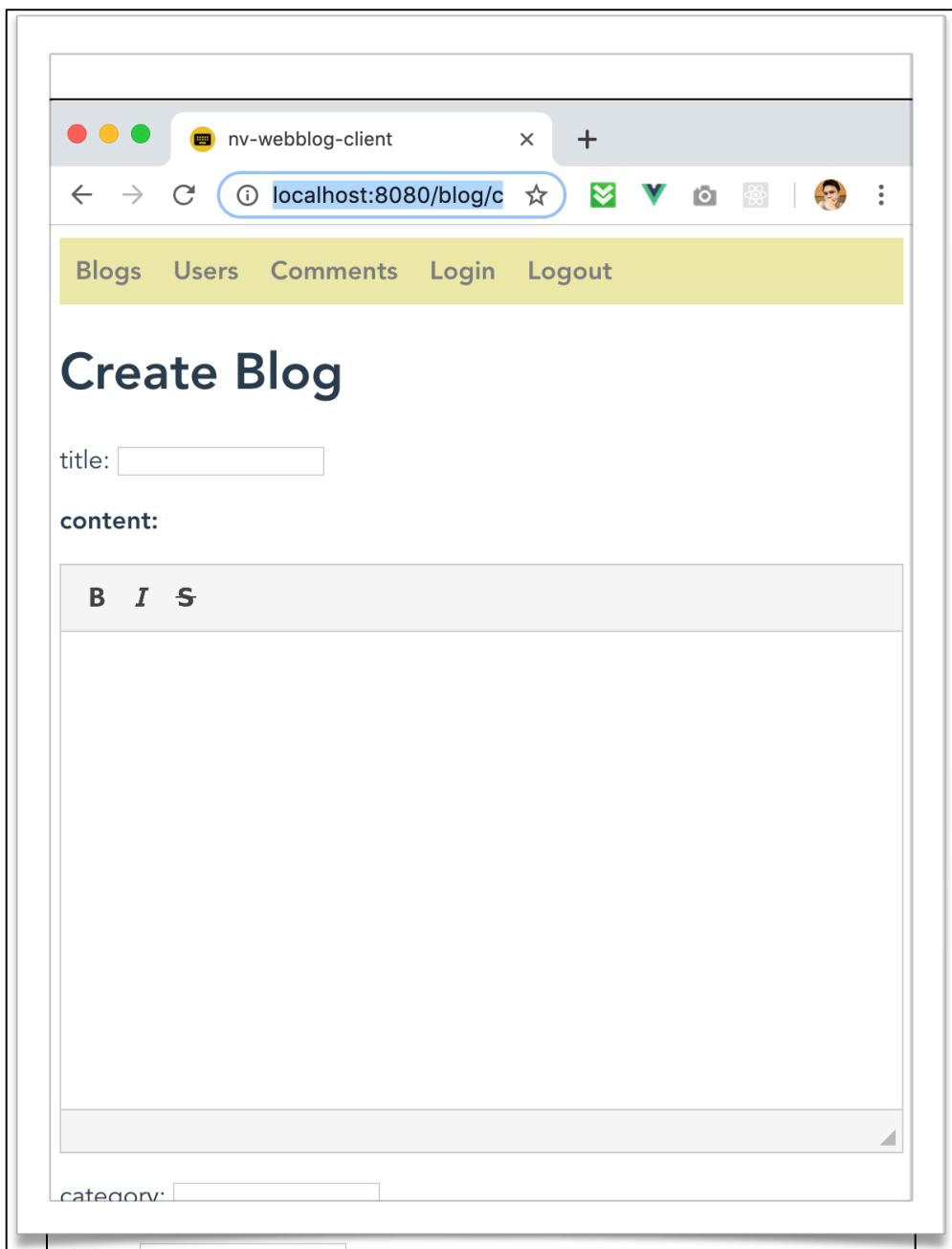
จากนั้นทำการเพิ่ม Component ของ VueCkeditor เข้าไป โดยทำการเพิ่มต่อจาก methods: {} ของเราดังนี้

```
components: {
    VueCkeditor
},
```

จากนั้นเปิด Web Browser ของเราขึ้นมา แล้วทำการเปิดที่

<http://localhost:8080/blog/create>

จะเห็นว่า หน้าตาของ Editor ของเราเปลี่ยนไปเป็น CKEditor แล้ว



ลองทดสอบ create blog ดูนะครับ ว่าเรารสามารถเพิ่ม Blog ได้ปกติหรือไม่ ผลกระทบของการทดสอบของผม สามารถทำการ create blog ได้ปกติครับ แต่จะเห็นว่า ckeditor ของผมทำการสร้าง HTML Style ส่งเข้าไปให้เลยนั่นเอง

เมื่อเราทำการใช้งาน ckeditor ในการเขียน blog และ เราจะใช้งาน ckeditor ในส่วนของการ edit blog ด้วยเช่นกัน

ทำการเปิด src/components/Blogs/EditBlog.vue ของเรามา และทำการแก้ไข code เมื่อ src/components/Blogs/CreateBlog.vue เลยก็รับ ผลจะไม่ทำให้ดูนะครับ แต่ถ้าติด Error ตรงไหน สอบตามได้เช่นเคยครับ

ตอนนี้เราทำการแก้ไข blog พร้อมใส่ HTML Style กันได้แล้วนะครับ เวลาเราเขียน Blog ก็จะออกมาสวยแล้วครับ แต่พมัยังใส่ Toolbar ให้ไม่เยอะมากนะครับ ยังไม่ได้เขียนตอบสนอง OnFocus และ OnBlur ของ CKEditor ด้วยเช่น แต่เราจะเบรค เรื่อง CKEditor กันไว้ตรงนี้ก่อนครับ เดียวเราจะอุปมาใส่ code เพิ่มกันภายหลัง เพราะเราจะทำส่วนต่อไป เราหาจุดเชื่อมในการเพิ่ม code ลำบากครับ

เพิ่ม Toolbar ให้กับ CKEditor

ในหัวข้อที่ผ่านมาจะเห็นว่า Toolbar ของเราไม่สามารถใช้งานได้จริง ดังนั้นเราจะทำการเพิ่ม Toolbar ให้มัน โดยพมทำ การเพิ่มให้มันใน cycle ของการ created () ดังนี้ *→ ก่อน async created() น้องค้ากันป่าน DB*

```
created () {
  this.config.toolbar = [
    {
      name: "document",
      items: [
        "Source",
        "-",
        "Save",
        "NewPage",
        "Preview",
        "Print",
        "-",
        "Templates"
      ]
    },
    {
      name: "clipboard",
      items: [
        "Cut",
        "Copy",
        "Paste"
      ]
    }
  ]
}
```

```
        "Cut",
        "Copy",
        "Paste",
        "PasteText",
        "PasteFromWord",
        "-",
        "Undo",
        "Redo"
    ],
},
{
    name: "editing",
    items: ["Find", "Replace", "-", "SelectAll", "-", "Scayt"]
},
{
    name: "forms",
    items: [
        "Form",
        "Checkbox",
        "Radio",
        "TextField",
        "Textarea",
        "Select",
        "Button",
        "ImageButton",
        "HiddenField"
    ]
},
"/",
{
    name: "basicstyles",
    items: [
        "Bold",
        "Italic",
        "Underline",
        "Strike",
        "Subscript",
        "Superscript",
        "-",
        "CopyFormatting",
        "RemoveFormat"
    ]
},
{
```

```

        name: "paragraph",
        items: [
            "NumberedList",
            "BulletedList",
            "-",
            "Outdent",
            "Indent",
            "-",
            "Blockquote",
            "CreateDiv",
            "-",
            "JustifyLeft",
            "JustifyCenter",
            "JustifyRight",
            "JustifyBlock",
            "-",
            "BidiLtr",
            "BidiRtl",
            "Language"
        ]
    },
    { name: "links", items: ["Link", "Unlink", "Anchor"] },
    {
        name: "insert",
        items: [
            "Image",
            "Flash",
            "Table",
            "HorizontalRule",
            "Smiley",
            "SpecialChar",
            "PageBreak",
            "Iframe",
            "InsertPre"
        ]
    },
    "/",
    { name: "styles", items: ["Styles", "Format", "Font", "FontSize"] }
},
{ name: "colors", items: ["TextColor", "BGColor"] },
{ name: "tools", items: ["Maximize", "ShowBlocks"] },
{ name: "about", items: ["About"] }
]
}

```

นอก~~X~~จากการเพิ่ม config แล้ว มี Toolbar หลายตัวที่ทำงานในลักษณะ Popup Dialog เราเลยจำเป็นต้องทำ Function ให้มันเรียกใช้ใน methods ของเราดังนี้

```
onBlur (editor) {  
    console.log(editor);  
,  
onFocus (editor) {  
    console.log(editor);  
,
```

ตอนนี้ CKEditor ของเราร้อมใช้งานจริงๆ แล้ว อย่าลืมไปเพิ่ม code ที่จำเป็นใน Edit Blog Component ด้วยนะครับ

บทที่ 13 ทำการ Upload File รูปภาพเพื่อใช้งานใน Blog กัน

ตอนนี้ระบบ Web Blog ของเราเริ่มเป็นรูปเป็นร่างแล้วนะครับ ตอนนี้เราเริ่มมีระบบหลังบ้าน หรือ Back Office สามารถทำการ Login เข้ามาเพื่อเขียน Blog ได้แล้ว แต่เนื่องจาก เรายังไม่ได้ทำการปิดประตูหลังบ้านของเราทั้งหมด บางคนอาจยังไม่ค่อยเห็นภาพเท่าไหร่ แต่ไม่กังวล ตามผมไปก่อนนะครับ ผมจะแทรกไปเรื่อย ๆ ครับ

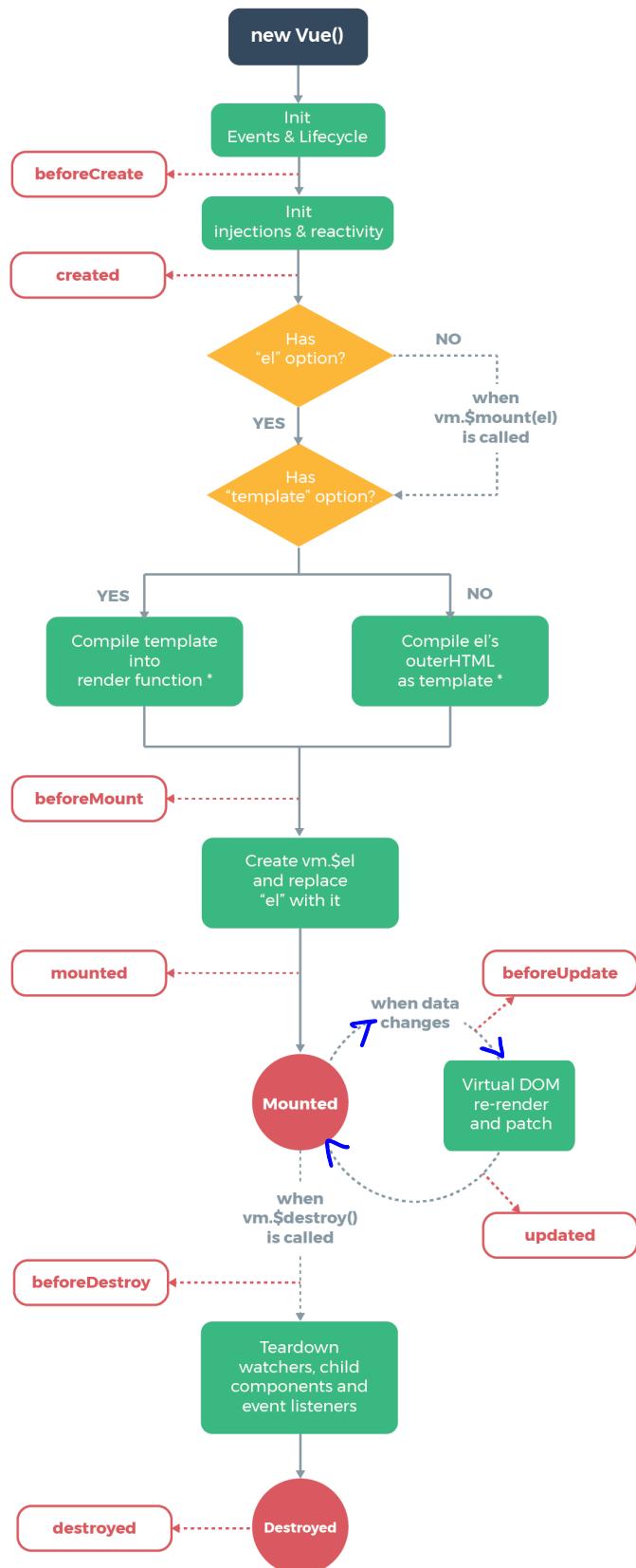
Vuejs Lifecycle

จากที่ผ่าน ๆ มา เราได้ใช้งาน vuejs เป็นต้นกันได้แล้ว แต่จริง ๆ แล้วเรายัง ⁹⁸ ความสามารถของ vuejs ไปส่วนหนึ่งเท่านั้นเอง เพราะจริง ๆ แล้วการที่เราใช้ javascript framework มาทำ front end นั้น เราต้องการให้มันตอบสนองการใช้งาน ได้แบบทันทีทันใด ในหน้านั้นเลย หรือ Update การแสดงผลทันที เมื่อเราดำเนินการอย่างหนึ่งอย่างใดกับระบบของเรา ในบทนี้จะเป็นตัวอย่างที่ดีในการใช้งานในลักษณะที่กล่าวมา

ไม่ต้องกังวลครับ การตอบสนองที่ว่ามันนั้น เป็นเพียงแค่การจัดการ JSON หรือ data ของเราเท่านั้นเอง เพียงเราจัดการเพิ่ม แก้ไข หรือ ลบ JSON ของเรา vuejs ของเราจะทำการ Update ให้เท่านั้นเองครับ

ก่อนจะไปกันต่อ เราจะมาพูดเรื่อง Lifecyle ของ Vuejs กันครับ ก่อนหน้านี้ เรารู้จักกับ created () และรู้แล้วว่ามันจะทำงานก่อน อะไรที่เราจะเริ่มต้น เช่น ดึงข้อมูลจาก api ของเรา เราจะมาจัดให้มันไว้ตรงนี้ created นี่แหละ เป็นหนึ่งใน Lifecycle ของเรา

เรามาดูรูปประกอบกันครับ



* template compilation is performed ahead-of-time if using a build step, e.g. single-file components

จากรูปจะอธิบายแบบง่าย ๆ บ้าน ๆ ตามสีต่อไปนี้ครับ

Lifecycle ของ vuejs ที่เราจะใช้กัน คือ

1. created () เกิดขึ้นก่อน จะดึง HTML มารวมกับ Vue Component ของเรา ถ้า จะดึง Data โดยไม่ยุ่ง DOM หรือ HTML Element ที่เราอ้างถึง ผู้มักจะใช้งาน
2. updated () เกิดขึ้นตอนที่ เกิดการเชื่อมโยง HTML ของเรา มั่งหมดแล้ว ดังนั้น ถ้าต้องการอ้าง class หรือ id ของ tag HTML ของเรา ต้องใช้ตรงนี้ครับ
3. mounted () เป็น cycle ยอดนิยม ประกอบด้วย beforeUpdated และ updated () ดังนั้น mounted () ไม่ garant ดีว่า เราจะสามารถอ้างถึง DOM หรือ HTML ของ เสมอไป แต่โดยส่วนใหญ่ได้ คิดอะไรไม่ออก ให้ mounted ไว้ก่อนครับ

นอกจาก lifecycle ที่กล่าวมาแล้ว ผู้ยังมีอีกตัวที่จะนำเสนอ คือ computed โดย computed นั้น คือ properties นะครับ ไม่ใช่ lifecycle แต่ผู้ ก็จำร่วม ๆ กันไป ชื่อนั้นสำคัญ ใจน รู้ว่าใช้งานยังไง เขียนยังไงนี่สิสำคัญครับ

โดย computed จะเป็น watcher อ่า ง เข้าไปอีก เอาเป็นว่าเราเรียกใช้ computed ในการเฝ้าดู data ของเรา ถ้า data มีการเปลี่ยนแปลง มันจะ updated ผ่าน function ภายใน computed นั้นเองครับ data เปลี่ยนปุ๊บ computed update ปั๊บ เรามักเอาไปใช้กับ search function นั้นเองครับ จบครับไม่พูดยะ ไปเขียน กันเลยครับ

เขียนโปรแกรมเพื่อ upload file จริง ๆ ล่ะ

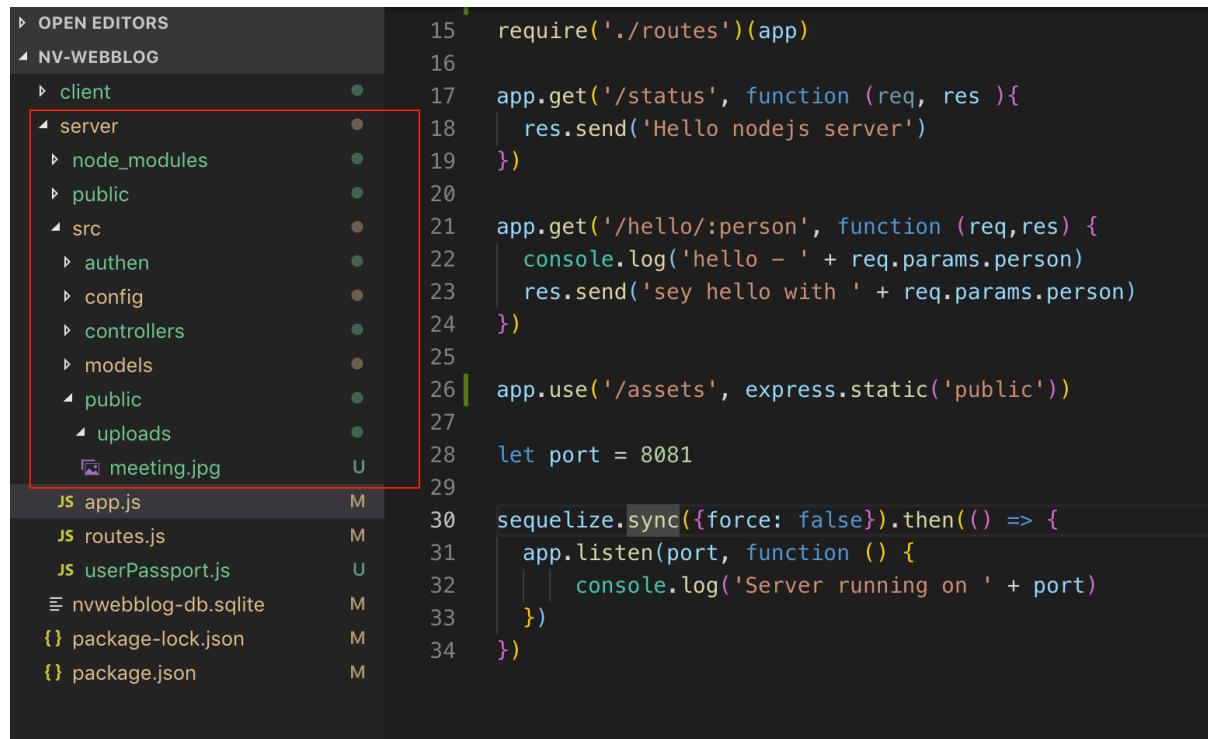
มาถึงตอนนี้เราจะมาทำการ Upload ไฟล์ รูปภาพเพื่อใช้ประกอบใน Blog และ Thumbnail ของเรา กันครับ โดยผู้จะทำการ Upload ไฟล์ไปเก็บไว้ใน Back End หรือ Server ของเรา ดังนั้นผู้ต้องเตรียมโฟล์เดอร์ ที่จะให้ client หรือ vuejs ของเราทำการ upload ไฟล์มาเก็บไว้กันก่อน

ผู้ทำการสร้างโฟล์เดอร์ public ขึ้นมา และสร้างโฟล์เดอร์ uploads ไว้ภายในดังภาพประกอบ

```
▶ OPEN EDITORS
◀ NV-WEBBLOG
  ▶ client
  ▶ server
  ▶ node_modules
  ▶ public
    ▲ src
      ▶ authen
      ▶ config
      ▶ controllers
      ▶ models
      ▲ public
        ▶ uploads
JS app.js M
JS routes.js M
JS userPassport.js U
≡ nvwebblog-db.sqlite M
{} package-lock.json M
{} package.json M
Show All Comm Go t
```

โดยก่อนหน้านี้ เราได้ทำการกำหนดให้ public ของเรามีการ access ไฟล์จากภายนอก server ได้แล้ว เรามาทำการทดสอบกันดูครับว่าเราสามารถ access หรือเข้าถึงไฟล์ในโฟล์เดอร์ public ผ่าน /assets ของเราได้จริงมั้ย โดยเราทำการ

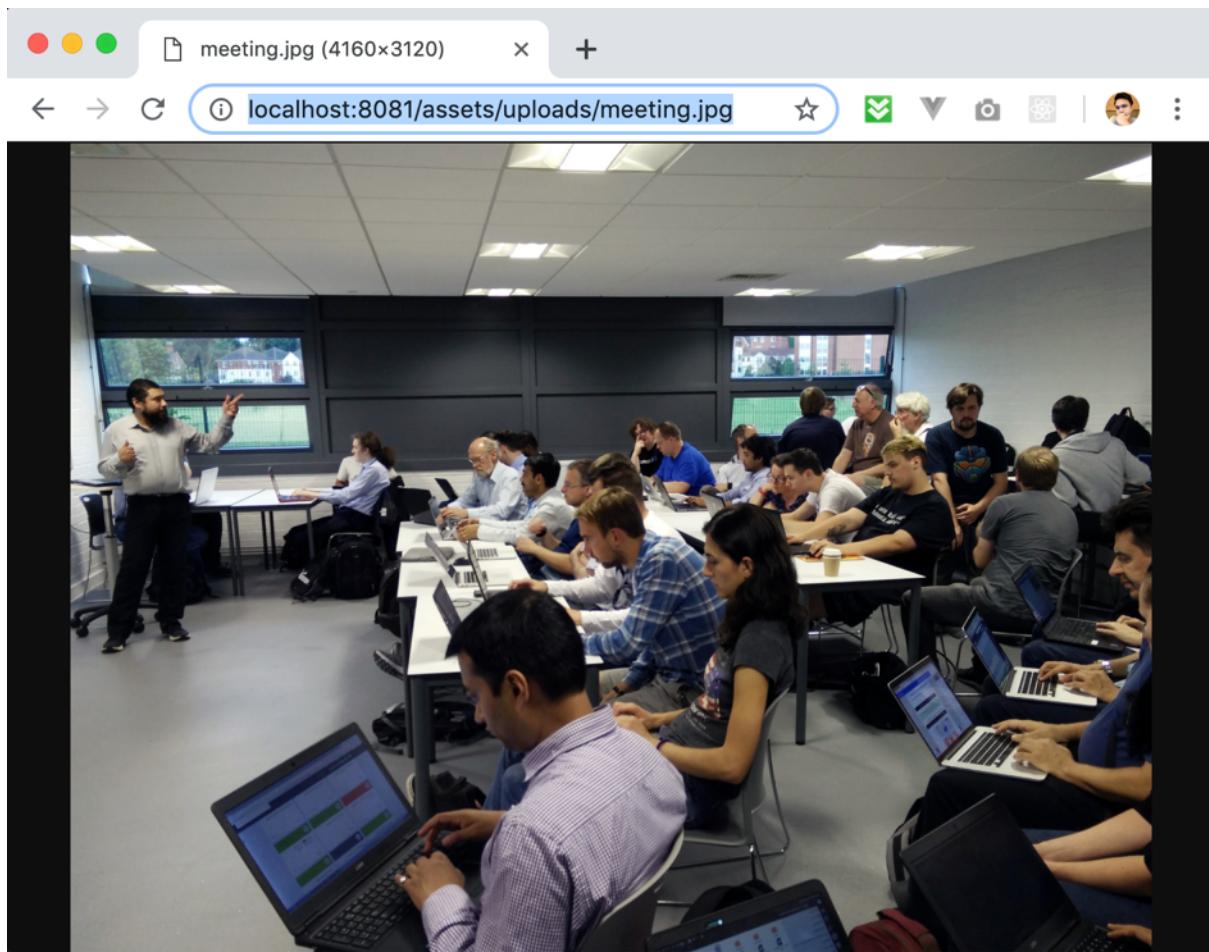
copy ไฟล์รูปภาพอะไรก็ได้ ไปไว้ในโฟล์เดอร์ public ของเรา จากนั้นเราทำการ



The screenshot shows a code editor interface with a sidebar on the left displaying the file structure of a project named 'NV-WEBBLOG'. The 'server' directory is selected and highlighted with a red border. Inside 'server', there are sub-directories 'client', 'node_modules', 'public', and 'src', along with files 'authen.js', 'config.js', 'controllers.js', 'models.js', 'public.js', 'uploads.js', and 'meeting.jpg'. Below the sidebar, the main editor area shows a portion of the 'app.js' file with line numbers 15 to 34. The code includes routes for '/status', '/hello/:person', and '/assets', and sets up a static server for the 'public' directory at port 8081.

```
15 require('./routes')(app)
16
17 app.get('/status', function (req, res ){
18   res.send('Hello nodejs server')
19 })
20
21 app.get('/hello/:person', function (req,res) {
22   console.log('hello - ' + req.params.person)
23   res.send('sey hello with ' + req.params.person)
24 })
25
26 app.use('/assets', express.static('public'))
27
28 let port = 8081
29
30 sequelize.sync({force: false}).then(() => {
31   app.listen(port, function () {
32     console.log('Server running on ' + port)
33   })
34 })
```

เรียกไปที่ url path รูปภาพของเรา แต่แทนที่ public ด้วย assets ครับ



แสดงว่าตอนนี้ เรามีโฟล์เดอร์ที่สามารถใช้งานแบบ public เพื่อทำการ upload ไฟล์ไปเก็บ แต่ไม่ต้องกลัวว่าใครจะมา upload ขึ้นมามั่วซึ่งจะครับ เดียวเรารายความ Authen ดักไว้ได้ครับ

เริ่มทำการเขียนโปรแกรม รับการ Upload File บน Back End

ในการเขียนโปรแกรม Upload File ขึ้น Server ของเราจะใช้งาน package ที่ชื่อว่า multer กันครับ ทำการติดตั้ง multer กันก่อนเลยครับ

```
npm install --save multer
```

จากนั้นเรามาเขียนโปรแกรม เพื่อรับการ upload ไฟล์จาก client มาที่ server ของเรากันครับ

ทำการเปิด src/routes.js ขึ้นมา และทำการเพิ่ม code ชุดนี้เข้าไปครับ เพื่อทำการ require multer ที่เราติดตั้งมาไว้

```
let multer = require("multer")
```

เอาไว้ก่อน

```
module.exports = (app) => {
```

นำครับ จากนั้น ต่อจาก code ด้านบน เราใส่ code ตามนี้ครับ

```
// upload section
let storage = multer.diskStorage({
  destination: function(req, file, callback) {
    callback(null, "./public/uploads");
  },
  filename: function(req, file, callback) {
    // callback(null, file.fieldname + '-' + Date.now());
    console.log(file);
    callback(null, file.originalname);
  }
})
let upload = multer({ storage: storage }).array("userPhoto", 10)
```

Code ชุดนี้ ทำงานผ่าน function diskStorage ใน multer ทำการอ่านไฟล์ ที่ส่งมาจาก Form ในส่วนของ Front End หรือ client ของเรา และทำการ copy ไฟล์ไปไว้ที่ Server ของเราเอง

userPhoto คือ ~~file~~ attribute file ของ HTML Form ที่เราทำการ Post Data มาหา Server จาก client นั่นเอง

เมื่อเรามี function ในการรับการ upload และเราต้องมาทำ api หรือ route path ให้ client ของเราเรียกใช้งานกันครับ ทำการเพิ่ม code ในส่วนท้าย ต่อจาก route path อีกดังนี้ครับ

```
// upload
app.post("/upload", function(req, res) {
  // isUserAuthenticated,
  upload(req, res, function(err) {
    if (err) {
      return res.end("Error uploading file.");
    }
    res.end("File is uploaded");
  })
})
```

นั่นหมายความว่า เมื่อมีการ form post ขึ้นมา และผ่านการ Authen ของเราจะไปเรียก function upload ของเราเพื่อทำการ upload ไฟล์ของเรานั้นเอง ถ้า upload สำเร็จจะตอบกลับตาม message ที่กำหนดไว้นั้นเอง โดยผมจะปิดการ Authen ของเราไว้ก่อน และจะมาทำการ ~~ใส่~~ ^{รีบ} Authen ทั้งหมดภายหลังครับ

ส่วนการลบไฟล์รูปภาพบน server

ในส่วนนี้เราจะทำ api สำหรับทำการลบไฟล์รูปภาพที่เราไม่ต้องแล้ว ^{บน} ~~บน~~ server ของเรา กัน เนื่องจากเรามี function ในการจัดการเรื่องของไฟล์ไม่เยอะมากนัก เราจึงจะใส่ function ในการลบไฟล์ของเราไว้ที่ src/routes.js เมื่อ我们要在 API 中实现文件删除功能，我们可以在 routes.js 中添加一个名为 delete 的路由处理函数。我们将使用 fs 模块来读取文件并删除它。

```
//delete file uploaded function
app.post('/upload/delete', async function (req, res) {
  try {
    const fs = require('fs');

    console.log(req.body.filename)
```

```
    fs.unlink(process.cwd() + '/public/uploads/' + req.body.filename,
  (err) => {
    if (err) throw err;
    res.send("Delete sucessful")
    // console.log('successfully deleted material file');
  });
} catch (err) {
  res.status(500).send({
    error: 'An error has occured trying to delete file the material'
  })
}
})X
```

โดย code ชุดนี้จะไปทำการลบไฟล์ตามชื่อไฟล์ที่ทาง client post data มาแน่นองซึ่งเราจะไปทำการทดสอบกันตอนที่เราส่วนหน้าบ้าน หรือ Front End ในส่วนของ Back Office เพื่อทำการลบไฟล์ครับ

ทำส่วน Front End เพื่อทำการ Upload File

จากที่ผ่านมา เราได้เตรียมส่วนการ Upload File บน Back End ของเราไว้เรียบร้อยแล้ว ผสมในส่วนนี้ผมจะทำการสร้างส่วนหน้าบ้านหรือ Front End ในส่วน Back Office ของผมเพื่อทำการ Upload File ไปให้ server หรือ Back End ของเราแน่นอง

Workflow ของผมมีดังนี้ครับ

1. สร้าง Service สำหรับ upload
2. สร้าง Vue Component เพื่อทำการ upload
3. ทำการสร้าง Route เพื่อเชื่อมส่วนต่าง ๆ เพื่อทำการทดสอบ

อันที่จริงแล้ว เรายังต้องการจะ upload file รูปภาพ เพื่อทำการใช้งานใน Blog ของเรา แต่ผมกลัวว่าถ้าเราเขียน code ต่อเลยจะสับสนได้ง่าย และอาจจะนำไปประยุกต์ใช้งานได้ยาก ผมเลยทำการแยกส่วน Upload File ออกมา ทำการเขียนเพื่อทดสอบการทำงานเสียก่อนว่าส่วนนี้ work แล้ว จึงนำไปใส่ในส่วน Create Blog และ Update Blog ต่อไป

สร้าง Upload Service

เราทำการสร้างไฟล์ src/services/UploadService.js ขึ้นมา แล้วทำการเขียน code เพื่อ Post Data ไปที่ api server ของเรา ดังนี้

```
import Api from '@/services/Api'

export default {
  upload(formData) {
    return Api().post('upload', formData)
  },
  delete(picture) {
    return Api().post('/upload/delete', picture)
  }
}
```

โดย service ของผมจะไปเรียกใช้ api บน server หรือ Back End ของเราที่สร้างไว้ก่อนหน้านี้ เพื่อทำการ upload และ delete ไฟล์ที่เราอัปโหลดแล้วนั่นเอง

สร้าง Vue Component สำหรับทำการ Upload File

ผมทำการสร้างโฟล์เดอร์

src/components/Utils

แล้ว

สร้างไฟล์

```
config
node_modules
src
  assets
  components
    Blogs
    Comments
    Users
    Utils
      Upload.vue
      Header.vue
      Login.vue
    router
```

src/components/Utils/Upload.vue ขึ้นมาเพื่อเป็น component สำหรับทดสอบ

การเขียน code ที่ src/components/Utils/Upload.vue ของเรา ผมเริ่มจากการทำ <template></template> หรือหน้าตา HTML ของก่อนครับ โดยผมเขียน code ดังนี้

```
<template>
<div>
  <!-- <back-header /> -->
  <h1>Upload Material:</h1>
  <form enctype="multipart/form-data" novalidate>
    <div class="dropbox">
      <input type="file" multiple :name="uploadFieldName"
        :disabled="isSaving" @change="filesChange($event.target.name,
        $event.target.files); fileCount = $event.target.files.length"
        accept="image/*" class="input-file">
```

```

<!-- <p v-if="isInitial || isSuccess"> -->
<p v-if="isInitial">
    Drag your file(s) here to begin<br> or click to browse
</p>
<p v-if="isSaving">
    Uploading {{ fileCount }} files...
</p>
<p v-if="isSuccess">
    Upload Successful.
</p>
</div>
</form>
</div>
</template>

```

โดย code ส่วนนี้ของเราก็จะเป็นการทำ Form HTML เพื่อทำการ Upload File นั่นเอง~~เป็น~~การ enctype="multipart/form-data" เพื่อทำการ อัปโหลดได้ครั้งละ หลาย ๆ ไฟล์ นั่นเองครับ *เป็นการป้องกัน*

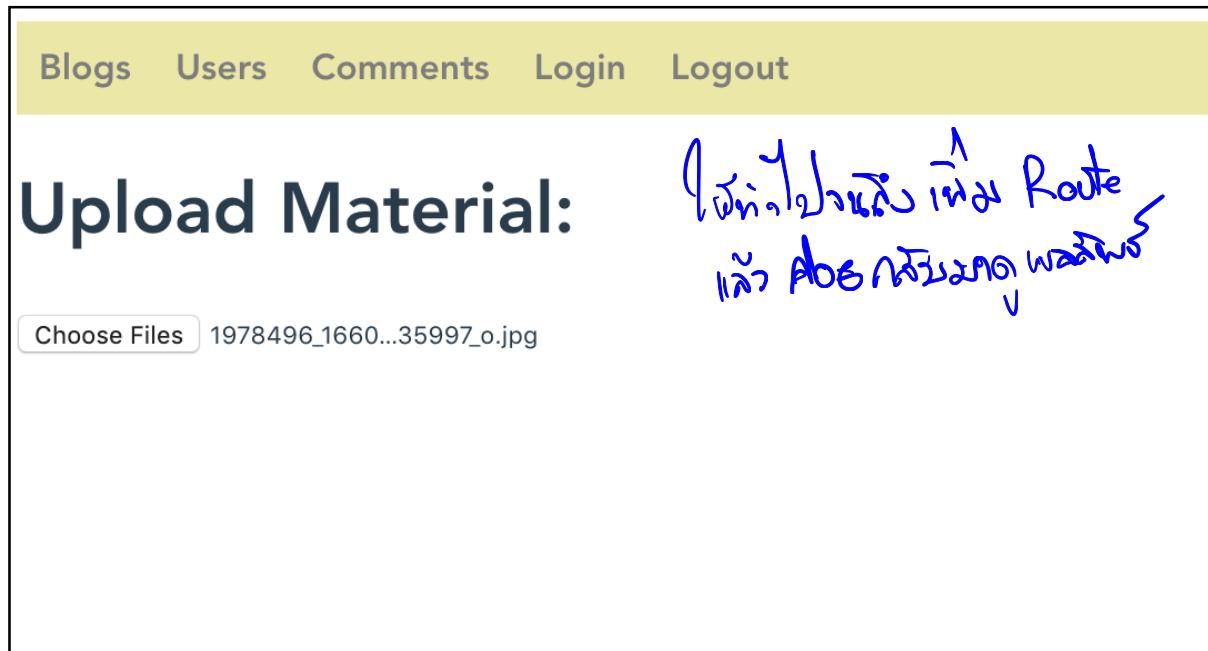
จะมีส่วนที่นำสันใจอีกส่วน คือ

```

<p v-if="isInitial">
    Drag your file(s) here to begin<br> or click to browse
</p>
<p v-if="isSaving">
    Uploading {{ fileCount }} files...
</p>
<p v-if="isSuccess">
    Upload Successful.
</p>

```

ซึ่งเป็นการแสดงสถานะการ upload ของเรา โดยใช้ v-if เป็นตัวกำหนดเงื่อนไขในการแสดงผลจาก function ตามที่เรากำหนดไว้ แต่เราตอนนี้เรายังไม่ได้เขียน แค่



ใส่ชื่อ function loyalty ไว้ก่อน ตอนนี้หน้าตา ส่วน upload ของเราจะเป็นดังนี้

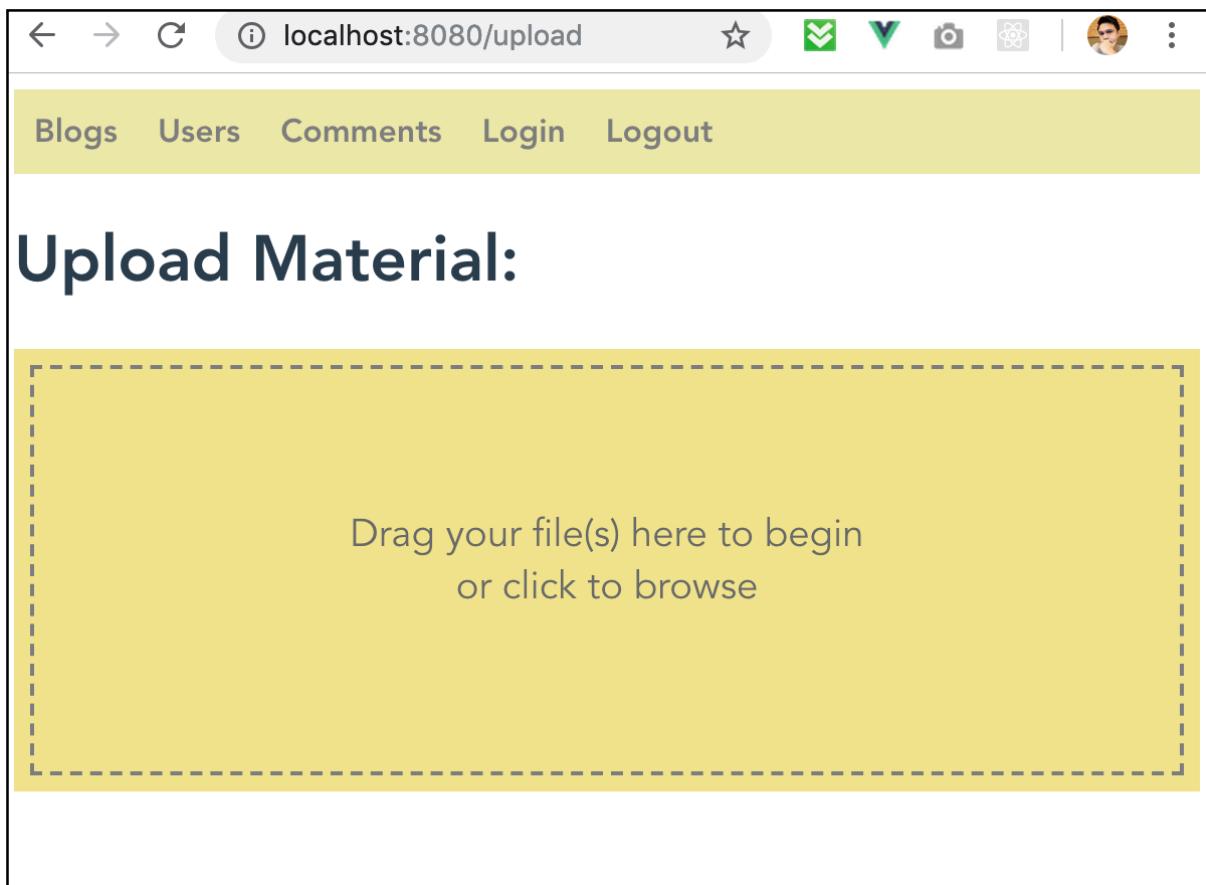
ทำการกำหนด Style ให้ Form Upload

เมื่อเราทำการเขียน HTML Form สำหรับการ Upload File ของเราแล้ว เราจะมาทำการเพิ่มสี และหน้าตาของมันให้น่าใช้งานขึ้นมาอีกหน่อย โดยการ เขียน code ในส่วนของ <style scope></style> ของเราดังนี้ครับ

```
<style scoped>
  .dropbox {
    outline: 2px dashed grey; /* the dash box */
    outline-offset: -10px;
    background: lemonchiffon;
    color: dimgray;
    padding: 10px 10px;
    min-height: 200px; /* minimum height */
    position: relative;
    cursor: pointer;
  }
```

```
.input-file {  
    opacity: 0; /* invisible but it's there! */  
    width: 100%;  
    height: 200px;  
    position: absolute;  
    cursor: pointer;  
}  
  
.dropbox:hover {  
    background:khaki; /* when mouse over to the drop zone, change color  
*/  
}  
  
.dropbox p {  
    font-size: 1.2em;  
    text-align: center;  
    padding: 50px 0;  
}  
</style>
```

โดย class ของเราจะตรงกับ html template ที่เราเขียนไว้ก่อนหน้านี้นั่นเอง เมื่อเราใส่ style ให้ form upload ของเราเรียบร้อยแล้ว จะมีหน้าตาแบบนี้ครับ



ทำ Script สำหรับการ Upload

หลังจากเราทำการสร้าง Form Upload รวมถึงทำหน้าตาให้น่าใช้ด้วย style กับการแสดงผลสถานะ การ upload ต่าง ๆ ไว้เรียบร้อยแล้ว เราจะมาเขียน script เพื่อทำการ upload กันครับ

โดยผมเริ่มจาก Import Upload Service ของเราเข้ามក่อน

```
import UploadService from '@services/UploadService'
```

จากนั้นผมจะทำตัวแปรเอาไว้บอกสถานะ โดยมีรายละเอียดดังนี้

```
const STATUS_INITIAL = 0,  
      STATUS_SAVING = 1,  
      STATUS_SUCCESS = 2,  
      STATUS_FAILED = 3;
```

ผมจะทำการสร้าง data () เพื่อใช้ในการเขียนโปรแกรม Upload File ดังนี้ครับ

```
data () {
  return {
    BASE_URL: "http://localhost:8081/assets/uploads/",
    error: null,
    // uploadedFiles: [],
    uploadError: null,
    currentStatus: null,
    uploadFieldName: "userPhoto",
    uploadedFileNames: []
  }
},
```

BASE_URL คือ ตำแหน่งที่จะ upload file ไปไว้

Error: เอาไว้บอกสถานะ error ที่ server ตอบกลับมา

uploadedFileNames [] คือ ไฟล์ที่เราจะทำการ upload

CurrentStatus เอาไว้เช็คสถานะที่เกิดขึ้น ระหว่างการ upload

uploadFileName: คือ ชื่อฟิลด์ของ tag file ใน HTML Form ของเรา

เมื่อเรามี data () แล้ว เราจะมาทำการเพิ่ม methods สำหรับการ upload กันครับ โดย ผมมี methods ที่เกี่ยวข้องทั้งหมดดังนี้ครับ

```
methods: {
  navigateTo (route) {
    console.log(route)
    this.$router.push(route)
  },
  wait(ms) {
    return x => {
      return new Promise(resolve => setTimeout(() => resolve(x), ms));
    };
  },
  reset() {
```

```

// reset form to initial state
this.currentStatus = STATUS_INITIAL
// this.uploadedFiles = []
this.uploadError = null
this.uploadedFileNames = []
},
async save(formData) {
// upload data to the server
try {
    this.currentStatus = STATUS_SAVING
    await UploadService.upload(formData)
    this.currentStatus = STATUS_SUCCESS
    this.clearUploadResult()
} catch (error) {
    console.log(error)
    this.currentStatus = STATUS_FAILED
}
},
filesChange(fieldName, fileList) {
// handle file changes
const formData = new FormData();

if (!fileList.length) return;

// append the files to FormData
Array.from(Array(fileList.length).keys()).map(x => {
    formData.append(fieldName, fileList[x], fileList[x].name);
    this.uploadedFileNames.push(fileList[x].name);
});

// save it
this.save(formData);
},
clearUploadResult: function(){
    setTimeout(() => this.reset(), 5000);
}
}

```

โดยที่แต่ละ methods ของ pm จะทำงานดังนี้ครับ การทำงานของ pm เริ่มที่

```
filesChange(fieldName, fileList) {
```

โดยทำงานเมื่อ file attribute ใน HTML Form ของเราเกิดการเปลี่ยนแปลงไม่ว่าจะทำการ Browse File หรือ Drag and Drop มา

โปรแกรมที่ pm เขียนขึ้นจะทำการสร้าง FormData จากนั้น ทำการอ่านค่าใน file attribute มาใส่ใน FormData ของเรา และส่ง FormData ของเราไปทำงานที่

```
async save(formData) {
```

Function นี้จะทำการ Post Form Data ของเราไปที่ api บน server ของเราในขณะที่ทำการ Post Data ไปนั้นเราจะทำการเปลี่ยนแปลงค่าใน currentStatus ด้วย ว่าตอนนี้เราทำงานอยู่ส่วนไหน รวมถึงบอกสถานะหากทำงานไม่สำเร็จด้วย

จัดการแสดงสถานะการ upload ต่าง ๆ ด้วย Computed

ก่อนหน้านี้ pm เลยพูดถึงเรื่อง computed () ซึ่งเป็น property หนึ่งของ vuejs เอาไว้ใช้ในการตรวจสอบการเปลี่ยนแปลงค่าของ data ที่เราสนใจ

การตรวจสอบสถานะ ของการ upload นี้เป็นตัวอย่างที่ดีมาก ในการทำความเข้าใจส่วนนี้

pm ทำการเขียนส่วนของ computed ของ pm ดังนี้ โดยต่อจาก methods

```
computed: {
  isInitial() {
    return this.currentStatus === STATUS_INITIAL;
  },
  isSaving() {
    return this.currentStatus === STATUS_SAVING;
  },
  isSuccess() {
```

```
        return this.currentStatus === STATUS_SUCCESS;
    },
    isFailed() {
        return this.currentStatus === STATUS_FAILED;
    }
},
```

เราจะเห็นว่ามี function เพื่ออ่านค่าสถานะต่าง ๆ ออกมา โดยผ่านเรียกใช้ผ่าน

```
<p v-if="isInitial">
  Drag your file(s) here to begin<br> or click to browse
</p>
<p v-if="isSaving">
  Uploading {{ fileCount }} files...
</p>
<p v-if="isSuccess">
  Upload Successful.
</p>
```

Code ด้านบน ของผ่านใน template โดย computed จะทำงาน function ต่าง ๆ เมื่อมีการเปลี่ยนแปลงค่าใน data ที่เกี่ยวข้อง และ return ค่าออกมาเพื่อไปแสดง ใน template ของผ่านนั่นเอง

Computed นี้มีประโยชน์มาก เอาไว้จัดการอัพเดตค่าต่าง ๆ ใน Front End ของเราได้อย่างมีประสิทธิภาพ

กลับมาที่ function save() ของผ่าน เมื่อทำการ upload file ได้เสร็จสมบูรณ์แล้ว โปรแกรมของเราจะไปทำงานที่

```
clearUploadResult: function(){
```

โดยใน function นี้ไม่มีอะไรมาก ผ่านจะ delay เพื่อหลังจากแสดงผลว่า upload เสร็จสมบูรณ์แล้ว ให้เริ่มต้นกระบวนการใหม่โดยการ reset ค่าต่าง ๆ ทั้งหมดใน ~~function~~ reset () นั่นเอง

function

แต่เนื่องจากผู้ต้องการให้ reset () ถูกเรียก ตอนเว็บไซต์ของเรา เริ่มทำงานใน url ที่เรากำหนดไว้ ผู้จังทำการสร้าง lifecycle ที่ชื่อว่า created () ขึ้นมาดังนี้

```
created () {  
    this.reset();  
}
```

โดยผู้ใช้ต่อไว้จาก computed () ของผู้ ซึ่งจะใส่ data (), methods:, computed (), หรือ created () นั้นจะเรียงลำดับยังไงไม่มีปัญหานะครับ เพียงแต่ คั่น ',' ไว้ให้ถูกต้องตาม syntax เพียงพอแล้วครับ แต่ส่วนใหญ่ผู้จะเอา data () และ ตามด้วย methods: ขึ้นก่อน ที่เหลือก็แล้วแต่จังหวะครับ

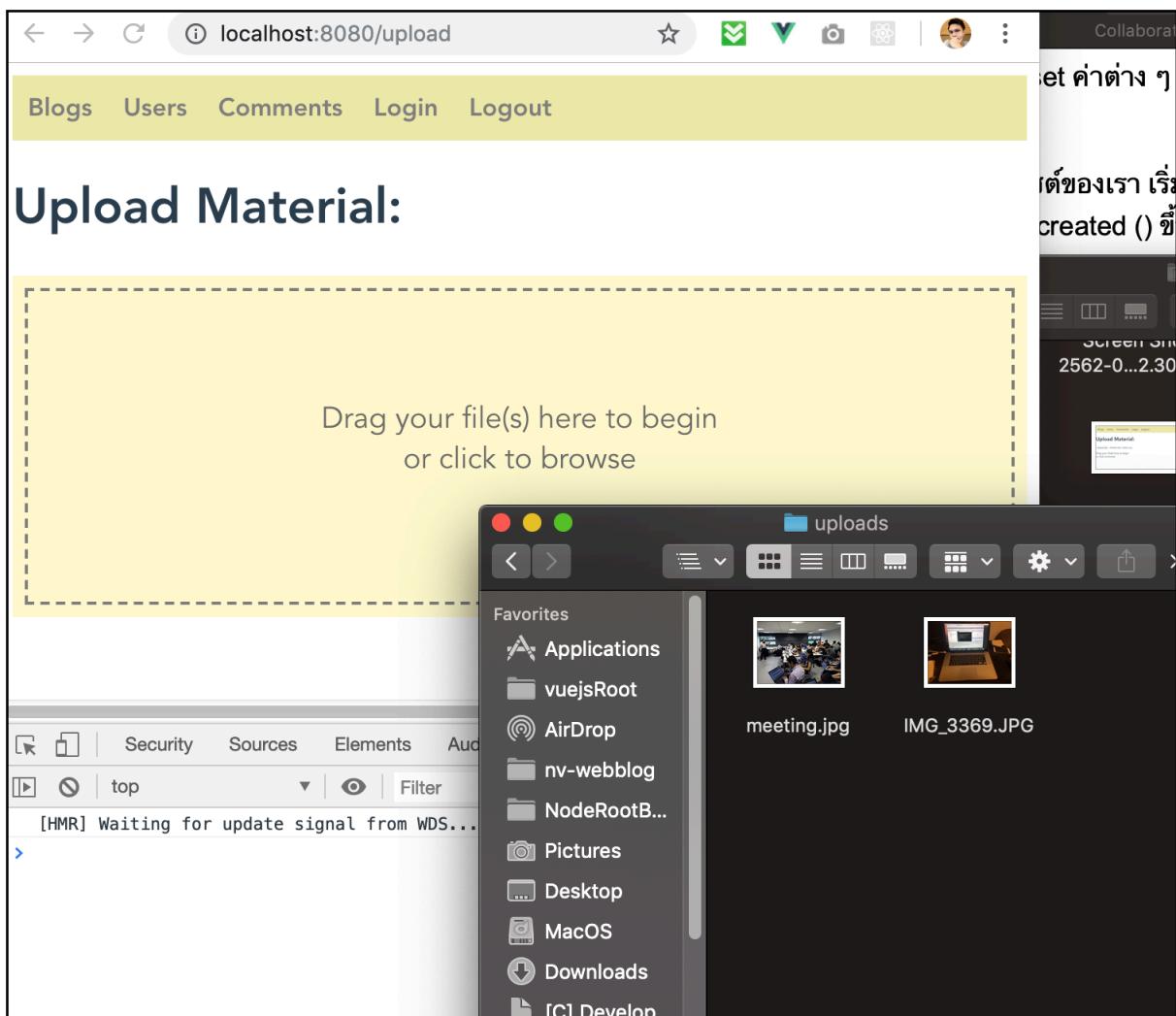
ทำการ mapping route ไปที่ upload component ของเราโดยเพิ่ม code ชุดนี้เข้าไปที่ router/index.js ครับ

```
// Upload Testing  
import Upload from '@/components/Utils/Upload'
```

```
// upload testing  
{  
    path: '/upload',  
    name: 'upload',  
    component: Upload  
},
```

ตอนนี้ผู้ส่วน Upload ของเราจะ work แล้ว ผู้ทำการทดสอบ Upload File ทั้งเลือกจากการ Browse และ Drag and Drop แล้วทำการเปิดโฟล์เดอร์ที่เก็บไฟล์ ของเรา คือ src/public/uploads เพื่อทำการตรวจสอบ

ผลคือ upload ได้อย่างดีครับ



ทำการ ใส่ code upload ใน Create Blog

เราได้ทำการเขียนส่วนของการ upload file ของเราเสร็จเรียบร้อยแล้วในหัวข้อ ก่อนหน้านี้ มาถึงตอนนี้เราจะเอาส่วนของการ upload ของเราไปใช้ในการเขียน blog ของเรา กันครับ

โดยทำการเพิ่ม code ส่วนนี้เข้าไปที่ template ของเรา

```
<form enctype="multipart/form-data" novalidate>
  <div class="dropbox">
    <input type="file" multiple :name="uploadFieldName"
      :disabled="isSaving" @change="filesChange($event.target.name,
      $event.target.files); fileList = $event.target.files.length">
```

```
accept="image/*" class="input-file">
<!-- <p v-if="isInitial || isSuccess"> -->
<p v-if="isInitial">
    Drag your file(s) here to begin<br> or click to browse
</p>
<p v-if="isSaving">
    Uploading {{ fileCount }} files...
</p>
<p v-if="isSuccess">
    Upload Successful.
</p>
</div>
</form>
```

โดยผู้เข้าไว้ต่อจาก

```
<p>title: <input type="text" v-model="blog.title"></p>
```

จากนั้นผู้ทำการเพิ่ม style ของผู้เข้าไปครับ

```
<style scoped>
/* upload */
.dropbox {
    outline: 2px dashed grey; /* the dash box */
    outline-offset: -10px;
    background: lemonchiffon;
    color: dimgray;
    padding: 10px 10px;
    min-height: 200px; /* minimum height */
    position: relative;
    cursor: pointer;
}

.input-file {
    opacity: 0; /* invisible but it's there! */
    width: 100%;
    height: 200px;
```

```
        position: absolute;
        cursor: pointer;
    }

.dropbox:hover {
    background:khaki; /* when mouse over to the drop zone, change color
*/
}

.dropbox p {
    font-size: 1.2em;
    text-align: center;
    padding: 50px 0;
}
</style>
```

ตอนนี้เราได้แก้ไข template และ style หรือหน้าตาของส่วน upload ของเรา
เรียบร้อยแล้ว ต่อมาเราจะทำการเขียน script กันครับ ผ่าน Import Upload
Service ของเราเข้ามาใช้งานก่อนครับ

```
import UploadService from '@/services/UploadService'
```

ทำการเพิ่มตัวแปรเพื่อแสดงสถานะต่าง ๆ ในการ upload

```
const STATUS_INITIAL = 0,
      STATUS_SAVING = 1,
      STATUS_SUCCESS = 2,
      STATUS_FAILED = 3
```

จากนั้นทำการเพิ่ม data ที่เราต้องใช้ในการ upload ครับ

```
// upload data
BASE_URL: "http://localhost:8081/assets/uploads/",
error: null,
uploadError: null,
```

```
currentStatus: null,  
uploadFieldName: "userPhoto",  
uploadedFileNames: []
```

แล้วทำการเพิ่ม method ในส่วน upload ของเราดังนี้ครับ

```
// upload  
wait(ms) {  
  return x => {  
    return new Promise(resolve => setTimeout(() => resolve(x), ms));  
  };  
},  
reset() {  
  // reset form to initial state  
  this.currentStatus = STATUS_INITIAL  
  // this.uploadedFiles = []  
  this.uploadError = null  
  this.uploadedFileNames = []  
},  
async save(formData) {  
  // upload data to the server  
  try {  
    this.currentStatus = STATUS_SAVING  
    await UploadService.upload(formData)  
    this.currentStatus = STATUS_SUCCESS  
    this.clearUploadResult()  
  } catch (error) {  
    console.log(error)  
    this.currentStatus = STATUS_FAILED  
  }  
},  
filesChange(fieldName, fileList) {  
  // handle file changes  
  const formData = new FormData();  
  
  if (!fileList.length) return;  
  
  // append the files to FormData  
  Array.from(Array(fileList.length).keys()).map(x => {  
    formData.append(fieldName, fileList[x], fileList[x].name);  
    this.uploadedFileNames.push(fileList[x].name);  
  })  
},
```

```
});

// save it
this.save(formData);
},
clearUploadResult: function(){
  setTimeout(() => this.reset(), 5000);
}
```

แล้วทำการเพิ่มในส่วนของ computed และ created เข้าไปครับ

```
computed: {
  isInitial() {
    return this.currentStatus === STATUS_INITIAL;
  },
  isSaving() {
    return this.currentStatus === STATUS_SAVING;
  },
  isSuccess() {
    return this.currentStatus === STATUS_SUCCESS;
  },
  isFailed() {
    return this.currentStatus === STATUS_FAILED;
  }
},
created () {
  this.reset()
}
```

ซึ่งจริง ๆ แล้ว ก็เป็นการ copy code จาก upload file ที่เราเขียนก่อนหน้ามาแปะ ใน create blog นั่นเอง จากนั้นลองทดสอบการ upload ดูครับว่าสามารถทำได้ ถูกต้องหรือไม่อย่างไร ถ้า error ให้เช็คเปลี่ยนเทียบกับโปรแกรมส่วน upload ที่ เราทำไว้ work แล้วดูนะครับ

แสดงผลไฟล์รูปภาพที่เรา upload ขึ้นไป

จากที่ผ่านมาเราสามารถทำการ upload ไฟล์รูปภาพของเราขึ้น Server ได้เป็นที่เรียบร้อยผ่านหน้า create blog หรือ เขียน blog ของเรา ในตอนนี้เราต้องการที่เห็นภาพที่เรา upload เข้าไป และเปลี่ยน blog ที่เราเขียน หรือใช้มันเป็น thumbnail ของ blog ของเรา กัน เพิ่ม

เรามาทำส่วนการแสดงผลภาพที่เราทำการ upload ขึ้นไปที่ server กันก่อนครับ โดยผมทำการ code เพิ่มเติมในส่วนของ template ดังนี้ครับ 追加เพิ่มต่อท้าย<div class="dropbox">

```
<div>
  <ul class="pictures">
    <li v-for="picture in pictures" v-bind:key="picture.id">
      
    </li>
  </ul>
  <div class="clearfix"></div>
</div>
```

จากนั้นผมทำการสร้าง

```
pictures: [],  
pictureIndex: 0,
```

เพิ่มใน data ~~เพิ่มทำการเก็บข้อมูลภาพที่เราทำการ upload ขึ้น server ของเรา~~ เพิ่ม

ใน function save (formData) ของเราจะแก้ไข code เพื่อเพิ่มส่วนการแสดงผลของเราดังนี้ code ของ save(formData) ต้องเพิ่ม Code เหล่านี้

```
try {
  this.currentStatus = STATUS_SAVING
  await UploadService.upload(formData)
  this.currentStatus = STATUS_SUCCESS

  // update image uploaded display
  let pictureJSON = []
  this.uploadedFileNames.forEach(uploadFilename => {
    let found = false;
    for(let i=0;i<this.pictures.length;i++){
      if(this.pictures[i].name == uploadFilename){
```

```

        found = true
        break
    }
}

if(!found){
    this.pictureIndex++
    let pictureJSON = {
        id: this.pictureIndex,
        name: uploadFilename
    }
    this.pictures.push(pictureJSON)
}
})

this.clearUploadResult()
} catch (error) {
    console.log(error)
    this.currentStatus = STATUS_FAILED
}
},

```

โดยการส่วนที่เพิ่มเข้าไป คือ

```

// update image uploaded display
let pictureJSON = []
this.uploadedFileNames.forEach(uploadFilename => {
    let found = false;
    for(let i=0;i<this.pictures.length;i++){
        if(this.pictures[i].name == uploadFilename){
            found = true
            break
        }
    }

    if(!found){
        this.pictureIndex++
        let pictureJSON = {
            id: this.pictureIndex,
            name: uploadFilename
        }
        this.pictures.push(pictureJSON)
    }
})

```

```
    }
})
```

จากนั้นทำการเพิ่ม style เพื่อไม่ให้รูปภาพของเราระเบิดไปตามขนาดของรูปภาพ และแสดงผลได้สวยงามขึ้น ดังนี้

```
ul.pictures {
    list-style: none;
    padding: 0;
    margin: 0;
    float: left;
    padding-top: 10px;
    padding-bottom: 10px;
}

ul.pictures li {
    float: left;
}

ul.pictures li img {
    max-width: 180px;
    margin-right: 20px;
}

.clearfix {
    clear: both;
}
```

การทำงานของ code ส่วนนี้ เริ่มจากผู้สร้าง pictureJSON ขึ้นมาเพิ่มเก็บ ข้อมูล ที่ไฟล์ที่เราทำการ upload โดยผู้เก็บไว้สองอย่างคือ id และ ชื่อไฟล์ เพื่อจะนำไปใช้ ในส่วน แสดงผล หรือ template html ของเรา ด้วย v-for นั้นเอง โดย id ของผู้ ใช้ pictrueIdex ที่สร้างไว้ใน data เพิ่มขึ้นตามจำนวนของ ไฟล์รูปภาพที่ทำการ upload ขึ้นไป โดยการที่ใส่ id เข้าไปทำให้เราอ้างอิงถึงไฟล์รูปของเราได้ง่ายขึ้น

เมื่อทำการทดสอบการ upload ไฟล์ของเราในส่วนเขียน blog ผล คือ เมื่อเราทำงาน upload ไฟล์ขึ้น server สำเร็จ ไฟล์ที่ถูก upload จะแสดงผลขึ้นมาทันทีดังรูป

← → C i localhost:8080/blog/create

Create Blog

title:

Drag your file(s) here to begin
or click to browse

content:

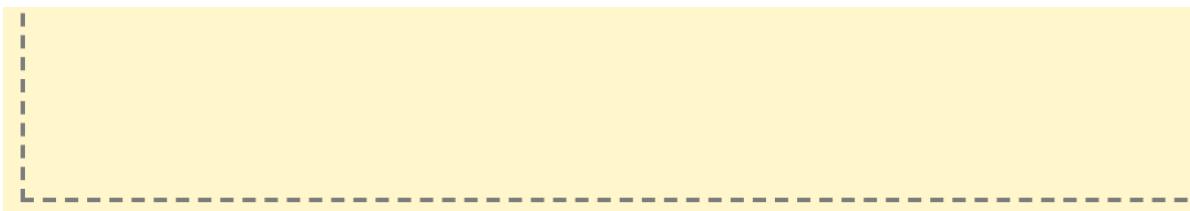
ทำการลบรูปที่ upload ขึ้นไป

หลังจากที่เราทำการเขียน code เพื่อ upload รูปภาพและแสดงผลการ upload ของเราได้เรียบร้อยแล้ว เราจะมาทำการลบรูปภาพของเราที่ upload ไปแล้วกันครับ เพราะบางที่เรา upload ผิด หรือไม่ได้ต้องการใช้รูปภาพนั้น ใน blog ครับ

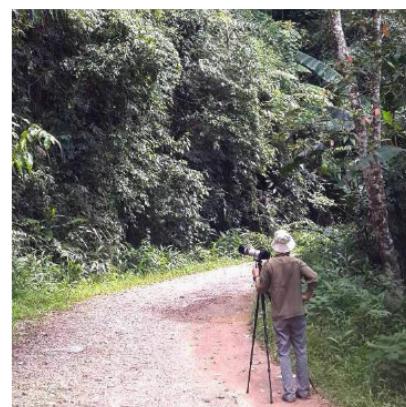
โดยเราจะทำการเพิ่มปุ่มกันสำหรับลบรูปภาพของเราก่อนใน template ของเราดังนี้ครับ

```
<ul class="pictures">
  <li v-for="picture in pictures" v-bind:key="picture.id">
    
    <br />
    <button v-on:click.prevent="delFile(picture)">Delete</button>
  </li>
</ul>
```

ปุ่มของผมจะแสดงขึ้นมาใต้ภาพที่ผมทำการ upload ดังภาพครับ



Delete



Delete

content:

จากนั้นผมทำการเขียน method เพื่อลบรูปภาพที่เราอัพโหลดขึ้นไปในระบบของเรา โดยทำการเพิ่ม method ในการลบไฟล์ของเราตามชื่อที่เราได้ใสไวเพื่อ

เรียกใช้ใน “v-on:click.prevent” คือ “delFile(picture)” โดยส่ง picture ไปลบ
นั้นเองครับ

```
async delFile (material){
    let result = confirm("Want to delete?")
    if (result) {
        let dataJSON = {
            "filename": material.name
        }

        await UploadService.delete(dataJSON)

        for (var i=0;i<this.pictures.length;i++){
            if(this.pictures[i].id === material.id) {
                this.pictures.splice(i, 1)
                this.materialIndex--
                break
            }
        }
    }
},
```

จากนั้นทำการทดสอบดูครับ โดยการ upload file ขึ้นไป แล้วเปิดโฟล์เดอร์ที่เราใช้
เก็บภาพขึ้นมา เมื่อทำการกดปุ่มลบไฟล์ที่ Front End ในส่วนของ Back Office
ของเรา ไฟล์บน server ของเราจะถูกลบไปด้วยนั่นเอง

เพิ่มข้อมูลไฟล์ที่เราอัพโหลดไว้ใน database ของเรา

มาถึงตอนนี้เราได้ทำการอัพโหลดและลบไฟล์ ใน server หรือ Back End ของเรา
ได้แล้ว ตอนนี้เราจะทำการใส่ข้อมูลไฟล์ที่เราทำการอัพโหลดไว้ในฐานข้อมูลของ
เราด้วย เวลาที่เราเปิดอ่าน หรือแก้ไข จะได้แสดงผลได้อย่างถูกต้องครับ

ก่อนหน้านี้ใน data () ของเรามี ตัวแปรที่ชื่อว่า blog อยู่และใน blog เรา มีฟิลด์ที่
ชื่อว่า pictures อยู่ โดยผนกำหนดไว้ให้เป็นค่า null ตอนเราเริ่มเขียนโปรแกรมกัน

```
// blog data
blog: {
    title: '',
    thumbnail: 'null',
```

```
    pictures: 'null',
    content: '',
    category: '',
    status: ''
},
```

โดยเราจะทำการ update ค่าของ blog.pictures กันก่อนการบันทึกข้อมูลนั้นเอง โดยผมทำการแก้ไข code ในส่วน method ของเรา ที่ createBlog () ดังนี้

```
async createBlog () {
  this.blog.pictures = JSON.stringify(this.pictures)
  try {
    await BlogsService.post(this.blog)
    this.$router.push({
      name: 'blogs'
    })
  } catch (err) {
    console.log(err)
  }
},
```

โดยผมทำการแปลง JSON pictures ที่เก็บข้อมูลไฟล์ภาพที่เราอัปโหลด จาก json object เป็น string ก่อนทำการเก็บข้อมูลข้อมูลของเรานั่นเอง

ทำการทดสอบการสร้าง blog พร้อมรูปภาพของเรา

มาถึงตอนนี้เราสามารถเขียน Blog พร้อมใส่รูปภาพของเราเข้าไปใน blog ที่เราเขียนได้แล้วนั่นเอง เราจะทดสอบโดย upload ไฟล์และบันทึกข้อมูล หรือกด create blog และทำการเปิดที่ database ของเราเพื่อตรวจสอบดูว่า pictures ของเราถูกบันทึกไว้หรือไม่

	title	thumbnail	pictures
1	blog title put xx	null put	null put
2	blog title	null	null
3	ckeditor test	null	null
4	5	null	null
5	test upload	null	null
6	test upload again	null	[{"id":1,"name":"11052025_1655477558033853_3304455179145642685_o.jpg"}, {"id":2,"name..."]

ตอนนี้เราได้ทำการเก็บข้อมูลภาพที่เราใช้ blog ของเราไว้เรียบร้อยแล้วนั่นเอง

นำรูปภาพมาใช้ Thumbnail

ในขั้นตอนนี้เราจะนำรูปภาพมาแสดงใน Thumbnail ของเรา กัน โดยทำการเพิ่ม ส่วนการแสดง Thumbnail ของเรา โดยการเพิ่ม code ในส่วน template ของ create blog ดังนี้

```
<p>title: <input type="text" v-model="blog.title"></p>
<div class="thumbnail-pic" v-if="blog.thumbnail != 'null'">
  
</div>
```

และปุ่มสำหรับเพิ่ม thumbnail ของเราดังนี้

```
<button v-
on:click.prevent="useThumbnail(picture.name)">Thumbnail</button>
```

จากนั้นทำการ เพิ่ม style เพื่อความคุณขนาดของภาพที่จะแสดงใน thumbnail โดย การเพิ่ม code ใน `<style></style>`

```
/* thumbnail */
.thumbnail-pic img{
  width:200px;
}
```

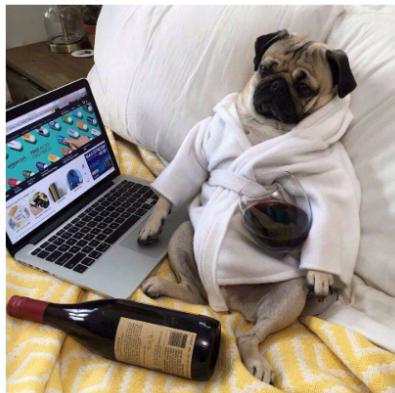
จากนั้นทำการเพิ่ม method เพื่อทำการ update thumbnail ใน methods: {

```
useThumbnail (filename) {  
  console.log(filename)  
  this.blog.thumbnail = filename  
},
```

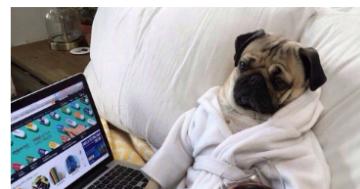
เมื่อเขียน code เสร็จแล้ว เราจะทำการทดสอบกัน โดยทำการ upload และ กดปุ่ม thumbnail เพื่อเพิ่ม thumbnail ของเรากัน thumbnail จะแสดงขึ้นทันที จากนั้นทำการกด create blog เพื่อทำการบันทึกข้อมูล

Create Blog

title:



Drag your file(s) here to begin
or click to browse



Thumbnail ของเรามาแล้วนั่นเองครับ

ปรับปรุงเรื่องการแสดงผลแบบ fade-in และ fade-out

ใน vuejs มีการแสดงผลแบบ fade in หรือการค่อย ๆ ปรากฏขึ้น ทำให้การแสดงผลไม่วุ่นวายเกินไป การใช้งานนั้นง่ายมาก โดยผมจะเพิ่ม style ของผมไว้ใน src/App.vue เลยนะครับ เพราะจะได้ใช้ร่วมกันจากหลาย Vue Component ของเราได้

```
/* Fade transition */
.fade-enter, .fade-leave-to {
    opacity: 0;
}

.fade-enter-active, .fade-leave-active {
    transition: opacity 0.5s;
}

.fade-enter-to {
    opacity: 1;
}
```

เวลาเราจะใช้งานเราก็แค่ เอา tag

```
<transition name="fade"> </transition>
```

ไปครอบไว้แบบนี้ครับ

```
<transition name="fade">
  <div class="thumbnail-pic" v-if="blog.thumbnail != 'null'">
    
  </div>
</transition>
```

ลองทดสอบโดยการกด thumbnail เมื่อก่อนหน้านี้ดูครับ thumbnail ของเราจะค่อย ๆ แสดงผลขึ้นมาหนึ่น点儿เอง

ในกรณีที่เราต้องการใช้งาน transition กับ tag เราสามารถใช้ <transition-group> แทน tag ของเราได้เลย ดังนี้ครับ

```
<transition-group tag="ul" name="fade" class="pictures">
  <li v-for="picture in pictures" v-bind:key="picture.id">
```

```


<br />
<button v-on:click.prevent="useThumbnail(picture.name)">Thumbnail</button>
<button v-on:click.prevent="delFile(picture)">Delete</button>
</li>
</transition-group>

```

ลองเอาไปประยุกต์ใช้กับส่วนอื่น ๆ ดูนะครับ ต่อไปจะไม่กล่าวถึงแล้วนะครับ

นำรูปภาพไปแสดงใน Editor ของเรา

การที่เราจะแสดงรูปภาพของเราใน ckeditor ของเรา นั้น เราต้องทำการเพิ่ม Toolbar เกี่ยวกับ picture ของเราใน ckeditor เลียก่อน ซึ่งเราได้เพิ่ม Toolbar ไว้เรียบร้อยแล้วจาก code ชุดนี้นี่เองครับ

```

created () {
  this.reset(),

  this.config.toolbar = [
    {
      name: "document",
      items: [
        "Source",
        "-",
        "Save",
        "NewPage",
        "Preview",
        "Print",
        "-",
        "Templates"
      ]
    },
    {
      name: "clipboard",
      items: [
        "Cut",
        "Copy",
        "Paste",
        "PasteHTML",
        "PasteFromWord",
        "-",
        "Undo",
        "Redo"
      ]
    }
  ]
}

```

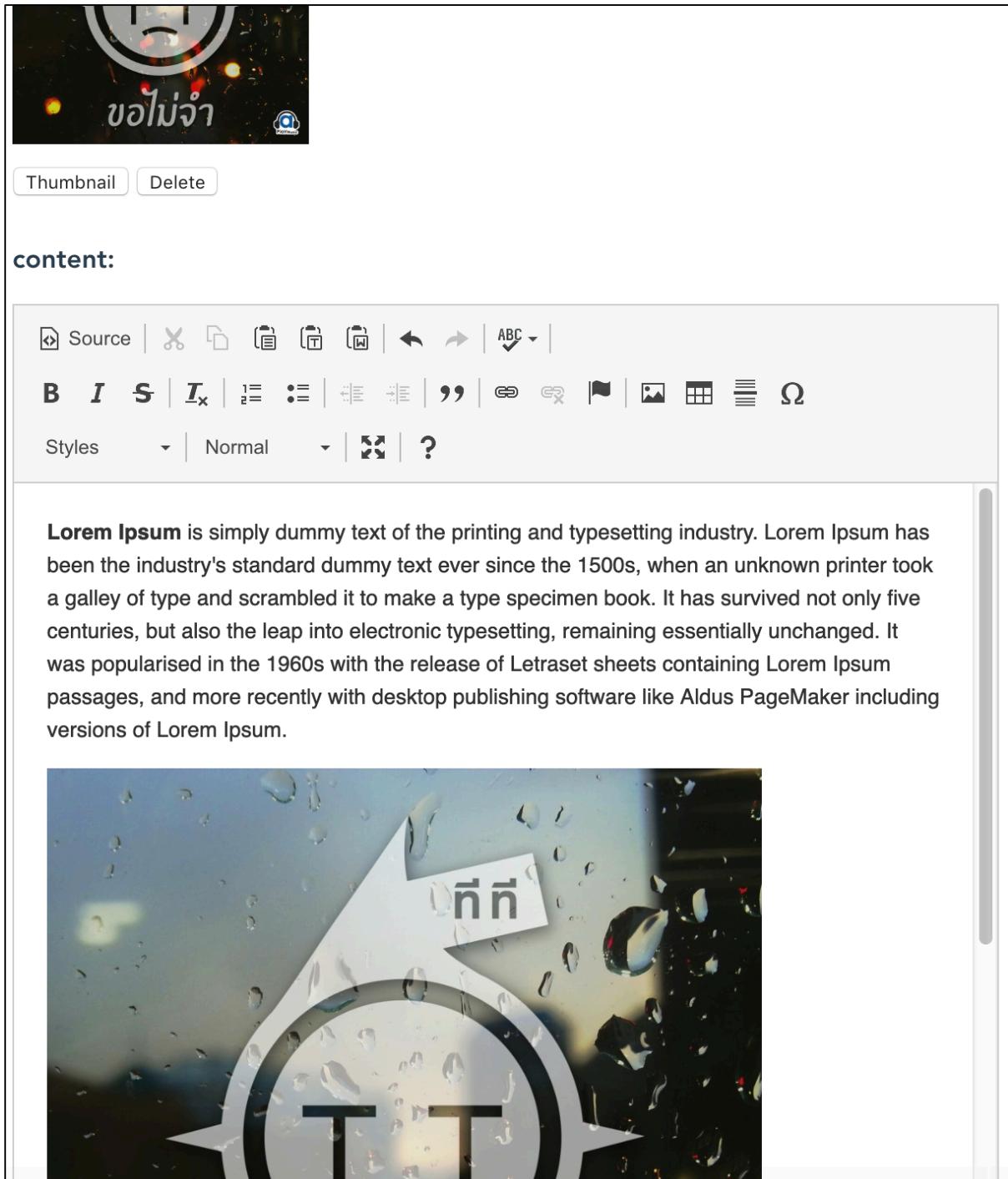
```
"PasteText",
"PasteFromWord",
"-",
"Undo",
"Redo"
]
},
{
  name: "editing",
  items: ["Find", "Replace", "-", "SelectAll", "-", "Scayt"]
},
{
  name: "forms",
  items: [
    "Form",
    "Checkbox",
    "Radio",
    "TextField",
    "Textarea",
    "Select",
    "Button",
    "ImageButton",
    "HiddenField"
  ]
},
"/",
{
  name: "basicstyles",
  items: [
    "Bold",
    "Italic",
    "Underline",
    "Strike",
    "Subscript",
    "Superscript",
    "-",
    "CopyFormatting",
    "RemoveFormat"
  ]
},
{
  name: "paragraph",
  items: [
    "NumberedList",
```

```

        "BulletedList",
        "-",
        "Outdent",
        "Indent",
        "-",
        "Blockquote",
        "CreateDiv",
        "-",
        "JustifyLeft",
        "JustifyCenter",
        "JustifyRight",
        "JustifyBlock",
        "-",
        "BidiLtr",
        "BidiRtl",
        "Language"
    ]
},
{ name: "links", items: ["Link", "Unlink", "Anchor"] },
{
    name: "insert",
    items: [
        "Image",
        "Flash",
        "Table",
        "HorizontalRule",
        "Smiley",
        "SpecialChar",
        "PageBreak",
        "Iframe",
        "InsertPre"
    ]
},
"/",
{ name: "styles", items: ["Styles", "Format", "Font", "FontSize"] },
{ name: "colors", items: ["TextColor", "BGColor"] },
{ name: "tools", items: ["Maximize", "ShowBlocks"] },
{ name: "about", items: ["About"] }
]
}

```

ถึงตอนนี้ Toolbar ของเราจะมีเพิ่มขึ้นมา焉ากมาย และการ Upload รูปภาพของเรา ก็ work แล้ว ผมจะทดสอบส่วนรูปภาพใน editor ของเรา โดยการลากรูปภาพมา วางเลยครับ บังโกสำเร็จแล้วครับ



The screenshot shows the CKEditor interface. At the top, there is a thumbnail of an image featuring a circular logo with the text "ขอain์จ้า" and a small "CK" icon. Below the thumbnail are two buttons: "Thumbnail" and "Delete". The main content area contains a block of text in English: "Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum." Above this text is the CKEditor toolbar, which includes icons for Source, Cut, Copy, Paste, Undo, Redo, ABC, Bold, Italic, Strike, Subscript, Superscript, Paragraph styles (Normal, Heading 1-5), and a question mark icon.

content:

Styles Normal ?

Content:

Text:

Image:

โดยเราสามารถปรับขนาดและรายละเอียดของรูปภาพได้โดย คลิกขวา ที่รูปภาพ ของเราใน ckeditor ครับ และทำการจัดการได้ตามที่ต้องการเลยครับ

The screenshot shows a blog editor interface. At the top, there's a header with the text "ຂວາມຈຳ" and a small logo. Below the header are two buttons: "Thumbnail" and "Delete".

The main area is labeled "content:" and contains a rich text editor toolbar with various icons for bold, italic, underline, etc. Below the toolbar is a preview area showing a blurred image of water droplets on a surface.

A context menu is open over the image in the preview area, enclosed in a red box. The menu items are:

- Cut (⌘+X)
- Copy (⌘+C)
- Paste (⌘+V)
- Image Properties

At the bottom of the editor, there's a code preview section showing "body p img" and a category selection field labeled "category: []".

ตอนนี้ส่วนการจัดการเกี่ยวกับรูปภาพทั้งหมดของเราในการเขียน Blog ของเรา การเขียน blog นั้นเสร็จสมบูรณ์แล้วต่อไปเราจะไปจัดการเรื่องการแก้ไข blog ของเรา กันครับ

ทำการปรับปรุง code เกี่ยวกับรูปภาพ ในส่วน Edit Blog

การปรับปรุง code ในส่วน Edit Blog นั้นง่ายมาก เราทำการ copy ส่วนที่ edit blog ไม่มีมาจาก create blog และใส่ให้เหมือนกันได้เลยครับ โดยผมจะไม่อธิบาย นะครับ

แต่จะมีส่วนที่เราต้องเขียนเพิ่ม คือ ดังนี้ครับ

```
this.blog.pictures = JSON.stringify(this.pictures)
```

จากที่ใส่ function createBlog ก็มาใส่ใน function updateBlog แทน

และการ แปลง blog.pictures ที่เป็น string อยู่จากฐานข้อมูลของเรามาให้กลายเป็น json object ดังนี้

```
this.pictures = JSON.parse(this.blog.pictures)
```

โดยผมใส่ไว้ที่นี่ครับ

```
try {
  let blogId = this.$route.params.blogId
  this.blog = (await BlogsService.show(blogId)).data
  this.pictures = JSON.parse(this.blog.pictures)
  this.pictureIndex = this.pictures.length
} catch (error) {
  console.log(error)
}
```

แต่เนื่องจาก ตอนนี้ผมต้องการไป update ข้อมูลในส่วนของ Thumbnail และการ แสดงผล ผลโดยเอา function ในการอ่านค่าของ pm ไว้ใน lifecycle mounted () แทนดังนี้

```
async mounted () {
  try {
    let blogId = this.$route.params.blogId
    this.blog = (await BlogsService.show(blogId)).data
    this.pictures = JSON.parse(this.blog.pictures)
  } catch (error) {
    console.log(error)
  }
},
```

มาถึงตอนนี้เมื่อเราทำการแก้ไข blog ของเรา รูปภาพของเราจะขึ้นหมวดแล้ว ทั้ง thumbnail และ ภาพที่เรา upload ไว้ เราสามารถจัดการได้เหมือน create blog เลยครับ

ในตอนนี้เราวัดการเรื่องการ Upload รูปภาพและการแสดงผลทั้งในการเขียน และ แก้ไข blog ของเราได้เรียบร้อยแล้ว จากที่ผ่านมาตอนนี้เราทำ Engine สำเร็จไป ก็อบหมดแล้ว เดินทางต่อไปอีกเล็กน้อย เราจะทำเว็บไซต์ของเราได้อย่างสมบูรณ์ ขึ้นครับ

บทที่ 14 ปรับปรุงการ Back End และส่วนการค้นหา (Search Component)

ในบทนี้เราจะทำการแสดงผลให้ดีกว่าขึ้นกว่าเดิม โดยการแสดงรายละเอียดของ blog ให้ครบถ้วน ดูดีขึ้นมาอีกหน่อย รวมถึงแสดง blog thumbnail พร้อมทั้งแสดง abstract ของ blog ของเรา ต่อจาก Title ด้วย

จัดการ การแสดงผลแบบแบ่งช่วง การจัดการการโหลดข้อมูลด้วยการ scroll และ การค้นหาจาก keywords และ category ของเรา กันครับ

เริ่มต้นจัดการ การแสดงผลกันก่อนเลยครับ

อย่างแรกเลย ผู้จะทำการลบ ข้อมูล Blog เก่าของผู้อภิภัต์หน้าก่อนเลยครับ จำนวน blog ของผู้ทั้งหมด เป็น 0 ครั้งแต่ผู้ต้องการให้มันแจ้งด้วยว่า 'ไม่มีข้อมูล' ครับ ทำการเปิดไฟล์ client ของเรารอแล้วทำการเปิด src/components/Blogs/Index.vue ขึ้นมาครับ และทำการแก้ไข code ใน <template></template> เป็นดังนี้

```
<h4>จำนวน blog {{blogs.length}}</h4>
<div v-if="blogs.length === 0" class="empty-blog">
    *** ไม่มีข้อมูล ***
</div>
```

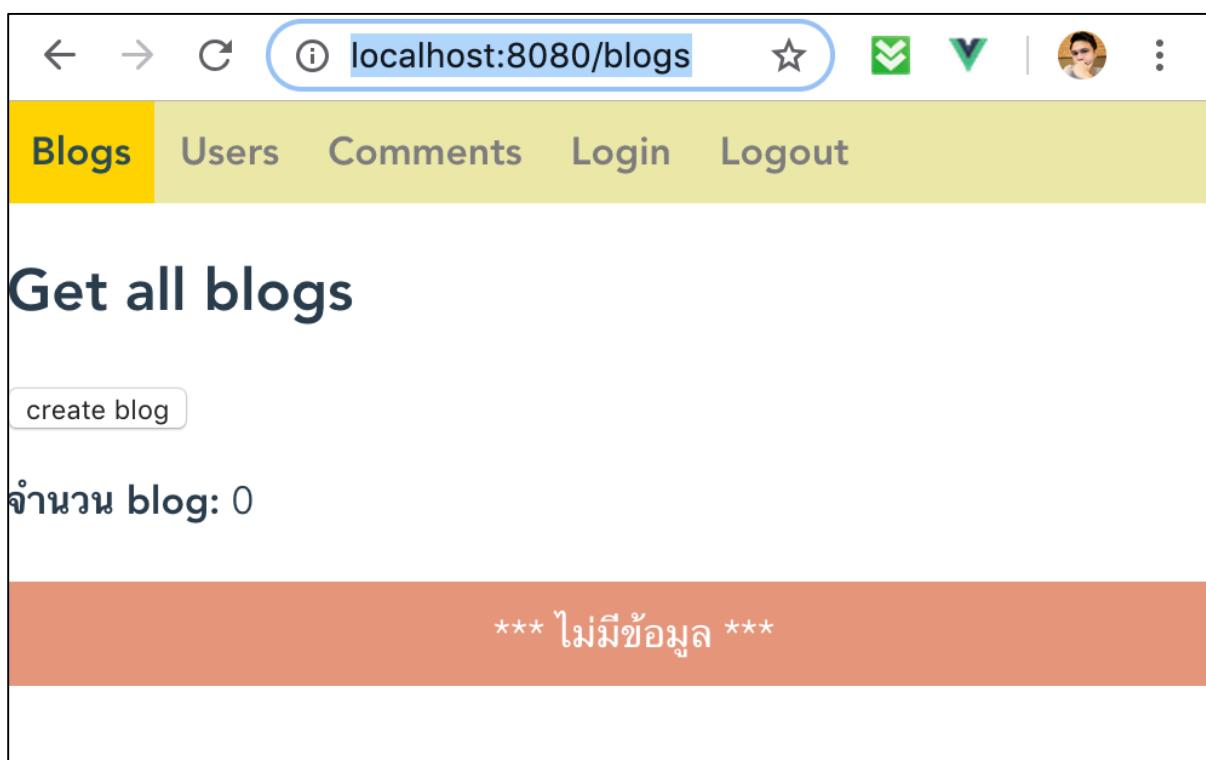
จากนั้นทำการเพิ่ม class empty-blog ใน <style></style>

```
<style scoped>
.empty-blog {
    width: 100%;
    text-align: center;
    padding: 10px;
    background: darksalmon;
    color: white;
}
</style>
```

ทำการเปิด src/App.vue ขึ้นมา และเพิ่ม code ใน <style></style> ดังนี้ครับ

```
body {  
  margin:0;  
  padding:0;  
}
```

เพื่อให้ขอบมันชิดไม่น่าหงุดหงิดอย่างเคย จากนั้นทำการเปิด web browser เรียกไปที่ <http://localhost:8080/blogs> จะได้ผลดังนี้ครับ



แก้ code ไปนิดเดียวดูดีขึ้นมาเลยครับ

จากนั้นทำการเพิ่มข้อมูล blog ของเรารสัก 5-10 ข้อมูล สรุปอะไรให้ครบนะครับจะได้เขียน code กันได้ไม่พลาด

ใส่ข้อมูลเสร็จแล้ว เปิด src/components/Blogs/Index.vue ขึ้นมาครับ

มาตอนนี้เราจะปรับหน้าตาของส่วนนี้ให้เราดูดีขึ้นมาอีกหน่อยครับ โดยการทำ การเขียน code ในส่วน <template></template>

```
<template>  
  <div>
```

```

<div class="blog-header">
    <h2>សំណងការប្រតិទិន</h2>
    <div>
        <button v-on:click="navigateTo('/blog/create')">create blog</button>
        <strong> ចំណាំ blog: {{blogs.length}}</strong>
        <br>
    </div>
    <div v-if="blogs.length === 0" class="empty-blog">
        *** មិនមែនម៉ូល ***
    </div>
    <div v-for="blog in blogs" v-bind:key="blog.id" class="blog-list">
        <!-- <p>id: {{ blog.id }}</p> -->
        <div class="blog-pic">
            <transition name="fade">
                <div class="thumbnail-pic" v-if="blog.thumbnail != 'null'">
                    
                </div>
            </transition>
        </div>
        <h3>{{ blog.title }}</h3>
        <div v-html="blog.content.slice(0,200) + '...'"></div>
        <div class="blog-info">
            <p><strong>Category:</strong> {{ blog.category }}</p>
            <p><strong>Create:</strong> {{ blog.createdAt }}</p>
            <!-- <p>status: {{ blog.status }}</p> -->

            <p>
                <button v-on:click="navigateTo('/blog/' + blog.id)">ក្នុង blog</button>
                <button v-on:click="navigateTo('/blog/edit/' + blog.id)">កែតាំង blog</button>
                <button v-on:click="deleteBlog(blog)">លើកខ្លួនូយ៍</button>
            </p>
        </div>
        <div class="clearfix"></div>
    </div>
</template>

```

ໃនស៊ុននេះមី code ពីនោសនី ឬយុទ្ធប៌នី គឺ

```
<div v-html="blog.content.slice(0,200) + '...'"></div>
```

ឬយុទ្ធប៌នី នឹងបង្ហាញការប្រព័ន្ធរបស់ការប្រព័ន្ធ ដែលបានបង្ហាញឡើង។

และ slice คือ การตัดคำใน blog content ของเราเพื่อมาแสดงเป็น abstract ของเรานั่นเอง

ทำการเพิ่มตัวแปร baseURL ใน data () ของเรา

```
BASE_URL: "http://localhost:8081/assets/uploads/",
```

จากนั้นเราจะทำการเพิ่ม style ใน tag <style></style> ของเราดังนี้

```
<style scoped>
.empty-blog {
  width: 100%;
  text-align: center;
  padding: 10px;
  background: darksalmon;
  color: white;
}

/* thumbnail */
.thumbnail-pic img {
  width: 200px;
  padding: 5px 10px 0px 0px;
}

.blog-info {
  float: left;
}

.blog-pic {
  float: left;
}

.clearfix {
  clear: both;
}

.blog-list {
  border: solid 1px #dfdfdf;
  margin-bottom: 10px;
  max-width: 900px;
  margin-left: auto;
  margin-right: auto;
  padding: 5px;
```

```

    box-shadow: 0 2px 4px 0 rgba(0,0,0,.1);
}

.blog-header {
    max-width: 900px;
    margin-left: auto;
    margin-right: auto;
}

</style>

```

ตอนนี้หน้าตาของเว็บไซต์ของเราก็จะดีขึ้นอีกหน่อยล่ะครับ

The screenshot shows a web application interface for a blog. At the top, there is a navigation bar with tabs: 'Blogs' (which is highlighted in yellow), 'Users', 'Comments', 'Login', and 'Logout'. Below the navigation bar, the main content area has a heading 'ส่วนจัดการบล็อก' (Blog Management). It displays three blog entries, each in its own card:

- Blog 1 ของเรา**: An image of a BMW car and a Rolex watch. The text reads: 'Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown pr...'. Category: nodejs. Create: 2019-03-17T16:22:53.657Z. Buttons: ดู blog, แก้ไข blog, ลบข้อมูล.
- Blog 2 ของเรา**: An image of a person using a telescope. The text reads: 'Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown pr...'. Category: vuejs. Create: 2019-03-17T16:26:17.967Z. Buttons: ดู blog, แก้ไข blog, ลบข้อมูล.
- Blog 3 ของเรา**: An image of a man's face. The text reads: 'Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown pr...'. Category: nodejs. Create: 2019-03-17T16:27:22.875Z. Buttons: ดู blog, แก้ไข blog, ลบข้อมูล.

ทำส่วนค้นหาใน blog ของเรา

เตรียม api บน Server ของเราให้พร้อมก่อน

มาถึงตอนนี้เราจะทำการ เขียน code ในส่วนค้นหา หรือ search ของเรา กัน โดย ส่งค่าจาก form search ของเราไปค้นหาข้อมูลของเรานั้น server หรือ back end ของเรา กัน โดยจะมี code api ใน server ของผู้นิดหน่อย โดยทำการเปิดไฟล์ src/controllers/BlogController.js และทำการแก้ไข function index () เป็นดังนี้ครับ

```
// idx with serach blog
async index (req, res) {
  try {
    let blogs = null
    const search = req.query.search
    // console.log('search key: ' + search)

    if (search) {
      blogs = await Blog.findAll({
        where: {
          $or: [
            'title', 'content', 'category'
          ].map(key => ({
            [key]: {
              $like: `%"${search}"%`,
            }
          })),
        },
        order: [['updatedAt', 'DESC']]
      })
    } else {
      blogs = await Blog.findAll({
        order: [['updatedAt', 'DESC']]
      })
    }
    res.send(blogs)
  } catch (err) {
```

```

    res.status(500).send({
      error: 'an error has occurred trying to fetch the blogs'
    })
  },
},

```

ใน code ส่วนนี้ไม่ได้มีอะไรเพิ่มขึ้นมากนัก เพียงแต่อ่านค่า search ที่ front end ส่งมา หากไม่ส่งมา Blog Model ของเราก็จะส่งข้อมูลทั้งหมดออกไปให้เมื่อเรา index () ใน version ก่อนหน้า แต่ถ้ามี search params แนบมา ก็จะไปทำการค้นหาในฟิลด์ ที่เรากำหนดคือ title content และ category ตามลำดับ

จัดการส่วนค้นให้ front end ของเรา

เมื่อเราทำการแก้ไข code ใน server หรือ Back End ของเรา สิ่งที่เราควรนึกถึง เป็นอันดับแรก ๆ คือ service ของเรา ที่เชื่อมต่อกับ api นั้น ๆ ก่อนหน้านี้เรา แก้ไข api ของเราไปแล้ว ให้รับ หรือสามารถ แนบ search keywords ไปค้นหาได้ ดังนั้นเราต้องทำการส่ง search keywords เหล่านั้น ไปให้ api ด้วย เช่นกัน

ทำการเปิดไฟล์ src/services/BlogsServices.js ขึ้นมา และทำการแก้ไข code ในส่วน function index () ของเราเป็นดังนี้

```

index (search) {
  return Api().get('blogs', {
    params: {
      search: search
    }
  })
},

```

โดยเป็นการ แนบ search keywords ส่งไปที่ api ของเราแล้ว

จากนั้นทำการเปิด src/components/Blogs/Index.vue ของเราขึ้นมา ทำการเพิ่ม form ในการค้นหาของเรา โดยผูกกำหนดตัวแปร ในการค้นหา ชื่อ search เชื่อมต่อกับ script ของเราผ่าน v-model นั้นเอง โดยผูกทำการแก้ไข code เป็นดังนี้

```
<h2>ส่วนจัดการล็อก</h2>
```

```
<div>
  <form>
    <input type="text" v-model="search" placeholder="Search">
  </form>
</div>
```

ทำการเพิ่มตัวแปร เพื่อใช้ในการค้นหาใน data () ของเรา

```
search: '',
```

จากนั้นจะใช้ watch ซึ่งเป็น propertie หนึ่งของ vuejs ทำงานคล้ายกับ computed () แต่ computed () จะ return ค่าตาม function ที่ใช้จัดการออกมาเมื่อมีการเปลี่ยนแปลงค่า data ที่มันเกี่ยวข้อง แต่ watch นี้จะเป็นการส่งค่าเข้าไปทำงานใน watch เมื่อ ตัวแปรที่ผูกอยู่มีการเปลี่ยนแปลงนั้นเอง

ในการที่เราใช้ watch นี้เมื่อมีค่าเปลี่ยนแปลงทุก character ของ string เลยซึ่งมีการส่งไป query data ที่ api ตลอดเวลาที่พิมพ์ แต่ไม่ต้องการให้มันเป็นอย่างนั้น ผู้จึงจะใช้ package ที่ชื่อว่า lodash มาช่วยครับ

ทำการติดตั้ง lodash ด้วยคำสั่ง

```
npm install --save lodash
```

ทำการ Import Lodash เข้ามาใน Vue Component ของเรา โดยใส่ code ที่ tag <script></script> ของเราดังนี้

```
import _ from 'lodash'
```

จากนั้นเราจะทำการเขียน code ใน tag <script></script> เพิ่มเข้าไปดังนี้ครับ

```
export default {
  watch: {
    search: _.debounce(async function (value) {
      const route = {
        name: 'blogs'
      }

      if(this.search !== '') {
        route.query = {
          search: this.search
        }
      }

      console.log('search: ' + this.search)
      this.$router.push(route)
    })
  }
}
```

```
}, 700),  
  
'$route.query.search': {  
    immediate: true,  
    async handler (value) {  
        this.blogs = (await BlogsService.index(value)).data  
    }  
},
```

โดย code ข้างต้นมีส่วนที่น่าสนใจ คือ `_.debounce()` ของผู้มีคือ lodash นั่นเอง โดยหลักการทำงาน คือ เมื่อมีการเปลี่ยนแปลงให้รอจน 700 ms แล้วก็จะส่งไปที่ api ของเราเพื่อทำการค้นหาในเงื่อนไขจริง แล้วก็จะส่งไปค้นหาตลอดเมื่อมีอนเดิม เพียงแต่ lodash จะเข้าไปเปลี่ยนแปลงค่า `this.search` ทุก 700 ms จากการเปลี่ยนแปลงค่า ทุก ๆ ครั้ง ต่างหากครับ

จากนั้นทำการทดสอบ การค้นหาใน blog ของเรา ผลการทดสอบ `this.blogs` ในของเราจะเปลี่ยนแปลงอย่างอัตโนมัติทันที ที่เราพิมพ์คำค้นหาลงในช่องค้นหาของเรา

The screenshot displays a Vue.js application interface. On the left, there's a header with 'Blogs' (highlighted in yellow), 'Users', 'Comments', 'Login', and 'Logout'. Below the header is a section titled 'ส่วนจัดการบล็อก' (Blog Management). A search bar contains 'nodejs' with a red box around it. Below the search bar is a button 'create blog' and the text 'จำนวน blog: 2'. Two blog cards are listed:

- Blog 5 ของเรา**: An image of a person working on a laptop, followed by the text: "Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown pr...". Category: nodejs. Create: 2019-03-13T04:43:58.495Z. Buttons: ดู blog, แก้ไข blog, ลบข้อมูล.
- Blog 1 ของเรา**: An image of a car with a sad face sticker, followed by the text: "Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown pr...". Category: nodejs.

On the right, the Vue Devtools sidebar shows the component tree and the data store. The component tree highlights the root component with the path <Index> = \$vm0 router-view: /blogs. The data store shows the \$route variable with BASE_URL: "http://localhost:8081/assets" and the blogs array containing two objects. A red box highlights the search term "nodejs" in the data store.

ในตัวอย่างนี้ผมใช้ vue devtools เพื่อทำการดูค่าการเปลี่ยนแปลงในตัวแปร blogs ของเรา ลองทำการทดสอบโดยการป้อน keywords เพื่อทำการค้นหาดู จะสังเกตว่า ค่าของ blogs ของเราจะเปลี่ยนแปลงตาม keywords ที่เราส่งไปนั่นเอง

ทำการโหลด และแสดงผลเป็นส่วน ๆ

จากบทที่ผ่านเราได้การดึงข้อมูลจาก api บน server หรือส่วนของ Back End ของเราได้เรียบร้อยแล้ว ทั้งการดึงข้อมูลทั้งหมด และการดึงข้อมูลจาก server ของเราแบบมีเงื่อน

หากข้อมูลของเรามีจำนวนมาก การอ่านค่าเป็น text data อย่าง json นั้นใช้เวลาไม่มาก หรือที่เรารู้กันว่าเบามาก แต่หากมีการนำค่าที่จาก json ไปทำการโหลดรูปภาพจำนวนมาก ในเวลาเดียวกันสัก 200 ภาพ คงทำให้การแสดงผลของเราง้าว

ลงมาก รวมถึงทำให้การทำทุกอย่างซ้ำๆ เราเลยควรแบ่งส่วนการแสดงผลเป็นส่วน ๆ แล้วแสดงจะดีกว่า

ผมจะใช้งาน scrollmonitor เพื่อช่วยในการ โหลดข้อมูลเพิ่มในแต่ละหน้าครับ โดยผมเริ่มจากการติดตั้ง package scrollmonitor ก่อน

```
npm install --save scrollmonitor
```

เปิดไฟล์ src/components/Blogs/Index.vue ใน client ของเราขึ้นมา ทำการ import เพื่อใช้งานใน Vue Component ของเรา โดยเพิ่ม code ดังนี้

```
import ScrollMonitor from 'scrollMonitor'
```

กำหนดตัวแปรสำหรับจำนวนข้อมูลที่เราจะโหลด เอาไว้นอก export default นะครับ เพราะจะใช้ร่วมกันหลายครั้ง ทำให้เป็น global เอาไว้นี่เอง รวมถึงสร้างตัวแบบ global เอาไว้ ตรวจจับตำแหน่งส่วนท้ายที่เรา想ไว้ดังนี้

```
let LOAD_NUM = 3
let pageWatcher
```

ไปทำการเพิ่ม tags เพื่อทำการตรวจสอบส่วนท้ายสุดของ Loading Page ของเรา ทำการแก้ไขส่วน template ของเราดังนี้ครับ

```
<template>
  <div>
    <div class="blog-header">
      <h2>ส่วนขั้นตอนลึก</h2>
      <div>
        <form>
          <input type="text" v-model="search" placeholder="Search">
        </form>
      </div>
      <div>
        <button v-on:click="navigateTo('/blog/create')">create blog</button>
        <strong> จำนวน blog: </strong> {{blogs.length}}</div>
        <br>
      </div>
      <div v-if="blogs.length === 0" class="empty-blog">
        *** ไม่มีข้อมูล ***
      </div>
      <div v-for="blog in blogs" v-bind:key="blog.id" class="blog-list">
        <!-- <p>id: {{ blog.id }}</p> -->
```

```

<div class="blog-pic">
  <transition name="fade">
    <div class="thumbnail-pic" v-if="blog.thumbnail != 'null'">
      
    </div>
  </transition>
</div>
<h3>{{ blog.title }}</h3>
<div v-html="blog.content.slice(0,200) + '...'"></div>
<div class="blog-info">
  <p><strong>Category:</strong> {{ blog.category }}</p>
  <p><strong>Create:</strong> {{ blog.createdAt }}</p>
  <!-- <p>status: {{ blog.status }}</p> -->

  <p>
    <button v-on:click="navigateTo('/blog/' + blog.id)">ແກ້ໄຂ
blog</button>
    <button v-on:click="navigateTo('/blog/edit/' + blog.id)">ແກ້ໄຂ
blog</button>
    <button v-on:click="deleteBlog(blog)">ລົບຂໍ້ມູນ</button>
  </p>
</div>
<div class="clearfix"></div>
</div>
<div id="blog-list-bottom">Blog Bottom</div>
</div>
</template>

```

จะເຫັນວ່າພມເອາ

```
<div id="blog-list-bottom">Blog Bottom</div>
```

ຕ່ອງຈາກ div v-for ອີ່ວິໂລ ລື ລົດ ຂໍອມລຸຂອງເຮົານີ້ເອງ

ຈາກນີ້ເຮົາຈະສ້າງ cycle updated () ປຶ້ນມາ ເພຣະສ້າງແລະຈັດກາຮັບ scrollmonitor ຂອງເຮົາ ໂດຍພມທຳການເຂົ້າໃນ code ດັ່ງນີ້

```
updated () {
  let sens = document.querySelector('#blog-list-bottom')
  pageWatcher = ScrollMonitor.create(sens)
  pageWatcher.enterViewport(function () {
    console.log('Im here, bottom of blog')
  })
},
```

ส่วนนี้จะ ตรวจจับการ scroll ของเราว่ามาที่ blog-list-bottom ของเราหรือไม่ ก้ามถึงจะแสดงข้อความ ออกมา Console

และผมจะเพิ่ม cycle อีกตัวเพื่อจัดการ clear ค่าต่าง ๆ ของ scrollmonitor ก่อนเริ่มต้นโปรแกรมทุกครั้ง

```
beforeUpdated () {
  if (pageWatcher) {
    pageWatcher.destroy()
    pageWatcher = null
  }
}
```

ลองทดสอบโปรแกรมโดยเลื่อนไปที่ส่วนท้ายสุดของ list ของเรา จะได้ผลดังนี้ครับ

The screenshot shows a blog post titled "Blog 1 ของเรา" with a preview image featuring a circular logo and text. Below the post are categories ("nodejs"), creation date ("Create: 2019-03-13T04:39:04.110Z"), and three buttons: "ดู blog", "แก้ไข blog", and "ลบข้อมูล". At the bottom of the page, the developer tools' "Console" tab is active, displaying the message "[HMR] Waiting for update signal from WDS...". In the log area, there are two entries: "2 Im here, bottom of blog" and "Index.vue?6200:108".

หมายความว่าตอนนี้ เราพร้อมที่จะแบ่งข้อมูลของเราเป็นส่วน ๆ และทำการโหลดเพิ่มเติมด้วย scrollmonitor และครับ

จากนั้นผมทำการเพิ่ม ตัวแปรใน data () ชื่อว่า

```
results: []
```

ขึ้นมาอีกหนึ่งตัว โดยผมจะใช้ json ที่โหลดมาได้ทั้งหมด และค่อยแยกเป็นส่วน ๆ ไปแปะไว้ใน this.blogs ของผม โดยผมทำการเพิ่ม function ใน methods ของผมดังนี้

```
appendResults: function () {
  if (this.blogs.length < this.results.length) {
    let toAppend = this.results.slice(
      this.blogs.length,
      LOAD_NUM + this.blogs.length
    )
    this.blogs = this.blogs.concat(toAppend)
  }
},
```

โดยการทำางานของส่วนนี้ใช้การ slice() json ตาม LOAD_NUM ที่เรากำหนดไว้ และทำการ concat หรือ ต่อเข้าไปใน this.blogs ของเราเอง หรือเราจะใช้คำสั่ง push ก็ได้เช่นกันครับ

จากนั้นผมจะให้ทำการ clear ค่าและ append ข้อมูลเข้าไปทีละส่วน โดยผมจะแก้โปรแกรมในส่วนของการค้นหาของผมดังนี้

```
watch: {
  search: _.debounce(async function (value) {
    const route = {
      name: 'blogs'
    }

    if(this.search !== '') {
      route.query = {
        search: this.search
      }
    }

    console.log('search: ' + this.search)
    this.$router.push(route)
  }, 700),

  '$route.query.search': {
    immediate: true,
    async handler (value) {
```

```
        this.blogs = []
        this.results = []
        this.results = (await BlogsService.index(value)).data
        this.appendResults()
    }
}
},
},
```

สังเกตว่า code ในส่วน created () ของ polymahay ไปแล้ว ผมทำการลบออก เพราะกำหนดการให้ทำงานทันทีเมื่อเริ่มโปรแกรม โดยกำหนด

```
immediate: true,
```

ดังนั้นเมื่อเริ่มทำงาน โปรแกรมของก็จะโหลดค่า blog ทั้งหมด ออกจากนั้นเอง เพราะไม่มี keywords ส่งไปด้วย ทำให้ไม่ต้องการใช้การโหลดข้อมูลจาก ตอน created แต่อย่างใด

จากนั้นเข้าไปแก้ไข code ที่ cycle การ updated () เพื่อทำการเพิ่มข้อมูลเป็นส่วน ๆ เข้าไปทุกครั้ง ดังนี้

```
updated () {
    let sens = document.querySelector('#blog-list-bottom')
    pageWatcher = ScrollMonitor.create(sens)
    pageWatcher.enterViewport(this.appendResults)
},
```

ทำการ refresh และทดสอบเลื่อนลงมาล่างสุด เว็บไซต์ของเราจะค่อยโหลดข้อมูล จนครบนั้นเอง

ทำการเพิ่มส่วนแสดงสถานะการโหลดข้อมูล

เราจะทำการแก้ไข ส่วน div blog-list-bottom ของเราเป็นดังนี้

```
<div id="blog-list-bottom">
    <div v-if="blogs.length === results.length && results.length > 0">
        โหลดข้อมูลครบแล้ว</div>
    </div>
```

เพื่อเช็คจำนวน Items ที่เราโหลดมาว่า เท่ากันหรือยังนั้นเอง

และทำการเพิ่ม style เข้าไปใน tag style ของเราดังนี้

```
#blog-list-bottom{
```

```
padding:4px;  
text-align: center;  
background: seagreen;  
color:white;  
}
```

เมื่อโหลดข้อมูลครบแล้วจะแสดงผลดังนี้ครับ

The screenshot shows a blog post interface. At the top, there are three buttons: 'ดู blog', 'แก้ไข blog', and 'ลบข้อมูล'. Below this is a placeholder image featuring a circular logo with 'กี๊ก' and 'TT' inside, and the text 'ขอไม่จำ' at the bottom. The title 'Blog 1 ของเรา' is displayed in bold. The content area contains placeholder text: 'Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown pr...'. Below the content, the category 'Category: nodejs' is listed, followed by the creation date 'Create: 2019-03-13T04:39:04.110Z'. At the bottom, there are three more buttons: 'ดู blog', 'แก้ไข blog', and 'ลบข้อมูล'.

Below the blog post is a screenshot of a browser's developer tools Console tab. The title bar says 'โหลดข้อมูลครบแล้ว'. The console output shows: 'Security Sources Elements Audits Console Network > ... X' and '[HMR] Waiting for update signal from WDS...'. The URL 'log.js?4244:23' is visible at the bottom right.

ทำ Category เมนูให้ทำการเลือกได้เลย

อันที่จริงแล้วที่วางแผนไว้ตอนแรกจะทำส่วนของ Category อีก Model นึง แต่จะไม่ได้ความรู้สึกไรมาก เพราะจะเหมือนกับ CRUD ที่ว่าไปเลยคิดว่าทำแบบนี้ ถึงจะไม่ถูกต้องมากนัก แต่เราจะได้เล่นกับ JSON ของเรามากหน่อย และน่าจะได้ความรู้มากกว่าด้วย

ผມเริ่มจากสร้าง ตัวแปรสำหรับเก็บ category ของเราใน data () ก่อนเลยครับ

```
category: [],
```

จากนั้นผมทำการแก้ไข code ในส่วนของการค้นหาของผมเป็นดังนี้ครับ

```
watch: {
  search: _.debounce(async function (value) {
    const route = {
      name: 'blogs'
    }

    if(this.search !== '') {
      route.query = {
        search: this.search
      }
    }

    console.log('search: ' + this.search)
    this.$router.push(route)
  }, 700),

  '$route.query.search': {
    immediate: true,
    async handler (value) {
      this.blogs = []
      this.results = []
      this.loading = true
      this.results = (await BlogsService.index(value)).data
      this.appendResults()

      this.results.forEach(blog => {
        if (this.category.length > 0) {
          // console.log(this.category.indexOf(blog.category))
          if(this.category.indexOf(blog.category) === -1) {
            this.category.push(blog.category)
          }
        } else {

          this.category.push(blog.category)
        }
      })
      this.loading = false
      this.search = value
      // console.log(this.category)
    }
  }
}
```

```
},
```

ซึ่งจริงๆ แล้วไม่มีอะไรมากเลยนอกจาก ดึง category ใน blogs data ของเรา
คันว่ามีใน category ของเราเมื่ย ถ้าไม่มีให้ทำการเพิ่มเข้าไป เราจะได้ category
ทั้งหมด ที่ไม่ซ้ำกันนั้นเองครับ

จากนั้นเอามันมาแสดงใน blog header ของเราครับ โดยเอาไว้ต่อจากปุ่ม create
blog ของเรา

```
<ul class="categories">
  <li v-for="cate in category" v-bind:key="cate.index"><a v-
on:click.prevent="setCategory(cate)" href="#">{{ cate }}</a></li>
  <li class="clear" ><a v-on:click.prevent="setCategory('')"
href="#">Clear</a></li>
</ul>
<div class="clearfix"></div>
```

ใส่ style ให้มันซะหน่อย จะได้ไม่ขี้เหล่มากครับ

```
.categories {
  padding: 0;
  list-style: none;
  float: left;
}

.categories li {
  float: left;
  padding: 2px;
}

.categories li a {
  padding: 5px 10px 5px 10px;
  background:paleturquoise;
  color: black;
  text-decoration: none;
}

.categories li.clear a {
  background: tomato;
  color: white
}
```

จากนั้น ผูกเขียน function หรือ method setCategory ขึ้นมา เพื่อส่งไปหา
search query นั้นเองครับ

```

setCategory (keyword) {
  if(keyword === ' '){
    this.search = ''
    console.log('null')
  } else {
    this.search = keyword
  }
},

```

ต้องบอกร่วมกับเวลาเขียนตามเค้าเนื่องจากจริง ๆ ครับ จากนั้นผมป้องกัน การแสดงผล ในส่วนของการ โหลดต่าง ๆ เพื่อไม่หน้าจอมัน flick หรือ กระพริบเว็บ ๆ ไม่น่าดู ครับ

โดยเพิ่ม

```
loading: false,
```

ไว้ใน data function search ของเรา จะทำการจัดการกำหนดค่าให้ ตัวแปรตัวนี้ เราไปทำการอ่านค่าที่ template โดยแก้ไข code ดังนี้ครับ

```

<div v-if="blogs.length === 0 && loading === false" class="empty-blog">
  *** ไม่มีข้อมูล ***
</div>
<div id="blog-list-bottom">
  <div class="blog-load-finished" v-if="blogs.length === results.length && results.length > 0 && loading === false">โหลดข้อมูลครบแล้ว</div>
</div>

```

เมื่อทำการทดสอบ โดยการ เลือก category ของเราจะสังเกตว่าจะมีการ flicker ของหน้าจอหรือหน้าจอกกระพริบ เพราะว่า เมื่อเรารีบต้นทำงานโปรแกรมที่เราเขียนจะเริ่มจากข้อมูลว่างเปล่า ทำให้แบบ blog-list-bottom ทำงาน เราแก้ได้โดย การแก้ไข style เป็นดังนี้ครับ

```

#blog-list-bottom{
  padding-top:4px;
}

.blog-load-finished{
  padding:4px;
  text-align: center;
  background: seagreen;
  color:white;
}

```

```
}
```

จากนั้นทำการเพิ่ม transition-group ไป colum ส่วน blog list ของเราไว้ จะได้แสดงผลได้ดีไม่วุบวับครับ

```
<transition-group name="fade">
  <div v-for="blog in blogs" v-bind:key="blog.id" class="blog-list">
    <!-- <p>id: {{ blog.id }}</p> -->
    <div class="blog-pic">
      <!-- <transition name="fade"> -->
      <div class="thumbnail-pic" v-if="blog.thumbnail != 'null'">
        
      </div>
      <!-- </transition> -->
    </div>
    <h3>{{ blog.title }}</h3>
    <div v-html="blog.content.slice(0,200) + '...'></div>
    <div class="blog-info">
      <p><strong>Category:</strong> {{ blog.category }}</p>
      <p><strong>Create:</strong> {{ blog.createdAt }}</p>
      <!-- <p>status: {{ blog.status }}</p> -->
      <p>
        <button v-on:click="navigateTo('/blog/' + blog.id)">อ่าน</button>
        <button v-on:click="navigateTo('/blog/edit/' + blog.id)">แก้ไข</button>
        <button v-on:click="deleteBlog(blog)">ลบข้อมูล</button>
      </p>
    </div>
    <div class="clearfix"></div>
  </div>
</transition-group>
```

ทดสอบ กด link category ของเราจะแสดงผลตาม category ได้อย่างดีเลยครับ แต่จำนวนของ blog ที่แสดงของเรายังมีค่าที่ผิดอยู่ทำการแก้ไขดังนี้ครับ

```
<strong> จำนวน blog: </strong> {{results.length}}
```

เพราะเราโหลด blog ทั้งหมดของเราไว้ที่ results[] ก่อนแล้วจึงแยก page display ไปที่ blogs[] ดังนั้นจึงต้องทำการเปลี่ยนตัวแปรในการแสดงผล

← → ⌂ i localhost:8080/blogs?search=v... ☆ ⌂ ⌂ ⌂ | ⌂ :

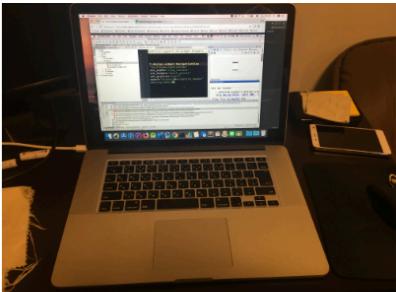
Blogs Users Comments Login Logout

ส่วนจัดการบล็อก

vuejs

create blog จำนวน blog: 3

css vuejs html nodejs Clear



Blog 8 ของเรา

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown pr...

Category: vuejs

Create: 2019-03-13T04:46:20.586Z

ดู blog แก้ไข blog ลบข้อมูล



Blog 6 ของเรา

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown pr...

ตอนนี้เว็บไซต์ blog ของเราเริ่มเป็นรูปเป็นร่างแล้วนะครับ หน้าตาดีพอใช้ การแสดงผลโดยรวมถือว่าใช้ได้ featured หลักๆ ก็ได้ครบเกือบหมดแล้ว ในบทต่อ ๆ จะเป็นเรื่องของการการแสดงผล และหน้าตาที่ดีขึ้นนั่นเองครับ

บทที่ 15 Bootstrap และการใช้งาน

Bootstrap คือ CSS Framework ที่นิยมมาก ใช้งานได้ดีมีประสิทธิภาพ ปกติถ้าเราใช้ Vue มักจะรีบไปที่ Vue Bootstrap แต่ผมกลับรู้สึกว่าผมเขียนด้วย Bootstrap ธรรมดานั้น ยืดหยุ่นแล้วปรับได้ง่ายกว่า Vue Bootstrap มาก ไม่ต้องไปปรับตามเงื่อนไข Lib ถึงแม้ ตัว Vue Bootstrap จะมีส่วนที่สำเร็จรูปมาให้เยอะ แต่ผมก็ไม่แคร์ เพราะเขียนเองก็ได้ และจัดการได้อย่างอิสระมากกว่า !!! แต่งานเร่ง ๆ ผมก็ชอบใช้อยู่บ้างนะครับ แต่ในหนังสือเล่มนี้เราจะใช้ Bootstrap กันครับ

Bootstrap เวลาผมใช้ Vue ผมใช้ผ่าน CDN เลยครับ เพราะการติดตั้งและงานผ่าน webpack นั้นดูวุ่นวายเกินไปสำหรับผม ทำการเปิด src/index.html ใน Client ของเราซึ่นมาครับ ทำการเพิ่ม CDN กันได้ครับ

```
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" integrity="sha384-BVYiiSIFeK1dGmJRAkycuHAHRg320mUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u" crossorigin="anonymous">
```

เอาใน header ของ index.html ของเรานะครับ จากนั้นใส่ส่วนของ script สำหรับ Bootstrap แต่ Bootstrap script นั้นใช้งาน jQuery ด้วย เราเลยต้องติดตั้ง Jquery ให้มันก่อนครับ

```
<script src="http://code.jquery.com/jquery-3.3.1.min.js" integrity="sha256-FgpCb/KJQlLNf0u91ta32o/NMZxltwRo8QtmkMRdAu8=" crossorigin="anonymous"></script>
```

```
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js" integrity="sha384-Tc5IQib027qvyjSMfHj0MaLkfufWVxZxUPnCJA7l2mCWNIpG9mGCD8wGNICPD7Txa" crossorigin="anonymous"></script>
```

เอาไว้ก่อนจบ tag </body> ของเรานะครับ

จากนั้นเปิด Web Browser ของเรามาไปที่ <http://localhost:8080/blogs> หน้าตาของเราจะเปลี่ยนไปทั้ง font ทั้ง button และ อื่น ๆ หากที่เราทำไว้ มันเพี้ยนก็ไม่ต้อง

ตกใจนะครับ ทุกอย่างเดียวเราปรับได้ ตอนนี้เราราใช้งาน Bootstrap ได้แล้วนั่นเอง
ครับ

Blogs Users Comments Login Logout

ส่วนจัดการบล็อก

Search

create blog จำนวน blog: 10

css vuejs html nodejs Clear



Blog 10 ของเรา

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown pr...

Category: css

Create: 2019-03-13T04:47:29.218Z

ดู blog แก้ไข blog ลบข้อมูล

Blog 9 ของเรา

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown pr...

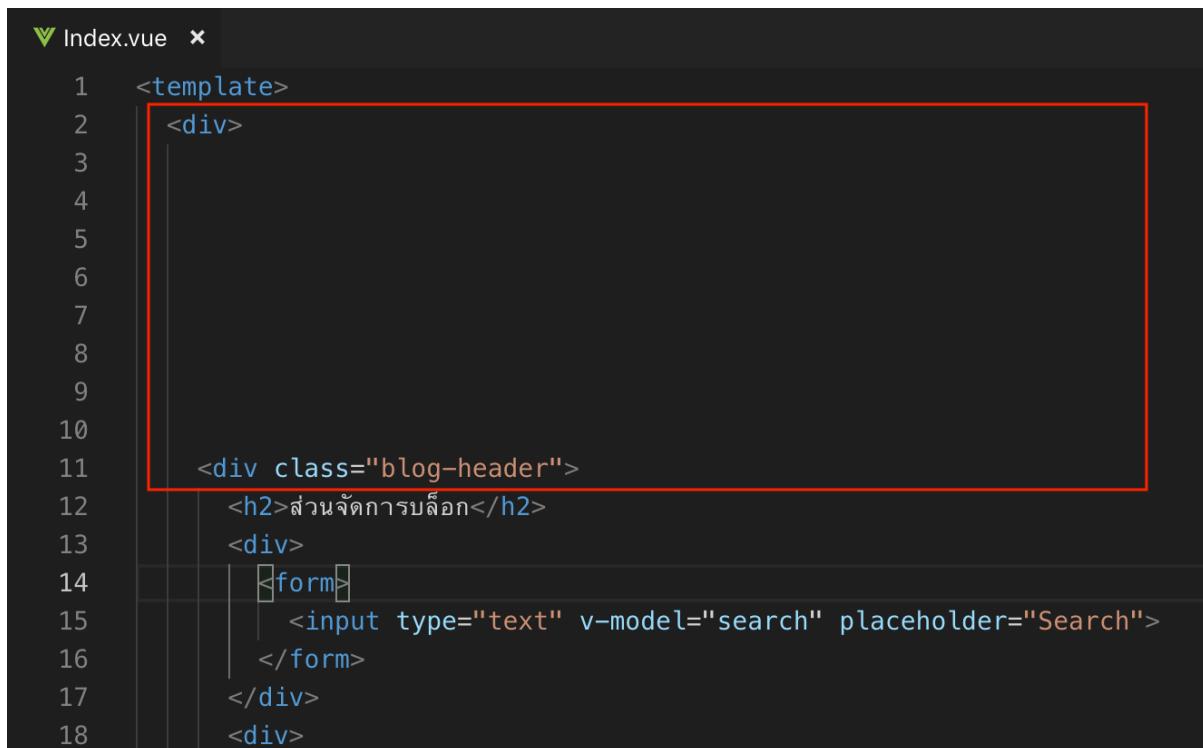
Category: css

Create: 2019-03-13T04:46:47.030Z

องค์ประกอบ และ Layout ของ Bootstrap

อย่างแรกเลยก่อนที่เราจะใช้งาน หลังจากติดตั้ง Bootstrap เรียบร้อยแล้ว เรา마다
องค์ประกอบและ วิธีใช้กันก่อนเลยครับ โดยผมใช้
src/components/Blogs/Index.vue นี่ล่ะครับ ทดสอบ เพราะมีเกี่ยวเสียเวลาไป
สร้าง components และ route ใหม่

ก่อนจะสร้างส่วนทดสอบ ผมขอเคลียร์พื้นที่ไฟล์ของผมก่อนดังนี้ครับ จะได้ไม่
ไปมีวักกับ ส่วนที่ work แล้วครับ



```
<template>
<div>
<div class="blog-header">
    <h2>ส่วนจัดการบล็อก</h2>
    <div>
        <form>
            <input type="text" v-model="search" placeholder="Search">
        </form>
    </div>
</div>
```

```
        }
    </script>
<style scoped>

.empty-blog {
    width: 100%;
    text-align: center;
    padding: 10px;
    background: #darksalmon;
    color: white;
}
```

จากนั้นเรามาเขียน code ทดสอบกันดังนี้ครับ

```
<template>
<div>
    <div class="container">
        <div class="row">
            <div class="col-md-4 col1">
                column #1
            </div>
            <div class="col-md-4 col2">
                column #2
            </div>
            <div class="col-md-4 col3">
                column #3
            </div>
        </div>
    </div>
```

แล้วมาทำการเพิ่ม Class style ให้กับ html template ของผลดังนี้

```
<style scoped>

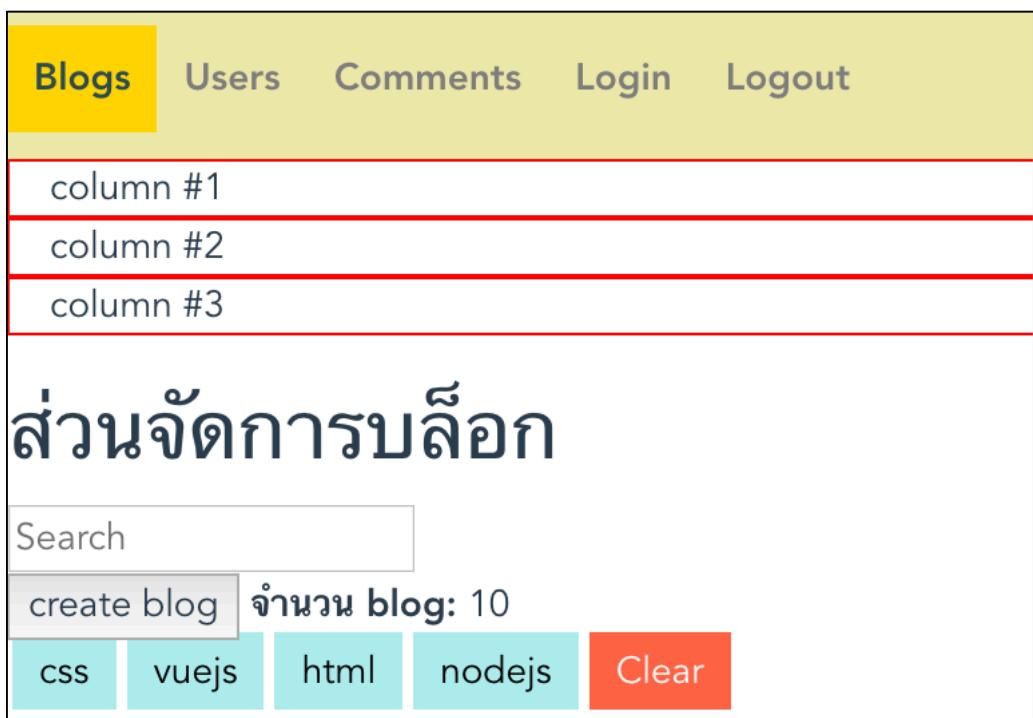
.col1, .col2, .col3 {
    border: solid 1px red;
```

}

เปิดหน้าจอแบบเก็บเต็มจอ จะได้ผลเป็นดังนี้ครับ



พอเราลองเปลี่ยนขนาดดู Bootstrap จะจัดเรียงให้เราใหม่ โดยอามาต่อ กัน



โดยการเริ่มต้นเราจะใช้ class container

```
<div class="container">
```

โดย .container นี้จะเป็นตัวปรับขนาดการแสดงผลของเราให้มีขนาดสูงสุด 1170px และจัดวางไว้ตรงกลางให้เรา เนื่องจากมีการเปลี่ยนแปลงขนาด ก็จะจัดเรียงให้เราอัตโนมัติทันที โดยกำหนดขนาดจาก .col-md-x โดยที่ x หมายถึงจำนวน

column ที่ span ของเรา โดย column grid ที่กำหนดไว้ คือ 12 columns ครับ ผม
นีงเลยครับ อาจารย์ท่านอธิบายมาอย่างนี้

มาแนวผมกันครับ คือ col ที่จะแสดงได้ก็ว่างเต็มหน้าพอดีโดยไม่ไปขึ้นบรรทัดอื่น
คือ 12 ชิ้น ถ้ากำหนด .col-md-1 ได้ 12 ชิ้น แต่ถ้า กำหนด .col-md-3 จะได้ 4
ชิ้นหรือ 4 columns ถ้ากำหนด col-md-4 จะได้สามชิ้นที่พ่อเรียงกันโดยไม่โดดไป
ขึ้น บรรทัดใหม่นั่นเองครับ

md คือ medium display คือสำหรับหน้าจอขนาดกลาง ให้แสดงขนาดนี้ ผมมักจะ^{ใช้ตัวนี้เป็นหลัก ส่วนตัวอื่น ๆ เช่น col-xs-x ใช้กับ หน้าจอขนาดเล็ก}

เช่นผมเขียนชื่อนักไว้ อย่างนี้

```
<div class="col-md-4 col-xs-6 col1">
```

ถ้าเป็นหน้าใจ 734px-992px มันจะเป็น 3 ชิ้นเต็มหน้าจอ แต่พอขนาดหน้าจอเป็น^{ขนาดเล็กกว่า 734px} มันจะกลายเป็น 2 ชิ้นกว้างพอดีหน้าจอ เพราะ ถ้า 3 ชิ้นอาจ^{เล็กไปถ้าไปอยู่บนจอเล็ก} ส่วนหน้าจอใหญ่ เราก็ว่ากันไปครับ คร่าว ๆ จะทำงาน^{เป็นอย่างนี้ครับ}

ไปอ่านเพิ่มเติมและลองเล่นอะไรเพิ่มเติมได้ที่นี่ครับ

<https://getbootstrap.com/docs/4.3/layout/overview/>

นอกจากความสามารถเรื่อง grid layout หรือเจ้า .col-md-x ของมันแล้ว Bootstrap ก็จะมี css build in มาให้ พวกลีตัวอักษร รูปแบบตัวอักษร default margin และ padding มาให้ ซึ่งผมว่า เหมาะสมสมส่วนมากตีครับ

แต่ที่ผมใช้ประจำก็คือ navbar หรือ form control ต่าง ๆ ซึ่งเราเรียกว่า Bootstrap Component ครับ การใช้งานสามารถ ดูได้จาก Document เลยครับ สามารถ copy code ที่เค้ามีให้ มาทำการเปลี่ยนได้เลยครับ

โดย Bootstrap components ต่าง ๆ สามารถดูได้จากตรงนี้ครับ

<https://getbootstrap.com/docs/4.3/components/alerts/>

เริ่มต้นการใช้ navbar กับ Back Office

เราจะมาทำ navbar แบบ responsive โดยใช้ bootstrap กันครับ ผมเปิดไปที่ components ของ bootstrap ที่นี่ครับ <https://getbootstrap.com/docs/4.3/components/navbar/> และ copy code ที่มีอยู่มาแก้ครับ โดย ผมเปิดไฟล์ src/components/Header.vue ขึ้นมาเพื่อทำการแก้ code ดังนี้

ผมจะลง code ในส่วนทดสอบ Bootstrap ที่เราเขียนขึ้นทิ้งไว้บนครับ จากนั้นผมจะเริ่มเขียนใหม่ ผมมองว่าสอน แนะนำแนวทางเพียงเล็กน้อย แล้วทำตัวอย่างให้ดูแล้วเขียน จะเข้าใจมากกว่าเรียนอย่างเดียวแล้วไม่ได้ลงมือทำ

เรามาเริ่มต้นจัดการ navbar ของเรากันเลยครับ วิธี copy มาแบบไม่ให้โปรแกรมของเรานั้นและ ผมเลือก navbar อันเก่าของผมลงมาแล้ว เปะ navbar อันใหม่ของผมไว้ข้างบน แล้วค่อยไล่ແປะ เฉพาะจุดที่เหมือนกันจนครบ โดย code navbar ใหม่ของผมมี ดังนี้ครับ copy ไปวางแบบรูป แล้วเด่าว่ายไปเชื่อมกันครับ

```
<!-- new navbar -->
<div class="navbar navbar-inverse navbar-fixed-top">
  <div class="container">
    <div class="navbar-header">
      <a class="navbar-brand" href="#" v-on:click.prevent="navigateTo('/dashboard')">
        
      </a>
      <button class="navbar-toggle collapsed" data-toggle="collapse" data-target="#navcol-1">
        <span class="sr-only">Toggle navigation</span>
        <span class="icon-bar"></span><span class="icon-bar"></span>
        <span class="icon-bar"></span>
      </button>
    </div>
    <div class="collapse navbar-collapse" id="navcol-1">
      <ul class="nav navbar-nav navbar-right">
        <li role="presentation"><a href="#">Blogs</a></li>
        <li role="presentation"><a href="#">Users</a></li>
        <li role="presentation"><a href="#">Comments</a></li>
        <li role="presentation"><a href="#">Login</a></li>
        <li role="presentation"><a href="#">Logout</a></li>
```

```

        </ul>
    </div>
</div>

<br />
<br />
<br />

<!-- old navbar -->
<div class="nv-navbar">
    <ul class="nav">
        <li><router-link :to="{name: 'blogs'}" >Blogs</router-link></li>
        <li><router-link :to="{name: 'users'}" >Users</router-link></li>
        <li><router-link      :to="{name:      'comments'}"      >Comments</router-
link></li>
        <li><router-link :to="{name: 'login'}" >Login</router-link></li>
        <li><a href="#" v-on:click="logout">Logout</a></li>
    </ul>
    <div class="clearfix"></div>
</div>

```

จากนั้น เพิ่ม style ไว้จัดการ logo และการ active ของ navbar ของเราสักหน่อย โดยเพิ่ม code ในส่วน style ดังนี้ครับ

```

.navbar-brand > img {
    width: 36px;
    padding: 12px 0;
    margin-top: -20px;
}
a.router-link-active{
    color:yellowgreen !important;
}

```

logo ของเราจะได้ไม่ระเบิดครับ และ navbar ของเราจะเปลี่ยนสีเมื่อ active ด้วย เช่นกัน

จากนั้นเปิด Web Browser ของเราระบุ URL <http://localhost:8080/blogs> ครับ หรือหน้าอื่น ๆ navbar ของเราจะติดไปด้วย เราเรียกใช้ navbar component ของเราที่ App.vue นั่นเอง

ตอนนี้ มีจะมี navbar ของผมทั้งหมดสองตัวดังนี้นะครับ

The screenshot shows a web application interface for managing blogs. At the top, there is a dark header bar with a logo 'V' on the left and navigation links: Blogs, Users, Comments, Login, and Logout. Below the header is a yellow navigation bar with tabs: Blogs (selected), Users, Comments, Login, and Logout. The main content area has a title 'ส่วนจัดการบล็อก' (Blog Management). It includes a search bar, a 'create blog' button, and a message 'จำนวน blog: 10'. Below this are four blue buttons labeled 'css', 'vuejs', 'html', and 'nodejs', followed by a red 'Clear' button. A thumbnail image of a laptop and a Starbucks cup is displayed next to the heading 'Blog 10 ของเรา'. The blog content summary reads: 'Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown pr...'. It includes a 'Category: css' label, a 'Create: 2019-03-13T04:47:29.218Z' timestamp, and three small buttons at the bottom: 'ดู blog', 'แก้ไข blog', and 'ลบข้อมูล'.

ทดลองเปลี่ยนขนาดของ Web Browser ของเราดูนะครับ เป็นไปครับ responsive navbar ของเราทำงานทันที เป็นไปครับ bootstrap ช่วยให้ชีวิตเรา่ง่ายขึ้นจริง ๆ ครับ



Blogs

Users

Comments

Login

Logout



Blog 10 ของเรา

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown pr...

Category: css

Create: 2019-03-13T04:47:29.218Z

มาถึงตอนนี้ใครอยากรอเปลี่ยนสี ก็สามารถเปิด dev tool ของแต่ละ browser หรือ กดคลิ๊กขวาแล้วเปิดไปที่ inspector เมนู แล้วจิ้มไปที่ element ต่างของเรา เราจะเป็นชื่อ class ต่าง ๆ เหล่านั้นคือ style ของเราเอง เราสามารถ copy ชื่อ class เหล่านั้น มาแปะใน style ใน vue component ของเราแล้วทำการแก้ไขได้เลยครับ

ใครอยากรู้ว่าเราทำอะไรกันบ้าง เล่นดูนะครับ ส่วนนี้จะใช้ชื่อย่างนี้ไปก่อน มาถึงตอนนี้เรามาทำการตั้งค่า navbar ให้เป็นแบบที่เราต้องการ เช่น ทำให้เป็นแบบ collapse หรือเปลี่ยนสีของ navbar เป็นสีอื่น ฯลฯ ทั้งหมดนี้สามารถทำได้โดยการแก้ไข code ดังนี้ครับ

```
<template>
<div>
    <!-- new navbar -->
    <div class="navbar navbar-inverse navbar-fixed-top">
        <div class="container">
            <div class="navbar-header">
                <a class="navbar-brand" href="#" v-on:click.prevent="navigateTo('/dashboard')">
                    
                </a>
                <button class="navbar-toggle collapsed" data-toggle="collapse" data-target="#navcol-1">
                    <span class="sr-only">Toggle navigation</span>
                    <span class="icon-bar"></span><span class="icon-bar"></span>
                </button>
            </div>
        </div>
    </div>
```

```

        </div>
        <div class="collapse navbar-collapse" id="navcol-1">
            <ul class="nav navbar-nav navbar-right">
                <li role="presentation"><router-link :to="{name: 'blogs'}">Blogs</router-link></li>
                <li role="presentation"><router-link :to="{name: 'users'}">Users</router-link></li>
                    <li role="presentation"><router-link :to="{name: 'comments'}">Comments</router-link></li>
                    <li role="presentation"><router-link :to="{name: 'login'}">Login</router-link></li>
                    <li role="presentation"><a href="#" v-on:click="logout">Logout</a></li>
            </ul>
        </div>
    </div>
</div>
</template>

```

```

        <span class="icon-bar"></span>
    </button>
</div>
<div class="collapse navbar-collapse" id="navcol-1">
    <ul class="nav navbar-nav navbar-right">
        <li role="presentation"><router-link :to="{name: 'blogs'}">Blogs</router-link></li>
        <li role="presentation"><a href="#">Users</a></li>
        <li role="presentation"><a href="#">Comments</a></li>
        <li role="presentation"><a href="#">Login</a></li>
        <li role="presentation"><a href="#">Logout</a></li>
    </ul>
</div>
</div>
<br />
<br />
<br />

<!-- old navbar -->
<div class="nv-navbar">
    <ul class="nav">
        <li><router-link :to="{name: 'blogs'}">Blogs</router-link></li>
        <li><router-link :to="{name: 'users'}">Users</router-link></li>
        <li><router-link :to="{name: 'comments'}">Comments</router-link></li>
        <li><router-link :to="{name: 'login'}">Login</router-link></li>
        <li><a href="#" v-on:click="logout">Logout</a></li>
    </ul>
</div>

```

จากนั้นทำการลบ style ที่เกี่ยวกับ navbar ที่เราเคยเขียนออกทั้งหมด เวลาที่เราเขียนจัดการ navbar จะได้ไม่สับสนครับ ยกเว้น

```
.navbar-brand > img {
```

ที่เราเพิ่งเพิ่มไปครับ

ทำการลบ

```
.clearfix {  
  clear: left;  
}
```

ออกด้วยนะครับ เพราะ bootstrap มี class นี้ให้ใช้งานอยู่แล้วครับ จะได้ไม่โหลดซ้ำกันครับ ดังนี้ style ของผมใน component นี้ จะเหลือแค่นี้ครับ

```
<style scoped>  
.navbar-brand > img {  
  width: 36px;  
  padding: 12px 0;  
  margin-top: -20px;  
}  
a.router-link-active{  
  color:yellowgreen !important;  
}  
</style>
```

ตอนนี้ ทั้ง navbar และ layout ต่าง ของเราได้ใช้งาน bootstrap เพื่อทำให้ รองรับ หลาย ๆ display size หรือ reponsive designed เล็ก ๆ แล้วนะครับ ทดสอบย่อขยาย web browser ของเราจะเห็นว่า layout ของเราจะปรับเปลี่ยนไปอัตโนมัติ ครับ

เริ่มต้นใช้งาน Bootstrap Component

หลังจากที่เริ่มใช้ Bootstrap เพื่อจัดการ CSS หรือ Style หรือ รูปร่างหน้าตาของ Webblog ของเราแล้ว เราสามารถใช้งาน bootstrap component ได้ทั้งหมดเลย ซึ่งรายละเอียดมีเยอะมาก ของจะไม่อธิบายมากนัก เพราะอยากรู้ตัวไหน ก็สามารถ copy มาแปะ และทดลองแก้ได้เลยครับ ผมจะเก็บงานส่วนของ src/components/Blogs/Index.vue หรือหน้า blogs index ของผมเป็นดังนี้ครับ

```

<p>
  <button class="btn btn-sm btn-info" v-on:click="navigateTo('/blog/' + blog.id)">View Blog</button>
  <button class="btn btn-sm btn-warning" v-on:click="navigateTo('/blog/edit/' + blog.id)">Edit blog</button>
  <button class="btn btn-sm btn-danger" v-on:click="deleteBlog(blog)">Delete</button>
</p>

```

โดยส่วนที่ผนวก คือ เพิ่ม class ของปุ่มของผู้ดูแล รายละเอียดเกี่ยวกับ button components สามารถได้ที่นี่ครับ

<https://getbootstrap.com/docs/4.3/components/buttons/>

ซึ่งผู้ดูแลสามารถดาวน์โหลด code ทั้งหมดเลยที่เดียว ผู้อ่านสามารถได้ดูจาก git ของหนังสือ ในบทนี้ได้เลยนะครับ

```

<template>
  <div class="container">
    <div class="blog-header">
      <h2>ส่วนจัดการบล็อก</h2>
      <div>
        <form class="form-inline form-search">
          <div class="form-group">
            <div class="input-group">
              <input type="text" v-model="search" class="form-control" id="exampleInputAmount" placeholder="Search">
              <div class="input-group-addon"><i class="fas fa-search"></i></div>
            </div>
          </div>
        </form>
      </div>
      <div class="create-blog">
        <button class="btn btn-success btn-sm" v-on:click="navigateTo('/blog/create')">Create blog</button>
        <strong> จำนวน blog: </strong> {{results.length}}
      </div>
      <ul class="categories">
        <li v-for="cate in category" v-bind:key="cate.index"><a v-on:click.prevent="setCategory(cate)" href="#">{{ cate }}</a></li>
      </ul>
    </div>
  </div>

```

```

        <li class="clear" ><a v-on:click.prevent="setCategory(' ')" href="#">Clear</a></li>
    </ul>
    <div class="clearfix"></div>
</div>
<transition-group name="fade">
    <div v-for="blog in blogs" v-bind:key="blog.id" class="blog-list">
        <!-- <p>id: {{ blog.id }}</p> -->
        <div class="blog-pic">
            <!-- <transition name="fade"> -->
            <div class="thumbnail-pic" v-if="blog.thumbnail != 'null'">
                
            </div>
            <!-- </transition> -->
        </div>
        <h3>{{ blog.title }}</h3>
        <div v-html="blog.content.slice(0,200) + '...'></div>
        <div class="blog-info">
            <p><strong>Category:</strong> {{ blog.category }}</p>
            <p><strong>Create:</strong> {{ blog.createdAt }}</p>
            <!-- <p>status: {{ blog.status }}</p> -->
            <p>
                <button class="btn btn-sm btn-info" v-on:click="navigateTo('/blog/' + blog.id)">View Blog</button>
                <button class="btn btn-sm btn-warning" v-on:click="navigateTo('/blog/edit/' + blog.id)">Edit blog</button>
                <button class="btn btn-sm btn-danger" v-on:click="deleteBlog(blog)">Delete</button>
            </p>
        </div>
        <div class="clearfix"></div>
    </div>
</transition-group>
<div v-if="blogs.length === 0 && loading === false" class="empty-blog">
    *** မျှမှန်ခွဲမှတ ***
```

```

</div>
<div id="blog-list-bottom">
    <div class="blog-load-finished" v-if="blogs.length === results.length && results.length > 0">ໄေဆင်ခွဲမှတရပါ။</div>
    </div>
</div>
```

```

</template>
<script>
```

```
import BlogsService from '@/services/BlogsService'
import _ from 'lodash'
import ScrollMonitor from 'scrollMonitor'

let LOAD_NUM = 3
let pageWatcher

export default {
  watch: {
    search: _.debounce(async function (value) {
      const route = {
        name: 'blogs'
      }

      if(this.search !== '') {
        route.query = {
          search: this.search
        }
      }

      console.log('search: ' + this.search)
      this.$router.push(route)
    }, 700),

    '$route.query.search': {
      immediate: true,
      async handler (value) {
        this.blogs = []
        this.results = []
        this.loading = true
        this.results = (await BlogsService.index(value)).data
        this.appendResults()

        this.results.forEach(blog => {
          if (this.category.length > 0) {
            // console.log(this.category.indexOf(blog.category))
            if(this.category.indexOf(blog.category) === -1) {
              this.category.push(blog.category)
            }
          } else {

            this.category.push(blog.category)
          }
        })
      }
    }
  }
}
```

```

        })
        this.loading = false
        this.search = value
        // console.log(this.category)
    }
}
},
data () {
    return {
        blogs: [],
        BASE_URL: "http://localhost:8081/assets/uploads/",
        search: '',
        results: [],
        category: [],
        loading: false,
    }
},
// async created () {
//     this.blogs = (await BlogsService.index()).data
// },
methods: {
    wait(ms) {
        return x => {
            return new Promise(resolve => setTimeout(() => resolve(x), ms));
        };
    },
    appendResults: function () {
        if (this.blogs.length < this.results.length) {
            let toAppend = this.results.slice(
                this.blogs.length,
                LOAD_NUM + this.blogs.length
            )
            this.blogs = this.blogs.concat(toAppend)
        }
    },
    navigateTo (route) {
        this.$router.push(route)
    },
    async deleteBlog (blog) {
        try {
            await BlogsService.delete(blog)
            this.refreshData()
        } catch (err) {
            console.log(err)
        }
    }
}

```

```

        }
    },
    async refreshData() {
        this.blogs = (await BlogsService.index()).data
    },
    setCategory (keyword) {
        if(keyword === ' '){
            this.search = ''
            console.log('null')
        } else {
            this.search = keyword
        }
    },
},
updated () {
    let sens = document.querySelector('#blog-list-bottom')
    pageWatcher = ScrollMonitor.create(sens)
    pageWatcher.enterViewport(this.appendResults)
},
beforeUpdated () {
    if (pageWatcher) {
        pageWatcher.destroy()
        pageWatcher = null
    }
}
}
</script>
<style scoped>
.empty-blog {
    width: 100%;
    text-align: center;
    padding: 10px;
    background: darksalmon;
    color: white;
}

/* thumbnail */
.thumbnail-pic img{
    width: 200px;
    padding: 5px 10px 0px 0px;
}

.blog-info {
    float: left;

```

```
}

.blog-pic {
    float: left;
}

.clearfix {
    clear: both;
}

.blog-list {
    border:solid 1px #dfdfdf;
    margin-bottom: 10px;
    margin-left: auto;
    margin-right: auto;
    padding: 5px;
    box-shadow: 0 2px 4px 0 rgba(0,0,0,.1);
}

.blog-header {
    margin-left: auto;
    margin-right: auto;
}

#blog-list-bottom{
    padding-top:4px;
}

.blog-load-finished{
    padding:4px;
    text-align: center;
    background: seagreen;
    color:white;
}

.categories {
    margin-top: 10px;
    padding: 0;
    list-style: none;
    float: left;
}

.categories li {
    float: left;
```

```

padding: 2px;
}

.categories li a {
padding: 5px 10px 5px 10px;
background:paleturquoise;
color: black;
text-decoration: none;
}

.categories li.clear a {
background: tomato;
color: white
}

.create-blog {
margin-top: 10px;
}

</style>

```

โดย code ในส่วนนี้จะมีจุดที่นำเสนอใจคือ

```

<form class="form-inline form-search">
    <div class="form-group">
        <div class="input-group">
            <input type="text" v-model="search" class="form-control"
id="exampleInputAmount" placeholder="Search">
            <div class="input-group-addon"><i class="fas fa-
search"></i></div>
        </div>
    </div>
</form>

```

ซึ่งเป็นการใช้ form group เข้ามาจัดการ form search ของเรา นั่นเอง แต่จะเห็น tag

```
<i class="fas fa-search"></i>
```

หรือ icon search เพิ่มเข้ามา ไม่ต้องแปลงไปในนะครับ မันจะอธิบายในหัวข้อถัดไป
นะครับ ตอนนี้หน้าตา webblog ของเราจะหล่อขึ้นมาดังนี้ครับ แต่ยังเป็น
Responsive ด้วยครับ



Blogs Users Comments Login Logout

ส่วนจัดการบล็อก

Search



Create blog

จำนวน blog: 7

html

vuejs

nodejs

css

Clear



Blog 7 ของเรา

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown pr...

Category: html

Create: 2019-03-17T18:48:32.394Z

View Blog

Edit Blog

Delete



Blog 6 ของเรา

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown pr...

Category: vuejs

Create: 2019-03-17T18:46:53.343Z

View Blog

Edit Blog

Delete



Blog 5 ของเรา

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown pr...

Category: nodejs

Create: 2019-03-17T18:45:10.805Z

View Blog

Edit Blog

Delete



Blog 4 ของเรา

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown pr...

Category: css

Create: 2019-03-17T18:44:00.529Z

View Blog

Edit Blog

Delete



Blog 3 ของเรา

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown pr...

Category: nodejs

Create: 2019-03-17T16:27:22.875Z

View Blog

Edit Blog

Delete



Blog 2 ของเรา

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown pr...

Category: vuejs

Create: 2019-03-17T16:26:17.967Z

View Blog

Edit Blog

Delete

ส่วนจัดการบล็อก

Search 🔍

Create blog จำนวน blog: 7

html vuejs nodejs css Clear

Blog 7 ของเรา



Category: html
Create: 2019-03-17T18:48:32.394Z

View Blog Edit Blog Delete

Blog 6 ของเรา



Category: vuejs
Create: 2019-03-17T18:46:53.343Z

View Blog Edit Blog Delete

Blog 5 ของเรา



Category: nodejs
Create: 2019-03-17T18:45:10.805Z

View Blog Edit Blog Delete

Blog 4 ของเรา



Category: css
Create: 2019-03-17T18:44:00.529Z

View Blog Edit Blog Delete

Blog 3 ของเรา



Category: nodejs
Create: 2019-03-17T16:27:22.875Z

View Blog Edit Blog Delete

Blog 2 ของเรา



Category: vuejs
Create: 2019-03-17T16:26:17.967Z

View Blog Edit Blog Delete

Blog 1 ของเรา



Category: nodejs
Create: 2019-03-17T16:22:53.657Z

View Blog Edit Blog Delete

โพลเดอร์มูลคามแล้ว

ทำความรู้จัก Font Awesome

Font Awesom คือ icon framework หรือ library ให้เราเรียกใช้ เพื่อที่จะได้ใส่ icon ใน เว็บไซต์ของเราได้ โดยทำการติดตั้งผ่าน CDN ใน src/index.html ใน ส่วนของ client ของเราได้เลย ดังนี้

```
<link rel="stylesheet"  
      href="https://use.fontawesome.com/releases/v5.1.0/css/all.css"  
      integrity="sha384-  
      lKuwvrZot6UHsBSfcMv0kWwlCMgc0TaWr+30HWe3a4ltaBwTZhYTEggF5tJv8tb"  
      crossorigin="anonymous">
```

เมื่อเราเรียกใช้งาน สามารถเรียกผ่าน tag <i></i> หรือ icon ได้เลย อยากรู้ว่า icon มีอะไรให้เราใช้บ้างดูได้ที่นี่เลยครับ

<https://fontawesome.com/icons?d=gallery>

วิธีใช้ ก็จิ้มไปที่ icon ที่เราต้องการเลยครับ และก็ทำการ copy code ไปวางในส่วน ที่เราต้องการแสดงผล icon ของเรา

search

Solid Style (fas)

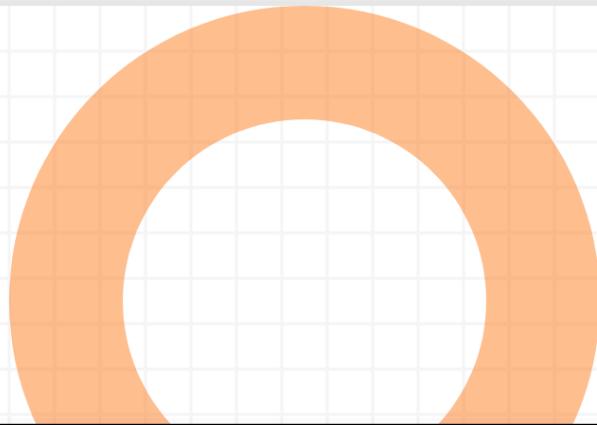


f002

<i class="fas fa-search"></i>



Interfaces • Updated: Version 5.0.0



ตอนนี้เรามีความรู้พอก็จะจัดการเรื่อง Theme และความสวยงามของเราได้แล้ว
เนื่องจาก เราได้ทำ Engine หรือส่วนหลัก ๆ ของระบบไว้หมดแล้ว กลับไปจัดการ
หน้าที่ผ่าน ๆ มาให้สวยงาม และ พังก์ชันการค้นหา กันเองนะครับ ถ้าใครทำไม่ได้
ให้เปิด git ในบทนี้ดูประกอบนะครับ ผมจะไม่อธิบายมากนัก เพราะ มันจะซ้ำ ๆ และ
หน้าเมื่อ และเป็นการเปิดโอกาสให้ได้ลองทำเองครับ

หน้าตาหลังรับแต่งแล้วในส่วนของ Blog ของผม

Blogs Users Comments Login Logout

ส่วนจัดการบล็อก

Search

จำนวน blog: 7

Blog 7 ของเรา

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown pr...
Category: html
Create: 2019-03-17T18:48:32.394Z

Blog 6 ของเรา

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown pr...
Category: vuejs
Create: 2019-03-17T18:46:53.343Z

Blog 5 ของเรา

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown pr...
Category: nodejs
Create: 2019-03-17T18:45:10.805Z

Blog 4 ของเรา

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown pr...
Category: css
Create: 2019-03-17T18:44:00.529Z

Blog 3 ของเรา

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown pr...
Category: nodejs
Create: 2019-03-17T16:27:22.875Z

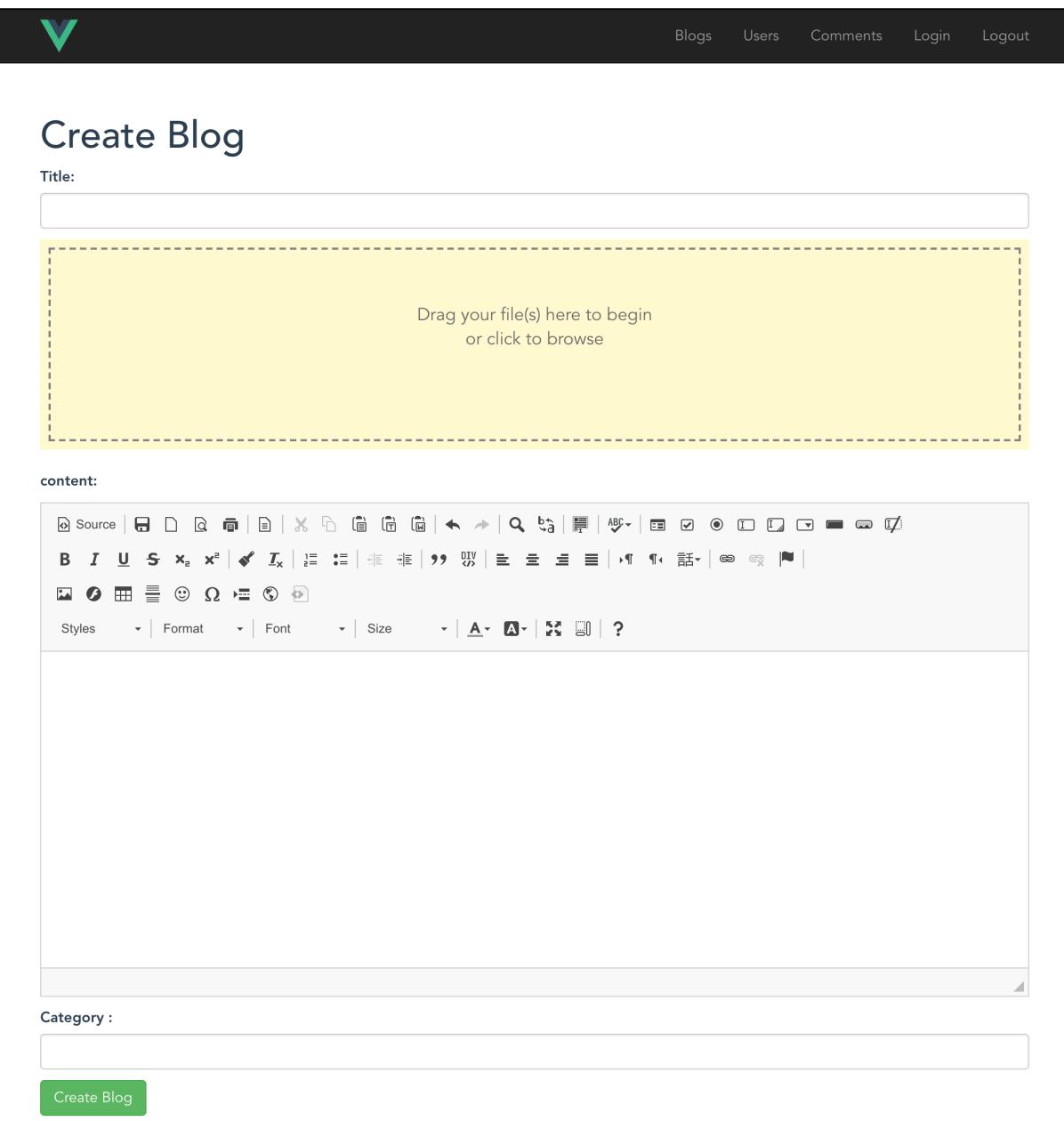
Blog 2 ของเรา

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown pr...
Category: vuejs
Create: 2019-03-17T16:26:17.967Z

Blog 1 ของเรา

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown pr...
Category: nodejs
Create: 2019-03-17T16:22:53.657Z

โหลดข้อมูลครบแล้ว



Edit Blog

Title:

Blog 7 ของเรา



Drag your file(s) here to begin
or click to browse



Thumbnail **Delete**

content:

The image shows the CKEditor toolbar, which includes a wide range of editing tools and icons. The top row contains icons for Source, Document, Find, Replace, Undo, Redo, Search, and various document formats like PDF and RTF. Below this is a row of icons for text selection, bold, italic, underline, superscript, subscript, and other styling options. The third row includes icons for lists, tables, and various symbols. At the bottom, there's a 'Styles' dropdown, a 'Format' dropdown, a 'Font' dropdown, a 'Size' dropdown, and a set of icons for bold, italic, underline, and other text effects.

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem
Insu



Category :

htm|

Update Blog



Blog 7 ของเรา

Category: : html

Status: saved

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.



Lorem Ipsum คือ เนื้อหาจำลองแบบเรียบๆ ที่ใช้กันในธุรกิจงานพิมพ์หรืองานเรียงพิมพ์ มันได้ถูกออกแบบมาเป็นเนื้อหาจำลองมาตรฐานของธุรกิจดังกล่าวมาตั้งแต่ศตวรรษที่ 16 เมื่อเครื่องพิมพ์ในเบนเน็ตติ่งหนึ่งนำร่างตัวพิมพ์มาลับสับต่ำให้แน่นตัวอักษรเพื่อกำหนดสีตัวอย่าง Lorem Ipsum อย่างคงกระพันมาให้เช่นเดียวทั่วโลก แต่เมื่อเวลาผ่านไป มนุษย์ก็สามารถเปลี่ยนแปลง มันได้รับความนิยมมากขึ้นในยุค ค.ศ. 1960 เมื่อแผ่น Letraset ทางจำหน่ายโดยมีข้อความบนบัน្តเป็น Lorem Ipsum และล่าสุดกว่านี้ คือเมื่อซอฟต์แวร์การกำสื่อสิ่งพิมพ์ (Desktop Publishing) อย่าง Aldus PageMaker ได้รวมเอา Lorem Ipsum เวอร์ชันต่างๆ เข้าไว้ในซอฟต์แวร์ด้วย Blog 7 ของเรา

[← Back..](#)

หน้าตาในส่วนของ User ของผมจะเป็นดังนี้ครับ

The screenshot shows a web application interface with a dark header bar containing a logo (a stylized 'V'), navigation links (Blogs, Users, Comments, Login, Logout), and a search bar. Below the header is a button labeled 'Create user' and a message indicating there are 5 users. The main content area displays four user entries, each in its own card:

- User 6:** id: 6
ชื่อ-นามสกุล: user3 - user3lastname
email: user3@gmail.com
สร้างเมื่อ: 2019-03-17T09:15:22.715Z
ระดับการใช้งาน: user
Buttons: View User (blue), Edit User (orange), Delete (red)
- User 7:** id: 7
ชื่อ-นามสกุล: korn - chayapon
email: korn.chayapon@gmail.com
สร้างเมื่อ: 2019-03-17T11:04:20.320Z
ระดับการใช้งาน: admin
Buttons: View User (blue), Edit User (orange), Delete (red)
- User 4:** id: 4
ชื่อ-นามสกุล: user1 - nakrub
email: user1@gmail.com
สร้างเมื่อ: 2019-03-17T07:35:05.241Z
ระดับการใช้งาน: admin
Buttons: View User (blue), Edit User (orange), Delete (red)
- User 3:** id: 3
ชื่อ-นามสกุล: user - nakrub
email: user@gmail.com
สร้างเมื่อ: 2019-03-17T07:34:13.716Z
ระดับการใช้งาน: admin
Buttons: View User (blue), Edit User (orange), Delete (red)



ແສດງຂໍ້ມູນຜູ້ໃຊ້ງານ

Name: user3

Lastname: user3

Email: user3@gmail.com

[← Back..](#)



ແກ້ໄຂຂໍ້ມູນຜູ້ໃຊ້ງານ

Name:

Lastname:

Email:

Password:

Type: ▲

✓ [Edit User](#) [Back](#)

The screenshot shows a user creation form titled "เพิ่มผู้ใช้งาน" (Add User). The form includes fields for Name, Lastname, Email, Password, and Type, with "User" selected. It also features a "Create User" button with a checkmark icon and a "Back" button.

Blogs Users Comments Login Logout

เพิ่มผู้ใช้งาน

Name:

Lastname:

Email:

Password:

Type:

หน้าตาของเรานี่ส่วน comment หลังจากทำการปรับแต่งใช้งานกับ Bootstrap
เรียบร้อยแล้ว

Search

จำนวน: 4 comments

Comment:
testing comment xxx
สร้างเมื่อ: 2019-03-17T12:15:28.179Z
[อุบล็อกที่ Comment](#) [ลบข้อมูล](#)

Comment:
testing comment, hello xxx
สร้างเมื่อ: 2019-03-17T12:15:39.913Z
[อุบล็อกที่ Comment](#) [ลบข้อมูล](#)

Comment:
power testing comment, hello xxx
สร้างเมื่อ: 2019-03-17T12:15:57.834Z
[อุบล็อกที่ Comment](#) [ลบข้อมูล](#)

Comment:
power ofxxxxxxxx testing comment, hello xxx
สร้างเมื่อ: 2019-03-17T12:16:10.319Z
[อุบล็อกที่ Comment](#) [ลบข้อมูล](#)

ໂທລດຂ້ອມສຸກຄະນະແລ້ວ

ในส่วนการเพิ่มเติมและปรับแต่ง code ของผม สิ่งที่ผมทำเพิ่มขึ้นมีดังนี้ ทำการเพิ่ม style ในส่วนที่เหลือทั้งหมด เพิ่ม icon จัดการเพิ่ม code ในส่วนค้นหา user และ comment ทั้งส่วน Front End และ Back End ทำการเพิ่ม ปุ่ม Back เพื่อกลับไปหน้าจัดการหลัก

นอกจากที่กล่าวมา ซึ่งจริง ๆ code ตัวอย่างมีเก็บหมดแล้ว รวมถึงผมคาดว่า ความรู้ที่สอนมาันนี้เพียงพอแล้วที่จะสามารถแก้ไข code ตัวเองตามผมได้ แต่ถึงอย่างไรสามารถ ดู code ของผมประกอบได้ครับ

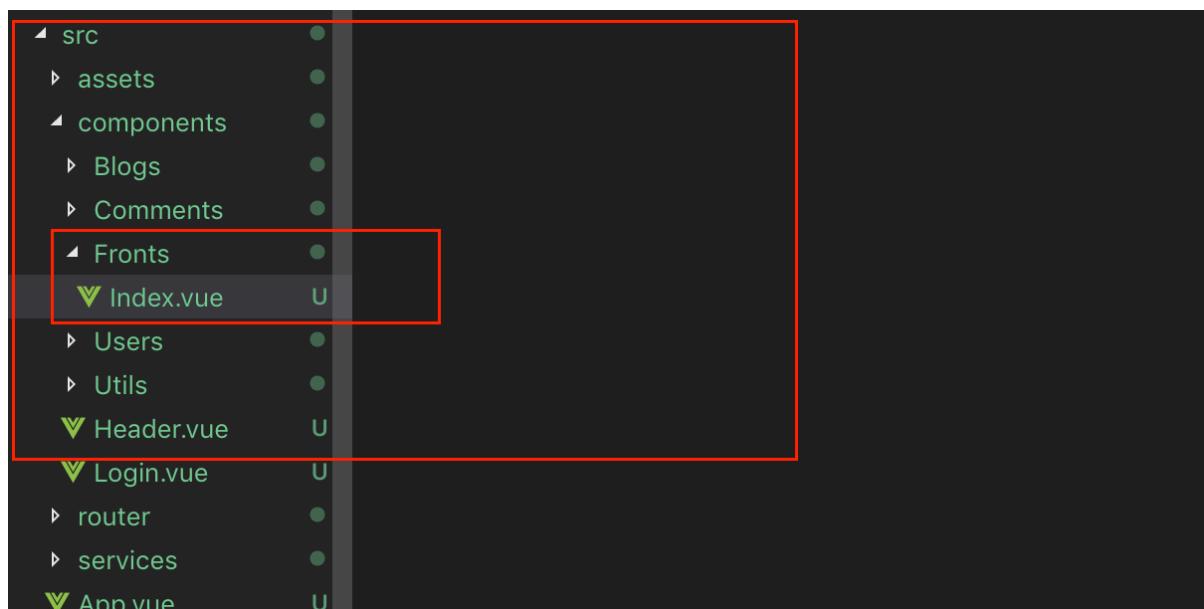
ในบทนี้เราเขียนโปรแกรม Front End ในส่วนของ Back Office เสร็จเกือบหมดแล้วนะครับ เหลือเพียงระบบ authen เพื่อ ปิดหลังบ้าน โดย ในหนังสือเล่มนี้ เป็นเพียงไกด์ไลน์ เพื่อให้สร้างระบบเว็บไซต์บล็อก เพื่อใช้งานได้อย่างไม่ยากนัก ในการจัดการเรื่องความปลอดภัย ผมเชื่อว่าท่านสามารถนำความรู้ที่ผ่านมาไป

พัฒนาต่อได้ไม่ยาก เช่น กำหนดครั้งในการ login หรือ เพิ่ม captcha เพื่อตรวจสอบ Bot หรือ Human

เริ่มต้นทำ Front Office หรือ หน้าบ้าน

เราเดินทางมาหลาย步 พอสมควร และส่วน Back Office ของเราก็เกือบจะสมบูรณ์แล้ว ต่อไปนี้เราจะมาเริ่มต้นทำ Front Office หรือ หน้าบ้านกัน ซึ่งต้องบอกว่าง่ายมาก เพราะเรามีทุกอย่างครบเกือบหมดแล้ว เรียกว่าสายแบะ นี่สายเลยครับ

สร้างโฟล์เดอร์ src/componts/Fronts ขึ้นมา และสร้าง Index.Vue ไว้ข้างในครับ โดย Fronts นี้จะมีไฟล์ทุกอย่างที่เกี่ยวกับ Front Office ของผู้คนของเรา โดยผมวางแผนว่า หน้าแรกจะเป็น blog ทั้งหมดในเบื้องต้นก่อน แบบหน้าเดียวจบเลย



โดยผมเอา code จาก src/Blogs/Index.vue มาแก้ไขดังนี้ครับ

```
<template>
  <div class="component-wrapper">
    <div class="hero">
      
      <h1>Webblot from nodejs + vuejs Ebook</h1>
      <p>By Gooddev.ME</p>
    </div>
```

```

<div class="clearfix"></div>
<div class="blog-header">
    <div>
        <form class="form-inline form-search">
            <span><strong> ຈຳນວນ blog: </strong> {{results.length}} </span>
            &nbsp;
            <div class="form-group">
                <div class="input-group">
                    <input type="text" v-model="search" class="form-control" id="exampleInputAmount" placeholder="Search">
                    <div class="input-group-addon"><i class="fas fa-search"></i></div>
                </div>
            </div>
        </form>
    </div>
    <ul class="categories">
        <li v-for="cate in category" v-bind:key="cate.index"><a v-on:click.prevent="setCategory(cate)" href="#">{{ cate }}</a></li>
        <li class="clear" ><a v-on:click.prevent="setCategory(' ')" href="#">Clear</a></li>
    </ul>
    <div class="clearfix"></div>
</div>
<transition-group name="fade">
    <div v-for="blog in blogs" v-bind:key="blog.id" class="blog-list">
        <!-- <p>id: {{ blog.id }}</p> -->
        <div class="blog-pic">
            <!-- <transition name="fade"> -->
            <div class="thumbnail-pic" v-if="blog.thumbnail != 'null'">
                
            </div>
            <!-- </transition> -->
        </div>
        <h3>{{ blog.title }}</h3>
        <div v-html="blog.content.slice(0,200) + '...'"></div>
        <div class="blog-info">
            <p><strong>Category:</strong> {{ blog.category }}</p>
            <p><strong>Create:</strong> {{ blog.createdAt }}</p>
            <!-- <p>status: {{ blog.status }}</p> -->
            <p>
                <button class="btn btn-sm btn-info" v-on:click="navigateTo('/blog/' + blog.id)"><i class="fab fa-readme"></i> View Blog</button>
            </p>
        </div>
    </div>
</transition-group>

```

```

        </p>
    </div>
    <div class="clearfix"></div>
</div>
</transition-group>
<div v-if="blogs.length === 0 && loading === false" class="empty-
blog">
    *** មានឯកសារ ***
</div>
<div id="blog-list-bottom">
    <div class="blog-load-finished" v-if="blogs.length ===
results.length && results.length > 0">ឯកសារបានដោះ</div>
    </div>
</div>
</template>
<script>

import BlogsService from '@/services/BlogsService'
import _ from 'lodash'
import ScrollMonitor from 'scrollMonitor'

let LOAD_NUM = 3
let pageWatcher

export default {
  watch: {
    search: _.debounce(async function (value) {
      const route = {
        name: 'front'
      }

      if(this.search !== '') {
        route.query = {
          search: this.search
        }
      }

      console.log('search: ' + this.search)
      this.$router.push(route)
    }, 700),
    '$route.query.search': {
      immediate: true,
      async handler (value) {

```

```

this.blogs = []
this.results = []
this.loading = true
this.results = (await BlogsService.index(value)).data
this.appendResults()

this.results.forEach(blog => {
  if (this.category.length > 0) {
    // console.log(this.category.indexOf(blog.category))
    if(this.category.indexOf(blog.category) === -1) {
      this.category.push(blog.category)
    }
  } else {
    this.category.push(blog.category)
  }
})
this.loading = false
this.search = value
// console.log(this.category)
}

},
data () {
  return {
    blogs: [],
    BASE_URL: "http://localhost:8081/assets/uploads/",
    search: '',
    results: [],
    category: [],
    loading: false,
  }
},
// async created () {
//   this.blogs = (await BlogsService.index()).data
// },
methods: {
  appendResults: function () {
    if (this.blogs.length < this.results.length) {
      let toAppend = this.results.slice(
        this.blogs.length,
        LOAD_NUM + this.blogs.length
      )
      this.blogs = this.blogs.concat(toAppend)
    }
  }
}

```

```

        }
    },
    navigateTo (route) {
        this.$router.push(route)
    },
    async deleteBlog (blog) {
        try {
            await BlogsService.delete(blog)
            this.refreshData()
        } catch (err) {
            console.log(err)
        }
    },
    async refreshData() {
        this.blogs = (await BlogsService.index()).data
    },
    setCategory (keyword) {
        if(keyword === ''){
            this.search = ''
            console.log('null')
        } else {
            this.search = keyword
        }
    },
},
updated () {
    let sens = document.querySelector('#blog-list-bottom')
    pageWatcher = ScrollMonitor.create(sens)
    pageWatcher.enterViewport(this.appendResults)
},
beforeUpdated () {
    if (pageWatcher) {
        pageWatcher.destroy()
        pageWatcher = null
    }
}
}
</script>
<style scoped>
.component-wrapper {
    padding-left:5px;
    padding-right:5px;
}
.hero {

```

```
margin-top: 80px;
max-width: 900px;
margin-left: auto;
margin-right: auto;
border-radius: 5px;
background: darkcyan;
height:250px;
color:white;
padding: 20px;
}

.hero h1 {
margin-top: 30px;
}
.logo {
padding-right: 20px;
}
.empty-blog {
width: 100%;
text-align: center;
padding:10px;
background:darksalmon;
color:white;
}

/* thumbnail */
.thumbnail-pic img{
width: 200px;
padding: 5px 5px 5px 5px;
border: solid 1px #ccc;
margin: 10px 10px 0px 0px;
}

.blog-info {
float: left;
}

.blog-pic {
float: left;
}

.clearfix {
clear: both;
}
```

```
.blog-list {
    border:solid 1px #fdfdf;
    margin-bottom: 10px;
    max-width: 900px;
    margin-left: auto;
    margin-right: auto;
    padding: 5px;
    box-shadow: 0 2px 4px 0 rgba(0,0,0,.1);
}

.blog-header {
    margin-top: 80px;
    max-width: 900px;
    margin-left: auto;
    margin-right: auto;
}

#blog-list-bottom{
    padding-top:4px;
}

.blog-load-finished{
    padding:4px;
    text-align: center;
    background: seagreen;
    color:white;
}

.categories {
    margin-top: 20px;
    padding: 0;
    list-style: none;
    float: left;
}

.categories li {
    float: left;
    padding: 2px;
}

.categories li a {
    padding: 5px 10px 5px 10px;
    background:paleturquoise;
```

```

        color: black;
        text-decoration: none;
    }

.categories li.clear a {
    background: tomato;
    color: white
}

.create-blog {
    margin-top: 10px;
}

@media (max-width: 768px) {
    .logo {
        width: 120px;
    }
}

</style>

```

ใน code ของผมไม่มีอะไรมาก มีเพียงเพิ่มส่วนที่เป็น hero เพิ่มเข้าไปเพื่อให้เห็นความแตกต่างระหว่าง หน้าบ้านและหลังบ้านเท่านั้นเอง แต่จะมีส่วนที่หน้าสนใจเกี่ยวกับ style responsive ของผมคือ

```

@media (max-width: 768px) {
    .logo {
        width: 120px;
    }
}

```

นั่นคือ ผมกำหนดใน ความกว้าง 768px และใช้ .logo เพื่อปรับขนาด logo ของผมนั่นเองครับ ส่วนเรื่อง responsive สามารถศึกษาเพิ่มเติมได้ที่ web ของ bootstrap เลยครับ

จากนั้นทำการเรียกใช้ Front Index Component ของเรา ใน src/router.js และทำการ route mapping

```

// Front
import FrontIndex from '@/components/Fronts/Index'

```

```
// front
{
  path: '/front',
  name: 'front',
  component: FrontIndex
},
```

สร้าง Header ในส่วนของ Front Office กัน

ก่อนหน้านี้เราได้ทำการสร้าง Header ในส่วนของ Back Office กันไว้แล้ว โดยทำการเรียกใช้ src/components/Header.vue ที่ src/App.vue ถ้าใคร สึมให้ลองกลับไปทบทวนกันดูนะครับ

ทำการสร้าง src/components/FrontHeader.vue และเพิ่ม code เข้าไปดังนี้

```
<template>
  <div>
    <!-- new navbar -->
    <div class="navbar navbar-inverse navbar-fixed-top">
      <div class="container">
        <div class="navbar-header">
          <a class="navbar-brand" href="#" v-on:click.prevent="navigateTo('/dashboard')">
            
          </a>
          <button class="navbar-toggle collapsed" data-toggle="collapse" data-target="#navcol-1">
            <span class="sr-only">Toggle navigation</span>
            <span class="icon-bar"></span><span class="icon-bar"></span>
            <span class="icon-bar"></span>
          </button>
        </div>
        <div class="collapse navbar-collapse" id="navcol-1">
          <ul class="nav navbar-nav navbar-right">
            <li role="presentation"><router-link :to="{name: 'front'}"><i class="fas fa-home"></i> Home</router-link></li>
            <li role="presentation"><router-link :to="{name: 'login'}">Login</router-link></li>
          </ul>
        </div>
      </div>
    </div>
  </div>
```

```

        <li role="presentation"><a href="#" v-
on:click="logout">Logout</a></li>
    </ul>
</div>
</div>
</div>
</div>
</template>
<script>
export default {
  methods: {
    logout () {
      this.$store.dispatch('setToken', null)
      this.$store.dispatch('setComment', null)
      this.$router.push({
        name: 'login'
      })
    },
  }
}
</script>
<style scoped>
.navbar-brand > img {
  width: 36px;
  padding: 12px 0;
  margin-top: -20px;
}
.navbar-inverse {
  background-color:#51415F;
}

.navbar-inverse .navbar-nav>li>a {
  color: #DBDBF6;
}

</style>

```

สังเกตว่า ไม่ได้มีอะไรเปลี่ยนแปลงจาก Hearder ในส่วนของ Back Office นอกจักสี และ Link Nav เท่านั้นเอง

ทำการเปิด src/main.js และทำการแก้ไข code เป็นดังนี้ครับ

```

// The Vue build version to load with the `import` command
// (runtime-only or standalone) has been set in webpack.base.conf with
// an alias.
import Vue from 'vue'
import App from './App'
import router from './router'
import { sync } from 'vuex-router-sync'
import store from './store'
import VueResource from 'vue-resource'

import BackHeader from '@/components/Header.vue'
import FrontHeader from '@/components/FrontHeader.vue'

Vue.config.productionTip = false

Vue.use(VueResource)

Vue.component('back-header', BackHeader)
Vue.component('front-header', FrontHeader)

sync(store, router)

/* eslint-disable no-new */
new Vue({
  el: '#app',
  router,
  store,
  components: { App },
  template: '<App/>'
})

```

คือผมเรียกใช้ Vue Component ขึ้นมาอีกตัวหนึ่งคือ Front Header นั้นเอง

จากนั้นเปิด src/App.vue ขึ้นมา แล้วลบ

```
<back-header />
```

ออก มาถึงตรงนี้ navbar ของเราจะหายไปทั้งหมด ไม่ต้องตกใจครับ เราเมื่อ components อยู่จะใส่มีอิหรือ ที่ไหนก็ได้ครับ

จากนั้นเปิด src/components/Fronts/Index.vue ของเราขึ้นมา แล้วทำการเพิ่ม code ดังนี้ครับ

```
<template>
<div>
  <front-header />
  <div class="hero-wrapper">
```

เมื่อเราทำการเพิ่ม Front Header มาใน Componet ตอนนี้ Front Office ของเราจะมี Navbar และครับ ทำการเปิด web browser ของเราแล้วเปิด ไปที่

<http://localhost:8080/front>

Home Login Logout

Webblot from nodejs + vuejs Ebook

By Gooddev.ME

จำนวน blog: 7 Search

html vuejs nodejs css Clear

Blog 7 ของเรา



Blog 6 ของเรา



Blog 5 ของเรา



Blog 4 ของเรา



Blog 3 ของเรา



Blog 2 ของเรา



Blog 1 ของเรา



โพลซ้อมครุภัลล์

Head ในส่วนของ Front Office ของเราแล้วครับ พร้อมหน้าตาของหน้าบ้านที่เราได้ทำการแก้ไขไปครับ

แต่พอเปิดไปที่ <http://localhost:8080/blogs>

ซึ่งเป็น Back Office navbar ของเราจะหายไป เพราะตอนนี้เราได้แยกการใช้งานโดยไม่ได้ใช้งาน navbar ที่ App.vue อีกต่อไปแต่จะแยกส่วนกันระหว่าง Component แต่ละ Component นั้นเองครับ

ทำความรู้จัก Probs

พอดูถึง Front End Development ขึ้นมา ไม่ว่าจะเป็น vuejs หรือ react จะได้ยินคำว่า probs and states บ่อย ๆ บางที่จะได้ยินคำพูดหรือ ๆ ว่า จัดการ probs and states ก็จะทำได้หมด ผลกระทบมีไปเลยครับ เพราะเข้าใจ probs and states ตั้งแต่วันแรก ๆ เลย แต่เขียนไม่ได้เช่น สงสัย IQ ผิดจะไม่ถึง ..

states คืออะไร ตอบลื้น ๆ ง่าย ๆ เลยครับ ตัวแปรทุกตัวที่เราเก็บไว้ใน data () คือ states ครับ

~~Props~~ คืออะไร อะไรที่ส่งไปให้ อีก components นึง เรียกว่า ~~probs~~ ครับ โดยเราส่งไป และต้องไปเขียนรับด้วยครับ สามารถใช้แทน data () ของเราได้เลยครับ เพราะจริง ๆ คือ data () จาก components อื่นส่งมาให้นั่นเอง

ทำการเลือกการแสดงผล Header ให้ตรงกับ Components

หัวข้อสวยหรือเลยครับ เลือก header หรือ navbar ของเรามาแสดงให้ถูกต้องว่าอันไหน หลังบ้าน อันไหน หน้าบ้านนั้นเองครับ

ทำการเปลี่ยนชื่อไฟล์ src/components/Header.vue ไปเป็น
src/components/BackHeader.vue ก่อนครับ

จากนั้นสร้างไฟล์ src/components/Header.vue ขึ้นมา แล้วทำการเขียน code ดังนี้ครับ

```
<template>
  <div>
    <front-header v-if="navsel === 'front' " />
    <back-header v-if="navsel === 'back'" />
  </div>
</template>
<script>
import FrontHeader from '@/components/FrontHeader.vue'
import BackHeader from '@/components/BackHeader.vue'

export default {
  props: ['navsel'],
  components: {
    FrontHeader,
    BackHeader
  },
}
</script>
```

สังเกต code ของเราตรงนี้นะครับ

```
props: ['navsel'],
```

คือผมเตรียมไว้รับ navsel ที่จะส่งมาตอนเรียกใช้ เพื่อไปเลือก component มาแสดงผลนั่นเองครับ

จากนั้นเราจะทำการผูก component ของเราเพื่อให้สามารถเรียกใช้งานได้โดยไม่ต้องมานั่ง Import กันบ่อย โดยผมเปิด src/main.js ขึ้นมาแล้วทำการแก้ไข code ของเราดังนี้

```
// The Vue build version to load with the `import` command
// (runtime-only or standalone) has been set in webpack.base.conf with
// an alias.
import Vue from 'vue'
import App from './App'
import router from './router'
import { sync } from 'vuex-router-sync'
import store from './store'
import VueResource from 'vue-resource'

import Header from '@/components/Header.vue'
```

```
Vue.config.productionTip = false

Vue.use(VueResource)

Vue.component('main-header', Header)

sync(store, router)

/* eslint-disable no-new */
new Vue({
  el: '#app',
  router,
  store,
  components: { App },
  template: '<App/>'
})
```

สังเกตมั้ยครับ ผมใช้งาน Header Components เพียงตัวเดียวเท่านั้น โดย Header ของผมจะไปเรียบ FrontHeader หรือ BackHeader จากเงื่อนไข หรือค่าของ probs ที่ส่งมาなんเอง

เปิด src/components/Blogs/Index.vue ขึ้นมาครับ แล้วแก้ไข code ดังนี้ครับ

```
<template>
  <div>
    <main-header navsel="back"></main-header>
```

เห็นมั้ยครับผมส่ง navsel="back" ไปลองเปิด Web browser ไปที่ <http://localhost:8080/blogs> จะเห็นว่า BackHeader ถูกเรียกใช้งานได้อย่างถูกต้องแล้ว

The screenshot shows a web application interface for managing blogs. At the top, there is a navigation bar with links for Blogs, Users, Comments, Login, and Logout. On the left, there is a sidebar with a search bar and a "Create blog" button. Below the search bar, there are four categories: html, vuejs, nodejs, and css, each with a corresponding colored button. A "Clear" button is also present.

Blog 7 ของเรา

 Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown pr...

Category: html
Create: 2019-03-17T18:48:32.394Z

[View Blog](#) [Edit Blog](#) [Delete](#)

Blog 6 ของเรา

 Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown pr...

Category: vuejs
Create: 2019-03-17T18:46:53.343Z

[View Blog](#) [Edit Blog](#) [Delete](#)

Blog 5 ของเรา

 Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown pr...

Category: nodejs
Create: 2019-03-17T18:45:10.805Z

[View Blog](#) [Edit Blog](#) [Delete](#)

จากนั้นทำการเปิด `src/components/Fronts/Index.vue` ขึ้นมาแล้ว ทำการเพิ่ม code ส่วนนี้เข้าไปครับ

```
<template>
  <div>
    <main-header navsel="front"></main-header>
```

แล้วเปิดไปที่ <http://localhost:8080/front>

จะเห็นว่าโปรแกรมที่เราเขียนขึ้นทำการเลือก Header มาแสดงผลได้อย่างถูกต้องแล้วนั่นเอง

The screenshot shows a web application interface. At the top, there is a dark header bar with a large green 'V' logo on the left, and 'Home', 'Login', and 'Logout' buttons on the right. Below the header is a teal-colored banner featuring a large white 'V' icon and the text 'Webblot from nodejs + vuejs Ebook' followed by 'By Gooddev.ME'. The main content area has a white background. It displays a list of three blog posts, each in its own card:

- Blog 7 ของเรา**
A thumbnail image of a white convertible car. The post title is 'Blog 7 ของเรา'. The content is placeholder text: 'Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown pr...'. Category: 'html', Create: '2019-03-17T18:48:32.394Z'. A blue 'View Blog' button is at the bottom.
- Blog 6 ของเรา**
A thumbnail image of a person sitting on a beach chair under a red umbrella. The post title is 'Blog 6 ของเรา'. The content is placeholder text: 'Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown pr...'. Category: 'vuejs', Create: '2019-03-17T18:46:53.343Z'. A blue 'View Blog' button is at the bottom.
- Blog 5 ของเรา**
A thumbnail image of pink flowers. The post title is 'Blog 5 ของเรา'. The content is placeholder text: 'Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown pr...'. Category: 'nodejs', Create: '2019-03-17T18:45:10.805Z'. A blue 'View Blog' button is at the bottom.

มาถึงตอนนี้เราได้มี function ในการเลือก navbar หรือ Header เพื่อทำการแสดงผลได้อย่างถูกต้อง เรา ก็นำ component ในส่วนของ Header ทั้ง Front Office และ Back Office ไปแปะในหน้าต่าง ๆ ของเรารวมทั้งค่าไปหา probs ของ Header Component ของเราให้ถูกต้อง เรา ก็จะเสร็จการทำงานในส่วน Header แล้วครับ

ทำการไล่เปลี่ยน header ของเราให้ถูกต้องตามการใช้งานดังนี้ครับ

Back Header

Blogs/Index.vue

Blogs/CreateBlog.vue

Blogs/EditBlog.vue

Blogs>ShowBlog.vue

Users/Index.vue

Users/EditUser.vue

Users>ShowUser.vue

Users/CreateUser.vue

Comments/Index.vue

Front Header

Fronts/Index.vue

Fronts>ShowBlog.vue

เมื่อทำการเปลี่ยน header เสร็จแล้วเราจะสามารถทำงานได้ง่ายขึ้นไม่ต้องค่อยกรอก URL เพื่อทดสอบ และหน้าตากำรใช้งานก็ดีขึ้นเช่นกัน

ทำส่วนการ อ่าน Blog ของ Front Office

ใกล้เข้ามายังนี่อีกนิดกับความสำเร็จในบทนี้เราจะมาทำการสร้างส่วนการอ่าน blog ของ Front Office ของเรา กันครับ โดยเราทำการสร้าง src/components/Front/ShowBlog.vue ขึ้นมา ไป copy code หลังบ้านที่เราเขียนไว้มาใส่ แล้วทำการปรับแต่งไม่ให้แสดงปุ่มต่าง ๆ ที่เราไม่ต้องการดังนี้ครับ

```
<template>
  <div class="component-wrapper container">
    <main-header navsel="front"></main-header>
    <div class="hero-wrapper">
      <div class="hero">
        
        <h1>Webblot from nodejs + vuejs Ebook</h1>
        <p>By Gooddev.ME</p>
      </div>
    </div>
    <div class="blog-wrapper" v-if="blog != null">
      <h1>{{ blog.title }}</h1>
      <p><strong>Category:</strong> <a href="#" v-on:click.prevent="navigateTo('/front?search=${blog.category}')">{{ blog.category }}</a></p>
      <div class="content" v-html="blog.content"></div>
      <!-- <p>category: {{ blog.category }}</p>
      <p>status: {{ blog.status }}</p> -->
    </div>
    <div class="back-nav"><button class="btn btn-success" v-on:click="navigateTo('/front')"><i class="fas fa-arrow-left"></i> Back..</button></div>
    <transition name="fade">
      <div v-if="resultUpdated != ''" class="popup-msg">
        <p>{{ resultUpdated }}</p>
      </div>
    </transition>
    <br>
  </div>
</template>
<script>

import BlogsService from '@/services/BlogsService'
import UsersService from '@/services/UsersService'

export default {


```

```

data () {
  return {
    blog: null,
    resultUpdated: '',
    users:null
  }
},
delete () {
  console.log('delete blog')
},
async created () {
  // get blog

  // check permission first
  try {
    let blogId = this.$route.params.blogId
    this.blog = (await BlogsService.show(blogId)).data
  } catch (error) {
    console.log (error)
  }
},
methods: {
  navigateTo (route) {
    this.$router.push(route)
  },
}
}
</script>
<style scoped>
.popup-msg {
  box-shadow: 0 2px 4px 0 rgba(0,0,0,.2);
  border: solid 1px #ddd;
  background: #fff;
  max-width:200px;
  padding: 10px;
  position:fixed;
  bottom:0;
  right:0;
  border-radius: 5px;
  margin-bottom: 5px;
  margin-right: 5px;
}
.hero {

```

```

margin-top: 80px;
border-radius: 5px;
background: darkcyan;
height:250px;
color:white;
padding: 20px;
}

.hero h1 {
  margin-top: 30px;
}

.blog-wrapper {
  margin-top:20px;
  padding: 40px;
  box-shadow: 0 2px 4px 0 rgba(0,0,0,.2);
}
.back-nav {
  margin-top: 20px;
  text-align: center;
}
.blog-wrapper h1{
  text-align: center;
  font-family: 'kanit';
  padding-bottom: 50px;
}

.blog-wrapper p {
  font-family: 'kanit';
  font-size: 1.4em;
  padding-bottom:20px;
}

.blog-wrapper .content {
  font-family: 'kanit';
  font-size: 1.2em;
}
</style>

```

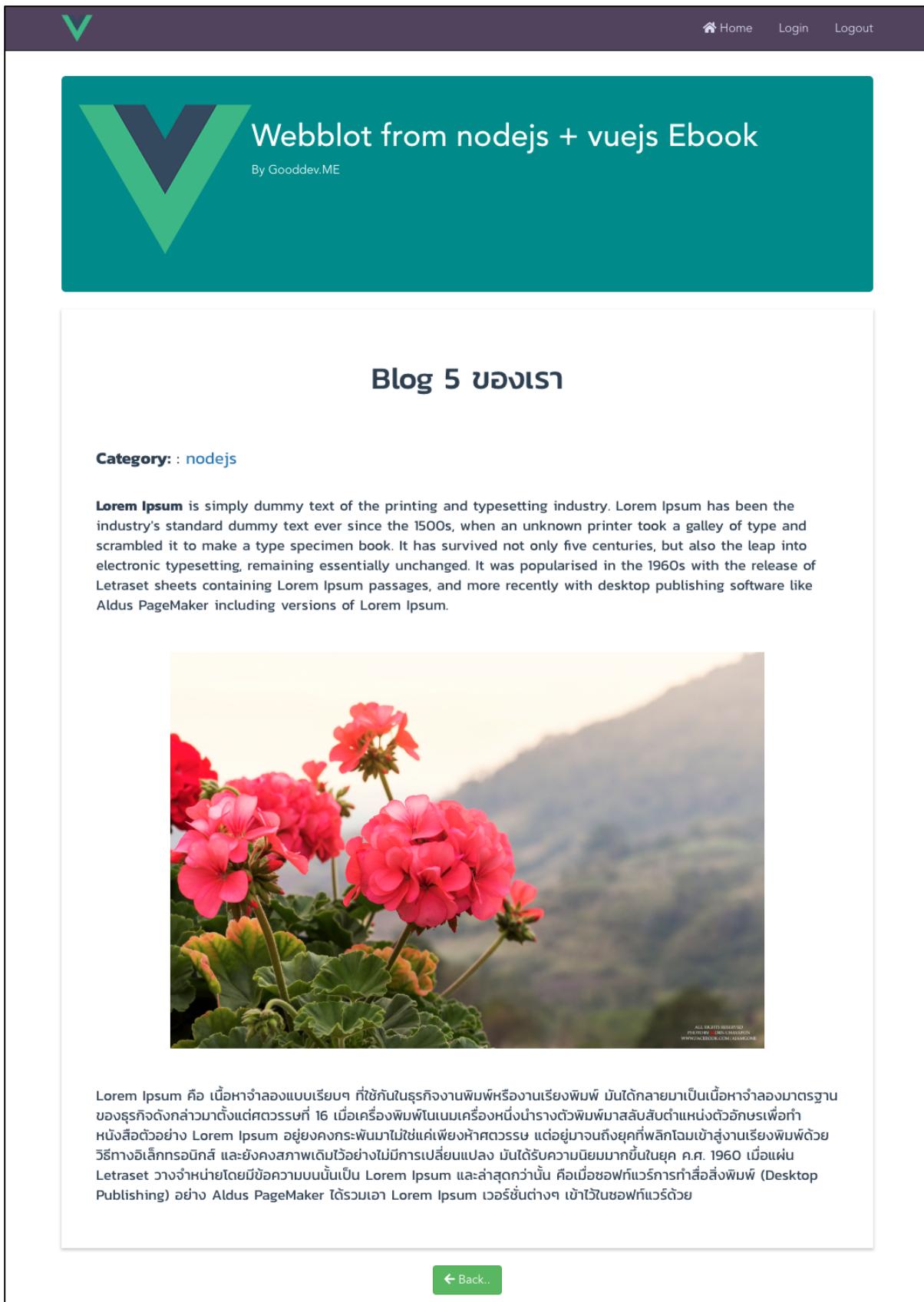
แล้วทำการ route mapping เพื่อเพิ่ม url link และเรียกใช้งาน Component ของเรา โดยเปิด router/index.js ขึ้นแล้วทำการเพิ่ม code ชุดนี้เข้าไป

```
import FrontShow from '@/components/Fronts>ShowBlog'
```

```
{
  path: '/front/read/:blogId',
  name: 'front-read',
  component: FrontShow
},
```

นั่นคือเราเรียกใช้ src/components/Fronts>ShowBlog.vue พร้อมทั้งส่ง blogId
ไปให้มันด้วยนั่นเอง

ทำการทดสอบโดยเปิด Blog จากหน้า Front Office ของเราจะได้ผลดังนี้



โดยความพยายามของการจัดเอกสาร และหัวข้อต่าง เราสามารถแก้ไขและปรับแต่งมาตั้งแต่ตอนเราทำการเขียน blog ด้วย CKEdit ขึ้นอยู่กับใจเดียวกัน ความสามารถของผู้เขียนเบล็อกแต่ละคนครับ

ໄຟລົດຂາມກົດ

บทที่ 16 ทำส่วน comment บทความ

สร้าง User Type เพื่อจัดการ Permission

แยกส่วน Login ใน Authen

เมื่อเรามี user สองชนิด ผມเลยแยกส่วนในการ login ขึ้นมาจากกัน ถ้าเป็น PHP เรายังทำการ redirect page ได้เลยจาก login form หน้าเดียวกัน แต่พอเป็น javascript นั้นจะไม่เหมาะสม เพราะลักษณะการใช้งาน login form นั้นแตกต่างกัน รวมถึงการส่ง user มาทำการเลือกที่ client นั้น ทำให้ระบบมีความเสี่ยงอยู่ พอกสมควร

ทำการแก้ไข function login ใน src/controllers/UserAuthenController.js ของ เรา เพื่อให้เช็ค admin permission ดังนี้

```
async login (req, res) {
  try {
    const {email, password} = req.body
    const user = await User.findOne({
      where: {
        email: email
        // password: password
        // status: 'active'
      }
    })

    if(!user) {
      return res.status(403).send({
        error: 'User/Password not correct'
      })
    }

    const isPasswordValid = await user.comparePassword(password)
    if (!isPasswordValid) {
      return res.status(403).send({
        error: 'User/Password not correct'
      })
    }

    if(user.type != "admin") {
```

```

        return res.status(403).send({
            error: 'Permission not correct'
        })
    }

const userJSON = user.toJSON()

res.send({
    user: userJSON,
    token: jwtSignUser(userJSON)
})

} catch (error) {
    res.status(500).send({
        error: 'Error! from get user'
    })
}
}
}

```

ลังเกตว่าจะมีการ เช็ค user.type ในขั้นตอนสุดท้ายก่อนทำการส่งข้อมูลออกไป

จากนั้นทำการ สร้างอีก function นึง เพื่อทำการตรวจสอบการ login เป็นชนิดของ ผู้ใช้งานโดยผม copy function login ไปลงส่วนของ สล็อกอิน นั่นเอง และทำการเปลี่ยนชื่อ พังก์ชันเป็น clientLogin ดังนี้

```

async clientLogin (req, res) {
    try {
        const {email, password} = req.body
        const user = await User.findOne({
            where: {
                email: email
                // password: password
                // status: 'active'
            }
        })

        if(!user) {
            return res.status(403).send({
                error: 'User/Password not correct'
            })
        }
    }
}

```

```

const isPasswordValid = await user.comparePassword(password)
if (!isPasswordValid) {
    return res.status(403).send({
        error: 'User/Password not correct'
    })
}

// dont't check permission type

const userJSON = user.toJSON()

res.send({
    user: userJSON,
    token: jwtSignUser(userJSON)
})

} catch (error) {
    res.status(500).send({
        error: 'Error! from get user'
    })
}
},

```

เมื่อเราสร้าง Controller Function เพิ่มขึ้นมาแล้ว เราต้องไปทำการ route mapping เพื่อให้สามารถเรียกใช้ผ่าน api ของเราได้ โดยทำการเพิ่ม

```

app.post('/front/login',
    UserAuthenController.clientLogin
)

```

ที่ src/routes.js ใน server ของเรา

The screenshot shows a 'Login' page with fields for 'Username' and 'Password', and a 'Sign In' button. Below the button, a red message says 'Permission not correct'. At the bottom of the page, there is a small '> |' icon.

The developer console (Console tab) is open, displaying the following log entries:

- [HMR] Waiting for update signal from WDS...
- search: undefined
- Failed to load resource: the server responded with a status of 403 (Forbidden)
 - Error: Request failed with status code 403
 - at createError (createError.js?716d0:16)
 - at settle (settle.js?7db52:18)
 - at XMLHttpRequest.handleLoad (xhr.js?ec6c:77)
- POST http://localhost:8081/login 403 (Forbidden)
 - Error: Request failed with status code 403
 - at createError (createError.js?716d0:16)
 - at settle (settle.js?7db52:18)
 - at XMLHttpRequest.handleLoad (xhr.js?ec6c:77)

จากนั้นทำการทดสอบการล็อกอินในส่วนของแอดมิน ทำการล็อกอินได้เมื่ອ่อนเดิม เมื่อทดสอบโดยการล็อกอินด้วยยูสเซอร์ที่เป็นชนิดของ Client สังเกตว่า จะไม่สามารถล็อกอินได้แล้ว และตอบกลับมาว่า Permission not correct

ทดสอบการล็อกอินในส่วนของ admin ด้วย Postman โดย Post User Data ไปที่ front/login ด้วย User ทั้งสองชนิด จะต้องทำการ login ได้ ทั้งนี้เพราะผู้ให้สิทธิ์ permission ของ admin นั้นใหญ่สุด สามารถใช้ account ตัวเองทำการ comment ได้เลยโดยไม่ต้องใช้ user client นั่นเอง

The screenshot shows the Postman interface with the following details:

- Request URL:** POST http://localhost:8081/front/login
- Body Content (JSON):**

```

1 {
2   "email": "user@gmail.com",
3   "password": "12345678"
4 }
    
```

- Response Status:** 200 OK
- Response Body (Pretty JSON):**

```

1 {
2   "user": {
3     "id": 3,
4     "email": "user@gmail.com",
5     "password": "$2a$08$Rx1BqeRyXTZcwAj6emozD.NGRF23aWGP0hwh1P2pBgSU0i9T/Q/GG",
6     "name": "user",
7     "lastname": "nakrub",
8     "status": "active",
9     "type": "admin",
10    "createdAt": "2019-03-17T07:34:13.716Z",
11    "updatedAt": "2019-03-17T07:34:13.716Z"
12  },
13  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
.eyJpZCI6MywiZW1hdWwiOiJ1c2VyQGdtYWlsLmNvbSIiInBhc3N3b3JkIjoiJDJhJDA4JFJ4bEJxZVJ5WFRAY3dBajZlbW96RC
GMjNhV0dQT2h3aDFQMrBCZ1NVT2k5VC9RL0dHiwibmFtZSI6InVzZXIiLCJsYXN0bmFtZSI6Im5ha3J1YiIsInN0YXR1cyI6Im
    
```

สร้าง form login เพื่อทำการ comment

ในส่วนนี้เราจะมาสร้าง Login Form ในส่วนของ Client เพื่อให้สามารถทำการคอมเม้นท์ความ เมื่อทำการล็อกอินเท่านั้น หลังจากล็อกอินแล้ว สามารถ ทำการแก้ไข และลบ คอมเม้นของตัวเองได้

ผนวกกับการเพิ่ม Service เพื่อทำการ Login เป็นแบบ Client ก่อน โดยทำการเพิ่ม service function เข้าไปใน src/services/AuthenService.js ดังนี้

```
clientLogin (credentials) {
```

```
    return Api().post('front/login', credentials)
}
```

แล้ว from login ง่าย ๆ เพื่อทดสอบระบบของเราขึ้น เนื่องจาก form login นี้เราจะทำให้สามารถใช้งานร่วมกันได้ในทุก ๆ หน้าของเรา ผ่านเลยไปมันไว้ที่ FrontHeader ของเรานั้นเอง เปิด src/components/FrontHeader.vue ในส่วน client ขึ้นมา จากนั้นทำการเพิ่ม form ของเราเข้าไปดังนี้

```
<div class="login-wrapper">
  <form v-on:submit.prevent="clientLogin">
    <p>Email: <input type="text" v-model="email"> </p>
    <p>Password: <input type="text" v-model="password"> </p>
    <p><button type="submit">Login</button></p>
  </form>
  <div class="error">
    <p v-if="error">{{error}}</p>
  </div>
</div>
```

จากนั้นเพิ่มตัวแปรของเราเข้าไปใน data () ดังนี้

```
email: '',
password: '',
error: ''
```

ทำการ Import Service เพื่อใช้งาน

```
import AuthenService from '@/services/AuthenService'
```

จากนั้นทำการเขียน function เพื่อทำการ login ใน methods ดังนี้

```
async clientLogin () {
  console.log(`acc: ${this.email} -${this.password}`)
  try {
    const response = await AuthenService.clientLogin({
      email: this.email,
      password: this.password
    })

    this.error = ''

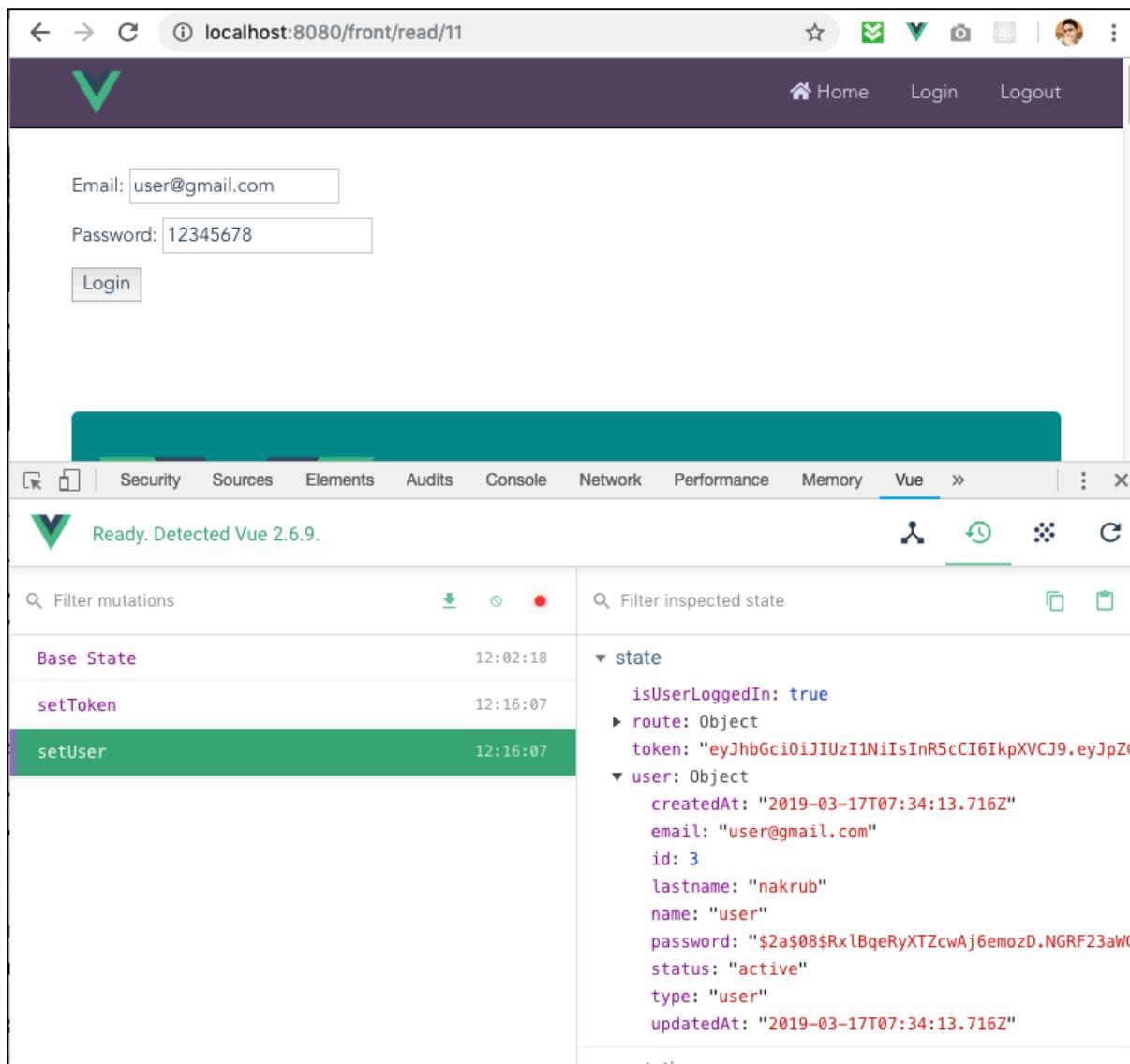
    this.$store.dispatch('setToken', response.data.token)
  } catch (err) {
    this.error = err.message
  }
}
```

```
    this.$store.dispatch('setUser', response.data.user)
} catch (error) {
  console.log(error)
  this.error = error.response.data.error
}
},
```

ทำการใส่ style ดังนี้

```
.login-wrapper {
  margin-top: 80px;
}
```

จากนั้นทำการทดสอบการ login และเปิด Vue Devtools เพื่อ เลือกไป Vuex เมนู จะเห็นว่า มีการส่งกลับ user data กลับมาจาก server และงว่าเรา login ผ่านแล้ว นั่นเอง หาก login ผิดพลาดก็จะแสดง error ออกมาที่หน้า form login จากที่เรา เขียนไว้นั่นเอง



เปลี่ยนสถานการ Login ที่ navbar

เมื่อเราทำการ login สำเร็จแล้ว เราจะทำการเปลี่ยน menu login ให้เป็นชื่อผู้ใช้งาน พร้อมทั้งแสดงสถานะ login/logout เสร็จสมบูรณ์

ตอนนี้ presuming ว่าทุกคนเรื่องเขียนกันเก่งแล้ว ผมจะเขียน code พร้อมใส่ style เข้าไป เลยนะคับ ถ้าไม่เข้าใจให้ไล่ code จาก template ไปหา style นะครับ

ทำการเปิด `FrontHeader.vue` ของเรารีบๆมา แก้ไข `<template>` ของเราเป็นดังนี้

```
<template>
<div>
  <!-- new navbar -->
```

```

<div class="navbar navbar-inverse navbar-fixed-top">
  <div class="container">
    <div class="navbar-header">
      <a class="navbar-brand" href="#" v-on:click.prevent="navigateTo('/dashboard')">
        
      </a>
      <button class="navbar-toggle collapsed" data-toggle="collapse" data-target="#navcol-1">
        <span class="sr-only">Toggle navigation</span>
        <span class="icon-bar"></span><span class="icon-bar"></span>
        <span class="icon-bar"></span>
      </button>
    </div>
    <div class="collapse navbar-collapse" id="navcol-1">
      <ul class="nav navbar-nav navbar-right">
        <li role="presentation"><router-link :to="{name: 'front'}">
<i class="fas fa-home"></i> Home</router-link></li>
        <li v-if="!isUserLoggedIn" role="presentation"><a href="#" v-on:click.prevent="showLogin = true" >Login</a></li>
          <transition name="fade">
            <li v-if="isUserLoggedIn" role="presentation"><router-link v-bind:to="{name: 'login'}" >{{user.name}}</router-link></li>
          </transition>
        <li v-if="isUserLoggedIn" role="presentation"><a href="#" v-on:click.prevent="logout">Logout</a></li>
      </ul>
    </div>
  </div>
<div class="modal" v-if="showLogin">
  <transition name="fade">
    <div class="login-wrapper">
      <h3>Client Login</h3>
      <form v-on:submit.prevent="clientLogin" class="form-horizontal">
        <div class="form-group">
          <label class="control-label col-md-3">Email:</label>
          <div class="col-md-9">
            <input placeholder="email" type="email" v-model="email" class="form-control" />
          </div>
        </div>
      </form>
    </div>
  </transition>
</div>

```

```

<div class="form-group">
    <label class="control-label col-md-3">Password:</label>
    <div class="col-md-9">
        <input type="password" placeholder="password" v-
model="password" class="form-control" />
    </div>
</div>
<div class="form-group">
    <div class="col-md-offset-3 col-md-9">
        <button class="btn btn-success btn-sm" type="submit"><i
class="fas fa-key"></i> Login</button>
        <button v-on:click.prevent="showLogin" = false"
class="btn btn-danger btn-sm" type="button"><i class="fas fa-times-
circle"></i> Close</button>
    </div>
</div>
<div class="error">
    <p v-if="error">{{error}}</p>
</div>
</form>
</div>
</transition>
</div>
<transition name="fade">
    <div v-if="resultUpdated != ''" class="popup-msg">
        <p>{{ resultUpdated }}</p>
    </div>
</transition>
</div>
</template>

```

จากนั้นแก้ไข script ของเรามาเป็นดังนี้

```

<script>
import {mapState} from 'vuex'
import AuthenService from '@/services/AuthenService'

export default {
  data () {
    return {
      email: '',
      password: '',
      error: ''
    }
  }
}

```

```

        showLogin: false,
        resultUpdated: ''
    }
},
computed: {
    ...mapState([
        'isUserLoggedIn',
        'user'
    ]),
},
methods: {
    logout () {
        this.$store.dispatch('setToken', null)
        this.$store.dispatch('setUser', null)
        // this.$router.push({
        //   name: 'login'
        // })
        this.resultUpdated = "Logout successful."
        setTimeout(() => this.resultUpdated = '', 3000)
    },
    async clientLogin () {
        console.log(`acc: ${this.email} -${this.password}`)
        try {
            const response = await AuthService.clientLogin({
                email: this.email,
                password: this.password
            })

            this.error = ''

            this.$store.dispatch('setToken', response.data.token)
            this.$store.dispatch('setUser', response.data.user)

            // this.$router.push({
            //   name: 'blogs'
            // })
            // console.log(response.data)
            this.email = '',
            this.password = '',
            this.showLogin = false
            this.resultUpdated = "Login successful."
            setTimeout(() => this.resultUpdated = '', 3000)
        }
    }
}

```

```

    } catch (error) {
      console.log(error)
      this.error = error.response.data.error
      this.email = ''
      this.password = ''
      setTimeout(() => this.error = '', 3000)
    }
  },
}
</script>

```

แล้วแก้ไข style ของเราเป็นดังนี้

```

<style scoped>
.error {
  color: red;
  text-align: center;
}
.popup-msg {
  box-shadow: 0 2px 4px 0 rgba(0,0,0,.2);
  border: solid 1px #ddd;
  background: #fff;
  max-width:200px;
  padding: 10px;
  position:fixed;
  bottom:0;
  right:0;
  border-radius: 5px;
  margin-bottom: 5px;
  margin-right: 5px;
}
.navbar-brand > img {
  width: 36px;
  padding: 12px 0;
  margin-top: -20px;
}
.navbar-inverse {
  background-color:#51415F;
}

.navbar-inverse .navbar-nav>li>a {
  color: #DBDBF6;
}

```

```

}

.modal {
  display: block; /* Hidden by default */
  position: fixed; /* Stay in place */
  z-index: 10; /* Sit on top */
  left: 0;
  top: 0;
  width: 100%; /* Full width */
  height: 100%; /* Full height */
  overflow: auto; /* Enable scroll if needed */
  background-color: rgb(0,0,0); /* Fallback color */
  background-color: rgba(0,0,0,0.4); /* Black w/ opacity */
}

.login-wrapper {
  box-shadow: 0 2px 4px 0 rgba(0,0,0,.5);
  border: solid 1px #ddd;
  width: 320px;
  padding: 10px 30px 20px 30px;
  background-color: #fefefe;
  margin: 15% auto; /* 15% from the top and centered */
  /* padding: 20px; */
}

.login-wrapper h3 {
  text-align: center;
  padding-bottom: 10px;
}
</style>

```

โดยปุ่ม login ใน navbar ของผมจะทำหน้าที่ซ่อนหรือ แสดง form login ของผมนั้นเอง จากนั้นเมื่อทำการ login ผ่านแล้วจะไปทำการ แสดง navbar link ที่ซ่อนอยู่ พร้อมทั้งล่งชื่อของผู้ใช้งานไปแสดงด้วยนั้นเอง

โดยทั้งการ login และ logout ผมใช้ตัวแปรเก็บสถานการ login / logout โดยกำหนดเงื่อนไขในการแสดงผล คือ ถ้าค่าเป็น null ไม่แสดงผล แล้วใช้ transition และ setTimeout มาจัดการเรื่องการหยุดการแสดงผลนั้นเอง

เมื่อทดสอบการ login/logout จะแสดงผลเป็น popup ที่มุ่งช้ายขึ้นมา พร้อม เมนู login ของเราเป็นชื่อ ผู้ใช้ ตาม user ที่ login เข้ามา

จัดการเรื่องเงื่อนไขในการสมัคร

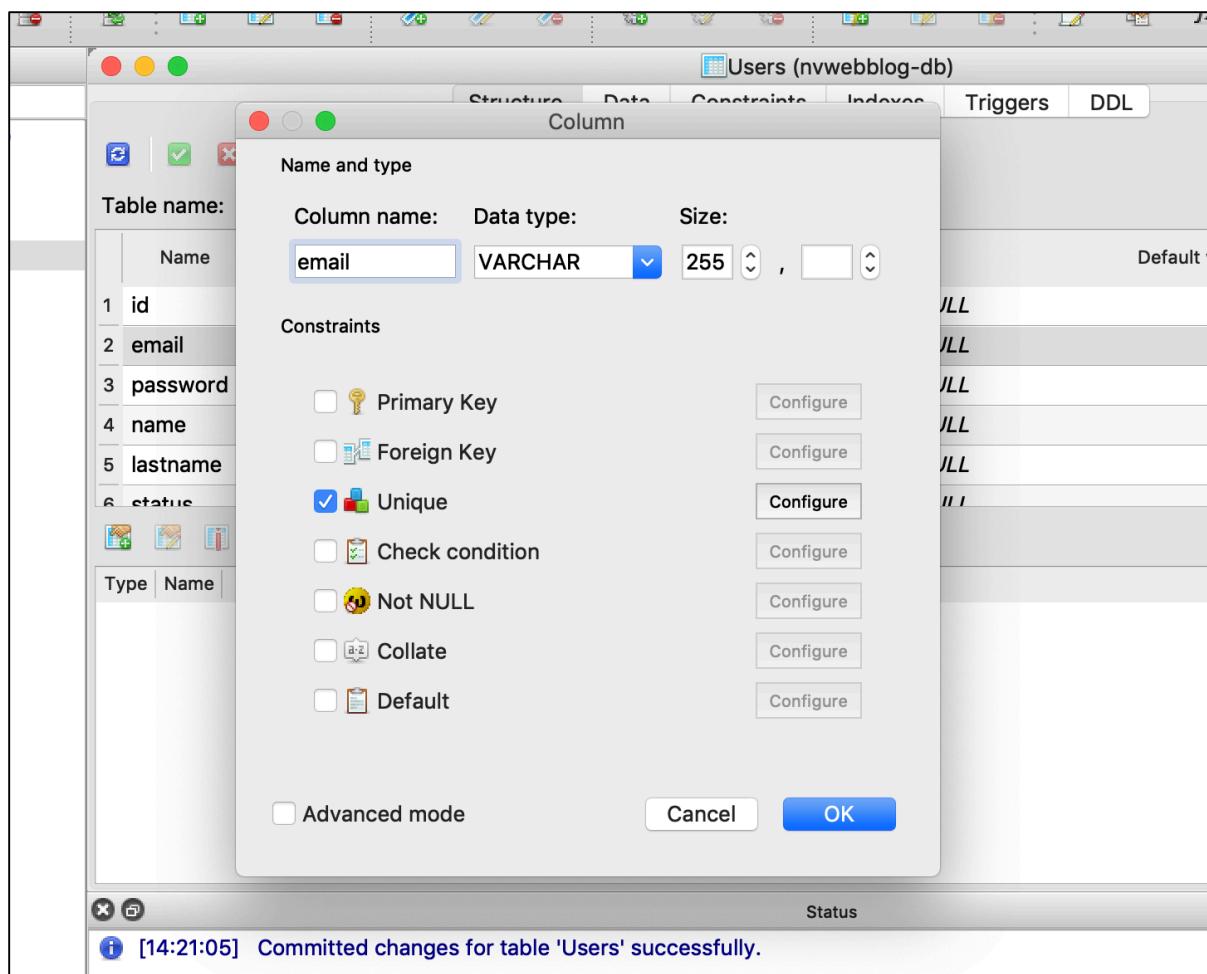
ในบทที่ผ่านมาเราได้ทำส่วนของการ login เข้าใช้งานเรียบร้อยแล้ว แต่ยังขาด ส่วนการสมัครของ client ที่เข้ามาใช้งานระบบ ในบทนี้เราจะทำส่วนของการสมัคร หรือ register กันครับ

โดยก่อนหน้านี้ model ของเราสามารถทำการสมัครโดยใช้งานซ้ำกันได้ ซึ่งใน ความเป็นจริงนั้น การทำเช่นนี้ ถือว่าผิดร้ายแรง เพราะเราจะแยก user ออกจาก กันยากมาก ดังนั้นเราต้องทำการ แก้ไขให้ไม่สามารถใช้ email ที่ทำการสมัครไป แล้วมาสมัครใหม่ได้ โดยทำการเปิด User Model ของเราขึ้นมาแล้วทำการแก้ไข field ของ email ของเราดังนี้

```
email: {  
    type: DataTypes.STRING,  
    unique: true  
},
```

แต่ทุกครั้งที่เราแก้ model ถ้าเราไม่ทำการ force true ที่ src/app.js ที่ server ของเรา database ของเราจะไม่มีการเปลี่ยนแปลงใด ๆ แต่หาก เราทำการ force true ข้อมูลที่เรากรอกไว้ก็จะหายไปหมด

เราสามารถทำการแก้ database ของเราให้ตรงกันได้ โดยใช้ SQLite Studio ทำการแก้ไข field ที่เราแก้ไขให้ตรงกันกับ Model ของเรา



จากนั้นทำการ Restart Server ของเรา

แล้วทำการแก้ไข ใน api ของเราให้แจ้งเตือน error กลับมาเมื่อมีการใช้งาน email ซ้ำกันดังนี้ เปิด UserController ของเราขึ้นมาแล้วทำการแก้ไขดังนี้ครับ

```
// create user
async create (req, res) {
  try {
    const user = await User.create(req.body)
    res.send(user.toJSON())
  } catch (err) {
    res.status(500).send({
      error: 'User already in system'
    })
  }
},
```

ตอนนี้เราจะไม่สามารถใช้ email ที่อยู่ในระบบทำการสมัครได้แล้วครับ ทดสอบโดย Postman กันก่อน

The screenshot shows the Postman interface with a failed POST request to `http://localhost:8081/user`. The request body is a JSON object:

```
1 {  
2   "email": "korn@gmail.com",  
3   "password": "12345678",  
4   "name": "korn1",  
5   "lastname": "korn1lastname",  
6   "status": "active",  
7   "type": "user"  
8 }
```

The response status is `500 Internal Server Error`, and the response body is:

```
1 {  
2   "error": "User already in system"  
3 }
```

เนื่องจาก email `korn@gmail.com` ของผมถูกใช้ในการสมัครไปแล้ว จึงไม่สามารถนำกลับมาใช้สมัครใหม่ได้อีกนั้นเอง

นอกจากนี้เราสามารถใช้ package อื่นมาจัดการเรื่องใช้ password ที่ปลอดภัยได้หลายตัว เช่น vue password ซึ่งสามารถศึกษาได้จากที่นี่ <https://www.npmjs.com/package/vue-password> การใช้งานนั้นง่ายมาก โดยหนังสือเล่มนี้จะไม่สอน แต่หากใช้งานแล้วติดปัญหา สามารถสอบถามได้ในกลุ่มครับ

ทำ Form Register

ในส่วนนี้เราจะทำการสร้าง form register เพื่อทำการสมัคร เพื่อให้สามารถทำการ comment บทความของเราได้นั่นเอง

โดยผมจะทำการเพิ่ม form register ที่ FrontHeader ของเราดังนี้

```
<div class="modal" v-if="showRegister">
    <transition name="fade">
        <div class="login-wrapper">
            <h3>Client Register</h3>
            <form      v-on:submit.prevent="clientRegister"      class="form-
horizontal">
                <div class="form-group">
                    <label class="control-label col-md-3">Email:</label>
                    <div class="col-md-9">
                        <input required placeholder="email" type="email" v-
model="client.email" class="form-control" />
                    </div>
                </div>
                <div class="form-group">
                    <label class="control-label col-md-3">Password:</label>
                    <div class="col-md-9">
                        <input required type="password" placeholder="password" v-
model="client.password" class="form-control" />
                    </div>
                </div>
                <div class="form-group">
                    <label class="control-label col-md-3">Name:</label>
                    <div class="col-md-9">
                        <input required type="text" placeholder="name" v-
model="client.name" class="form-control" />
                    </div>
                </div>
                <div class="form-group">
                    <label class="control-label col-md-3">Lastname:</label>
                    <div class="col-md-9">
                        <input required type="text" placeholder="lastname" v-
model="client.lastname" class="form-control" />
                    </div>
                </div>
                <div class="form-group">
                    <div class="col-md-offset-3 col-md-9">
```

```

        <button class="btn btn-success btn-sm" type="submit"><i
class="fas fa-key"></i> Register</button>
        <button v-on:click.prevent="showRegister = false"
class="btn btn-danger btn-sm" type="button"><i class="fas fa-times-
circle"></i> Close</button>
    </div>
</div>
<div class="error">
    <p v-if="error">{{error}}</p>
</div>
</form>
</div>
</transition>
</div>
<transition name="fade">
    <div v-if="resultUpdated != ''" class="popup-msg">
        <p>{{ resultUpdated }}</p>
    </div>
</transition>

```

เราใส่ส่วนนี้เข้าไปต่อจาก form login ของเราใน FrontHeader ครับ

จากนั้นเราไปเพิ่มตัวแปรใน data () ดังนี้

```

client: {
    name: '',
    lastname: '',
    email: '',
    password: '',
    status: 'active',
    type: 'user'
}

```

แล้วทำการเพิ่ม function ในการ register ใน methods ของเราดังนี้

```

async clientRegister () {
    console.log(this.client)
    try {
        await UserService.post(this.client)
        this.client = {}
        this.showRegister = false
    } catch (err) {
        this.error = err.message
    }
}

```

```
        this.resultUpdated = "Register successful, Please login first."
        setTimeout(() => this.resultUpdated = '', 5000)
    } catch (error) {
        console.log(error)
        this.client = {}
        this.error = error.response.data.error
        setTimeout(() => this.error = '', 5000)
    }
},
```

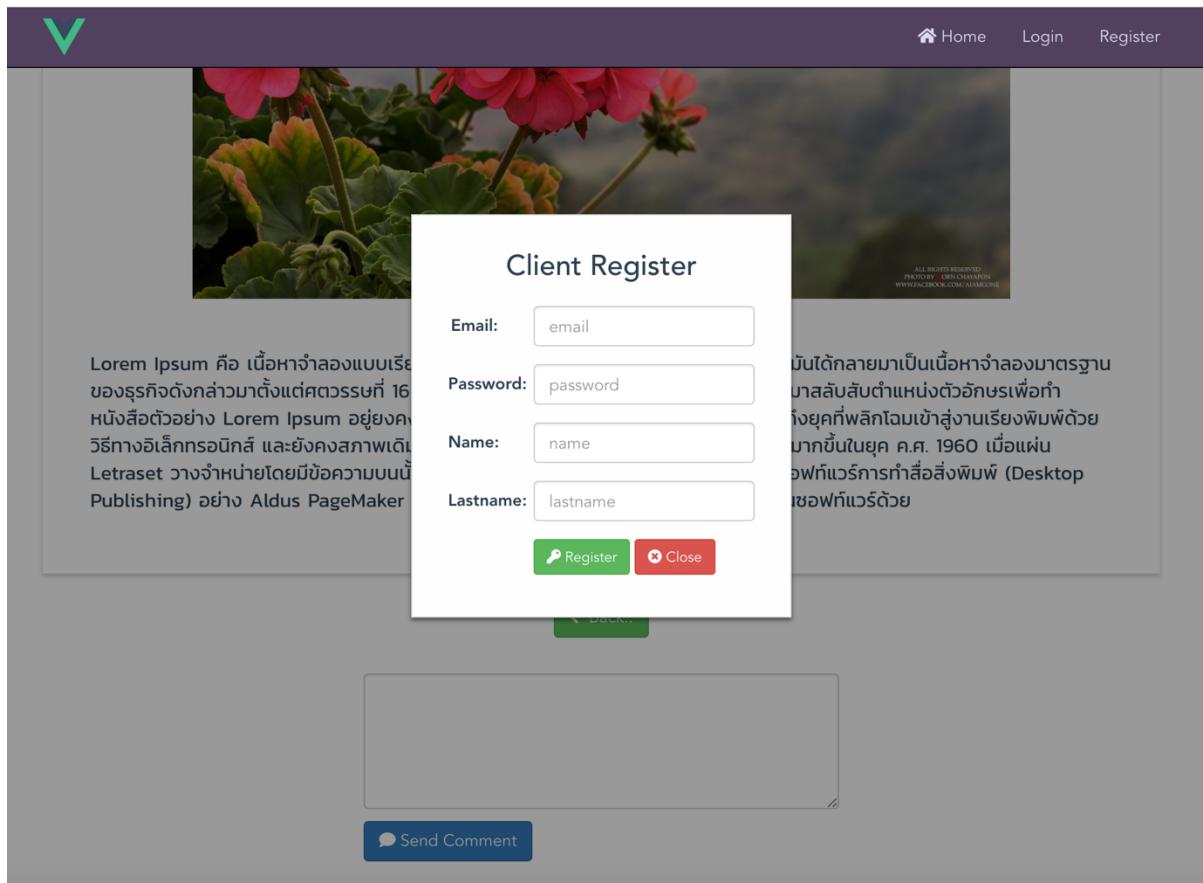
จากนั้นทำการเพิ่ม navbar link ของเราเพื่อทำการใช้งาน register form ของเรา
ดังนี้ครับ

```
<li v-if="!isUserLoggedIn" role="presentation"><a href="#" v-on:click.prevent="showRegister = true" >Register</a></li>
```

เมื่อทำการกดที่ Register จะแสดง Register Form ขึ้นมา เมื่อทำการ Register
เสร็จแล้วจะแสดง Pop

```
"Register successful, Please login first."
```

ให้ทำการ Login เพื่อเข้าระบบอีกครั้งนั่นเอง



→ սեղան թույլատրուն բառ User Register

เริ่มต้นทำส่วน comment

หลังจากเราทำการ login และ register ได้แล้ว เราจะทำการสร้างส่วนใช้งาน การ comment ของบทความกัน โดยในส่วนนี้เราจะได้ทำความเข้าใจ และใช้งาน vuejs กันมากขึ้น และเรายังสามารถนำความรู้ในบทนี้ กลับไปปรับปรุง เว็บบล็อกของ เราได้อีกด้วยครับ เพราะการสอนของผมนั้นสอนจากง่ายไปหากคุณ ฯ เพิ่มเติม ความรู้ไปเรื่อยๆ ครับ

จากบทที่ผ่านมาเราได้สร้าง api สำหรับจัดการ comment ของเราไว้หมดแล้ว ดังนั้น ที่เราจะต้องทำ คือ ส่วนของ front office เพียงส่วนเดียว

mapState ด้วย Vuex

จากเรื่อง Vuex ที่ผ่านมาทำให้เราสามารถใช้งาน ตัวแปร ผ่าน \$store ในทุก ๆ component ของเราได้ แต่เรามีวิธีที่ง่ายกว่านั้น คือ การ mapState

โดยการ

```
import {mapState} from 'vuex'
```

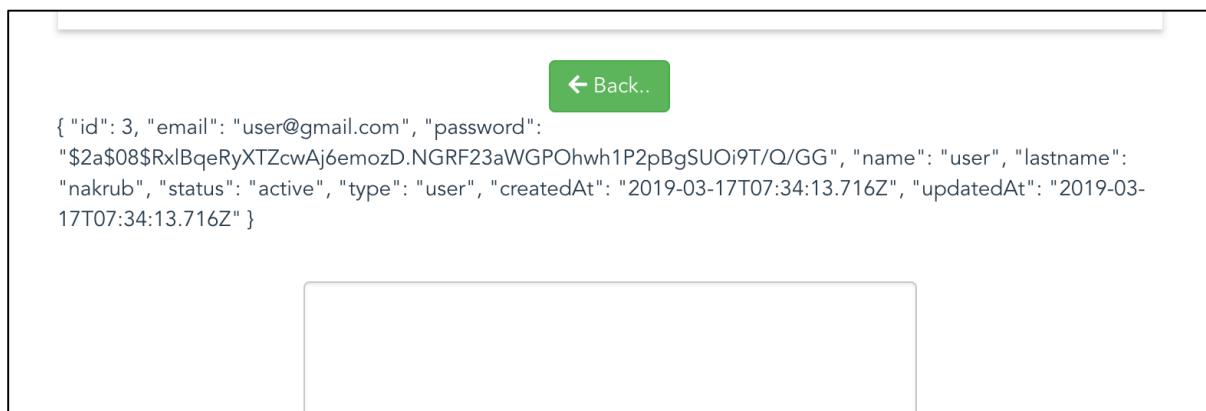
แล้วใส่ ตัวแปรที่ทำการ mapState ใน function mapState เอาไว้ที่ computed ดังนี้

```
computed: {
  ...mapState([
    'isUserLoggedIn',
    'user'
  ])
}
```

ผมทดสอบโดยการ ใส่

```
<p>{{user}}</p>
```

เนื้อ comment form ของเรา ผลที่ได้จะเป็นดังนี้



ผลที่ได้ คือ เราสามารถอ่านค่า user ได้เหมือน ตัวแปรหนึ่งใน data () ของเรา แต่ หากเราต้องการ set คือให้มันต้องทำการ dispatch เมื่อไบท์ที่ผ่าน ๆ มาจะ รับ ไม่สามารถ set ค่าให้ได้โดยตรงเหมือน state หรือ ตัวแปรใน data ของเรา

เพิ่มการค้นหา comment ใน api

เนื่องจาก comment ของเราอ้างอิงจาก blogId และ userId ดังนั้นเราจึงต้องทำการสร้าง api เพื่อใช้ในการค้นหา comment จาก blogId และ userId ตามลำดับ ทำการเปิด Comment Controller ของเราใน Server ขึ้นมา แล้วทำการเพิ่ม code เข้าไปดังนี้

```
// get comment by blog id
async blog (req, res) {
    try {
        const comment = await Comment.findAll({
            where: {
                blogId: req.params.blogId
            },
            order: [['updatedAt', 'DESC']]
        })

        res.send(comment)
    } catch (err) {
        res.status(500).send({
            error: 'The comment information was incorrect'
        })
    }
}

// get comment by user id
async user (req, res) {
    try {
        const comment = await Comment.findAll({
            where: {
                userId: req.params.userId
            },
            order: [['updatedAt', 'DESC']]
        })

        res.send(comment)
    } catch (err) {
        res.status(500).send({
            error: 'The comment information was incorrect'
        })
    }
}
```

เมื่อเราเพิ่ม function ใน Controller ของเราแล้ว เราต้องทำการ route mapping เพื่อให้ api ของเราสามารถเรียกใช้งาน function ใน controller ของเราได้ ทำการ เปิด routes.js ขึ้นมาแล้วทำการเพิ่ม code ดังนี้

```
// get comment by id
app.get('/comment/blog/:blogId',
  CommentController.blog
)

// get comment by id
app.get('/comment/user/:userId',
  CommentController.user
)
```

ตอนนี้ api บน server หรือ Back End ของเราพร้อมให้บริการดังกล่าวแล้ว

เพิ่ม Comment server การค้นหาโดย blogId และ userId

หลังจากที่เราสร้าง api บน server ไว้แล้ว เราจะมาทำการสร้าง Service เพื่อใช้งาน api ของเรา กัน โดยเราทำการเปิด comment service ของเราขึ้นแล้วทำการ เพิ่ม code เข้าไปดังนี้

```
blog (blogId) {
  return Api().get('comment/blog/' + blogId)
},
user (userId) {
  return Api().get('comment/user/' + userId)
}
```

สร้างส่วนของการ comment และการแสดงผลการ comment

เปิด src/components/Fronts/ShowBlog.vue เราจะทำส่วน comment แน่นอน เราต้องทำฟอร์มสำหรับ comment รวมถึงส่วนการแสดงผลการ comment

ผนเมื่ม form และส่วนการแสดง comment ดังนี้ครับ

```
<div class="comments-wrapper">
  <div class="comment-form-wrapper">
    <h4>Comments</h4>
    <form v-on:submit.prevent="sendComment">
      <p><textarea rows="5" class="form-control" v-model="comment"></textarea></p>
      <p v-if="user == null">Login / Register for commented.</p>
      <p v-else><button type="submit" class="btn btn-primary"><i class="fas fa-comment"></i> Send Comment</button></p>
    </form>
  </div>
  <transition-group tag="ul" name="fade" class="comment-list">
    <li v-for="comment in comments" :key="comment.id">
      <h4>user id: {{comment.userId}}</h4>
      <p>{{comment.comment}}</p>
    </li>
  </transition-group>
</div>
```

แล้วทำการเพิ่ม style ดังนี้

```
.comment-list {
  list-style: none;
  padding: 0;
  margin: 0;
  max-width: 400px;
  margin-left: auto;
  margin-right: auto;
}

.comment-form-wrapper {
  max-width: 400px;
  margin-left: auto;
  margin-right: auto;
  margin-top: 30px;
}

.comment-list li {
  border: solid 1px #dfdfdf;
  margin-bottom: 10px;
  padding: 10px;
  box-shadow: 0 2px 4px 0 rgba(0,0,0,.1);
  border-radius: 5px;
```

```
}
```

ทำการ Import Comment Service เพื่อใช้งานดังนี้

```
import CommentsService from '@/services/CommentsService'
```

จากนั้นเขียน function sendComment ไว้ใน methods ดังนี้

```
async sendComment () {
  // console.log(`comment: ${this.comment}`)
  try {
    let comment = {
      blogId:this.blog.id,
      comment:this.comment,
      userId:this.user.id
    }

    console.log(comment)
    await CommentsService.post(comment)
    this.comment = ''
    this.resultUpdated = "We are received"
    setTimeout(() => this.resultUpdated = '', 3000)
    this.reloadComment()
  } catch (err) {
    console.log(err)
  }
},
```

เพื่อทำการ Post comment ไปที่ server และเก็บไว้ใน database โดยสังเกตว่า user.id ของเรานี้มาจากการที่เรา mapStateToProps และดึงข้อมูลมาใช้งานนั่นเอง

ถึงตอนนี้เรารสามารถ comment blog ของเราได้แล้ว โดยผมทำการผูก comment ไว้ กับ blog id และ user id แต่เนื่องจาก เมื่อเราทำการ comment ไปแล้ว เรา ต้องการให้ผลการ comment ขึ้นเลยในทันที เราเลยต้องเขียนโปรแกรม เพื่อ reload comment ของเราดังนี้

โดยผมทำการสร้างตัวแปร comments ใน data () ของเรา ก่อน

```
comments: '',
```

และเขียน code เพื่อทำการ ดึงข้อมูลจาก back end ของเราดังนี้

```
async reloadComment () {
  try {
```

```
    this.comments = (await CommentsService.blog(this.blog.id)).data
} catch (error) {
  console.log(error)
}
}
```

ซึ่งจะไปเรียกใช้ function นี้ในตอนเริ่มต้นด้วยเพื่อตั้ง comment ที่มีกับ blogId นี้มาแสดงทั้งหมด ผ่านเพื่อ code ที่ created () ดังนี้

```
this.reloadComment()
```

มาถึงตอนนี้เราสามารถทำการ comment และแสดงผลการ comment ได้แล้ว แต่ ผ่านเพิ่มส่วน popup result เพื่อแสดงผลการทำงานต่าง ๆ เพื่อให้การรายงานสถานะชัดเจนขึ้นอีก โดย เพิ่มตัวแปร ไว้ใน data () ดังนี้

```
resultUpdated: '',
```

ซึ่งจริง ๆ แล้วใน code การ sendComment ของเรานั้นได้ทำการจัดการ เพิ่มสถานะให้เราเรียบร้อยแล้ว จากความรู้เรื่องการ login/logout เราได้รู้แล้วว่าเราจะ ทำยังไงกับ การแสดงสถานะลักษณะนี้ โดยผ่านเพิ่มส่วนการแสดงผลของเราไว้ใน template ดังนี้

```
<transition name="fade">
  <div v-if="resultUpdated != ''" class="popup-msg">
    <p>{{ resultUpdated }}</p>
  </div>
</transition>
```

ถึงตอนนี้เราทำการ Login และ comment บทความของเรามาได้แล้วครับ หากเราไม่ทำการ login ปุ่ม login จะไม่ขึ้น และแสดงข้อความให้ login ก่อนทำการ comment จาก code ชุดนี้ครับ

```
<p v-if="user == null">Login / Register for commented.</p>
  <p v-else><button type="submit" class="btn btn-primary"><i class="fas fa-comment"></i> Send Comment</button></p>
```

แยก Comment ออกจาก ShowBlog

ในตอนนี้ส่วนของการ comment ของเราจะอยู่รวมกับ ShowBlog Component ที่ Front Office ของเรา ซึ่งในอนาคตถ้าเวลานานไป จะแก้ไขส่วน comment เราจะหาไม่เจอ เพราะเราจะลืม

ทำการแยกส่วนนี้ออกมาเป็นอีก component หนึ่ง โดยทำการสร้าง src/components/Fronts/Comment.vue ขึ้นมา แล้วทำการเขียน code ดังนี้ ซึ่งจริง ๆ เราก็ตัดมาจาก ShowBlog component ของเราเอง

โดยผมทำการใส่ code ในส่วนของ template ดังนี้

```
<template>
  <div id="comment">
    <div class="comments-wrapper">
      <div class="comment-form-wrapper">
        <h4>Comments</h4>
        <form v-on:submit.prevent="sendComment">
          <p><textarea rows="5" class="form-control" v-model="comment"></textarea></p>
          <p v-if="user == null">Login / Register for commented.</p>
          <p v-else ><button type="submit" class="btn btn-primary"><i class="fas fa-comment"></i> Send Comment</button></p>
        </form>
      </div>
      <transition-group tag="ul" name="fade" class="comment-list">
        <li v-for="comment in comments" :key="comment.id">
          <h4>user id: {{comment.userId}}</h4>
          <p>{{comment.comment}}</p>
        </li>
      </transition-group>
    </div>
    <transition name="fade">
      <div v-if="resultUpdated != ''" class="popup-msg">
        <p>{{ resultUpdated }}</p>
      </div>
    </transition>
  </div>
</template>
```

แล้วเพิ่ม script ดังนี้

```
<script>
```

```
import CommentsService from '@/services/CommentsService'

export default {
  props:['blogid', 'user'],
  data () {
    return {
      comment: null,
      comments: '',
      resultUpdated: '',
    }
  },
  methods: {
    async reloadComment () {
      try {
        this.comments = (await CommentsService.blog(this.blogid)).data
      } catch (error) {
        console.log (error)
      }
    },
    async sendComment () {
      // console.log(`comment: ${this.comment}`)
      try {
        let comment = {
          blogId:this.blogid,
          comment:this.comment,
          userId:this.user.id
        }

        console.log(comment)
        await CommentsService.post(comment)
        this.comment = ''
        this.resultUpdated = "We are received"
        setTimeout(() => this.resultUpdated = '', 3000)
        this.reloadComment()
      } catch (err) {
        console.log(err)
      }
    },
    created () {
      this.reloadComment()
    }
  }
}</script>
```

จาก script ด้านบนจะเห็นว่า ผู้ทำการรับค่า props เข้ามา เพื่อจัดการแทน data () ในส่วนของ blogid และ user เพื่อนำค่าที่ส่งมาไปใช้งานต่อเนื่อง

ทำการใส่ style ดังนี้

```
<style scoped>
  .popup-msg {
    box-shadow: 0 2px 4px 0 rgba(0,0,0,.2);
    border: solid 1px #ddd;
    background: #fff;
    max-width: 200px;
    padding: 10px;
    position: fixed;
    bottom: 0;
    right: 0;
    border-radius: 5px;
    margin-bottom: 5px;
    margin-right: 5px;
  }

  .comment-list {
    list-style: none;
    padding: 0;
    margin: 0;
    max-width: 400px;
    margin-left: auto;
    margin-right: auto;
  }

  .comment-form-wrapper {
    max-width: 400px;
    margin-left: auto;
    margin-right: auto;
    margin-top: 30px;
  }

  .comment-list li {
    border: solid 1px #dfdfdf;
    margin-bottom: 10px;
    padding: 10px;
    box-shadow: 0 2px 4px 0 rgba(0,0,0,.1);
    border-radius: 5px;
  }
```

```
}
```

```
</style>
```

จากนั้นทำการเปิด src/components/Fronts>ShowBlog.vue ขึ้นมา

แล้วทำการแก้ไข code template เป็นดังนี้

```
<template>
  <div class="component-wrapper container">
    <main-header navsel="front"></main-header>
    <div v-if="blog" >
      <div class="hero-wrapper">
        <div class="hero">
          
          <h1>Webblot from nodejs + vuejs Ebook</h1>
          <p>By Gooddev.ME</p>
        </div>
      </div>
      <div class="blog-wrapper" v-if="blog != null">
        <h1>{{ blog.title }}</h1>
        <p><strong>Category: </strong>: <a href="#" v-on:click.prevent="navigateTo(`/front?search=${blog.category}`)">{{ blog.category }}</a></p>
        <div class="content" v-html="blog.content"></div>
        <!-- <p>category: {{ blog.category }}</p>
        <p>status: {{ blog.status }}</p> -->
      </div>
      <div class="back-nav"><button class="btn btn-success" v-on:click="navigateTo('/front')"><i class="fas fa-arrow-left"></i> Back..</button></div>
      <comment-comp v-bind:blogid="blog.id" v-bind:user="user"></comment-comp>
      <transition name="fade">
        <div v-if="resultUpdated != ''" class="popup-msg">
          <p>{{ resultUpdated }}</p>
        </div>
      </transition>
      <br>
    </div>
  </div>
</template>
```

จะเห็นว่าผมเรียกใช้ component comment ของเรา พร้อมทำการส่งค่าไปให้ props ของเราด้วย v-bind

```
<comment-comp v-bind:blogid="blog.id" v-bind:user="user" ></comment-comp>
```

จากนั้นทำการ แก้ไข script ของเราเป็นดังนี้

```
<script>
import {mapState} from 'vuex'
import BlogsService from '@/services/BlogsService'
import UsersService from '@/services/UsersService'
import CommentComp from '@/components/Fronts/Comment'

export default {
  data () {
    return {
      blog: null,
      users:null,
    }
  },
  components : {
    CommentComp
  },
  async created () {
    // get blog

    // check permission first
    try {
      let blogId = this.$route.params.blogId
      this.blog = (await BlogsService.show(blogId)).data
    } catch (error) {
      console.log (error)
    }
  },
  methods: {
    navigateTo (route) {
      this.$router.push(route)
    }
  },
  computed: {
    ...mapState([
      'isUserLoggedIn',
```

```

        'user'
    ])
}
}
</script>

```

สังเกตว่าเรา Import Comment.vue เข้ามาใช้งานในลักษณะของ component นั้นเอง จัดการแก้ไข style ที่ไม่ใช้ออกดังนี้

```

<style scoped>
.hero {
  margin-top: 80px;
  border-radius: 5px;
  background: darkcyan;
  height:250px;
  color:white;
  padding: 20px;
}

.hero h1 {
  margin-top: 30px;
}

.blog-wrapper {
  margin-top:20px;
  padding: 40px;
  box-shadow: 0 2px 4px 0 rgba(0,0,0,.2);
}
.back-nav {
  margin-top: 20px;
  text-align: center;
}
.blog-wrapper h1{
  text-align: center;
  font-family: 'kanit';
  padding-bottom: 50px;
}

.blog-wrapper p {
  font-family: 'kanit';
  font-size: 1.4em;
  padding-bottom:20px;
}

```

```

}

.blog-wrapper .content {
  font-family: 'kanit';
  font-size: 1.2em;
}
</style>
```

ในส่วนนี้เราจะสามารถทำการ comment และแสดงสถานะการ comment ได้เหมือนเดิม เพียงแต่เราแยก comment component ของเราออกมาเป็นสัดส่วนเพื่อให้ง่ายต่อการปรับปรุงหรือแก้ไขในอนาคตนั่นเอง

ทำส่วนแก้ไข และลบ comment

สร้าง api สำหรับให้ข้อมูล User ที่ Front Office

จากที่ผ่านเราได้ทำการโหลด comment ทั้งหมด ที่เกี่ยวกับ blog ที่เราอ่านได้แล้วจากการ แต่สังเกตว่า user ที่แสดงนั้น ยังคงเป็นเพียงแค่ userId อยู่ เราต้องทำการแสดงเป็นชื่อผู้ใช้งาน จึงจะถูกต้องและรู้ว่าใคร comment อะไร

ในการทำงานลักษณะนี้ สามารถทำได้หลายแบบ ทั้ง join table และ ทำการดึงค่าจากฐานข้อมูล ทุก ๆ comment ซึ่งจะต้องเสียเวลาในการดึงข้อมูลจาก server มาก รวมถึงกินทรัพยากรในส่วน client มากเช่นกัน

ดังนี้ผมจึงจะทำการดึงข้อมูลผู้ใช้งานมาเลยทั้งหมดในครั้งเดียวจะประหยัดเวลา และทรัพยากรมากกว่า แต่เนื่องจากการ Get All User ของเรานั้น ใช้กับ Back Office ซึ่งจะทำให้ User มาทั้งหมดรวมถึง ข้อมูลทั้งหมดของ User ทุกคนด้วย ซึ่งเราไม่ได้ต้องการให้เป็นอย่างนั้น ด้วยเหตุผลด้านความปลอดภัย เราจึงต้องเขียน api ส่วนนี้ขึ้นมา

ทำการเปิด UserController ของเราขึ้นมา และทำการเพิ่ม function ดังนี้

```

async getFront (req, res) {
  try {
    const users = await User.findAll()

    let listNames = []
    users.forEach(user => {
```

```

        let name = {
            "id":user.id,
            "name":`${user.name} ${user.lastname}`
        }
        listNames.push(name)
    })

    res.send(listNames)
} catch (err){
    res.status(500).send({
        error: 'The users information was incorrect'
    })
}
},

```

สังเกตว่าผมจะทำการ สร้าง json ใหม่ ส่งออกมา หรือเป็นการ filter เอาเฉพาะ id name และ lastname ส่งออกมาเท่านั้น

จากนั้นทำการเปิด routes.js ขึ้นมาแล้วทำการ mapping route กับ function ใน controller ของเรา

```

// users
// get front
app.get('/users/front',
    UserController.getFront
)

```

เมื่อเราทำการเพิ่ม api เราต้องไปทำการเพิ่ม service ในการติดต่อกับ api ของ เราที่ front end ของเราด้วย ทำการเปิด src/services/UsersService.js ขึ้นมาแล้วทำการเพิ่ม code ดังนี้

```

getFront () {
    return Api().get('users/front')
}

```

ตอนนี้เรามี api และ service สำหรับการดึงข้อมูล user ที่ Front Office แล้วนะ ครับ เราจะไปใช้งานกันในส่วนต่อ ๆ ไป

เริ่มต้นส่วน Edit และ Delete

วางแผนกันก่อนว่าจะทำการแก้ไขยังไงดี ถ้าทำเป็น form ใส่ไว้ใน comment นั้น คนจะไม่ได้ คงจะต้องมี form ในทุก ๆ comment ถ้าทำให้ redirect ไปแก้อีกหน้า เรา ก็คงไม่ใช้ vuejs กัน ดังนั้นเราจะแก้ไข ที่หน้านี้เลย โดยให้เงื่อนไขว่า ถ้าทำการ login เข้ามา จะสามารถทำการกดที่ comment เพื่อทำการแก้ไข comment ของตัวเองได้ สิ่งที่เราต้อง รู้คือ

1. เรากดที่ comment อันไหน และใช้ component ของเรารีอ่อน
2. จัดการสลับการแสดงผลระหว่าง comment และ form สำหรับแก้ไข

โดยเราจะทำการแยก edit component ออกจาก component นึง เพื่อจัดการโดยผ่านการทำการเขียน template ดังนี้

```
<template>
  <div id="edit-comment">
    <h4 v-if="user && compUser.id === user.id" class="comment-user">{{ compUser.name }}</h4>
    <h4 v-else>{{ compUser.name }}</h4>
    <hr>
    <transition name="fade">
      <p v-if="!editable" v-on:click.prevent="editComment(true, comment.id, compUser.id, user.id)">{{comment.comment}}</p>
    </transition>
    <transition name="fade">
      <div v-if="updateEditable" class="form-edit-wrapper">
        <form v-on:submit.prevent="updateComment">
          <p><textarea rows="5" class="form-control" v-model="comment.comment"></textarea></p>
          <p>
            <button type="submit" class="btn btn-warning btn-xs"><i class="fas fa-save"></i> Update</button>
            <button type="button" v-on:click.prevent="editComment(false, comment.id, compUser.id, user.id)" class="btn btn-success btn-xs"><i class="fas fa-times-circle"></i> Close</button>
          </p>
        </form>
      </div>
    </transition>
```

```

<p v-if="!editable && user != null && compUser.id === user.id"><button v-on:click="deleteComment(comment)" class="btn btn-xs btn-danger"><i class="fas fa-trash-alt"></i> Delete</button></p>

</div>
</template>

```

โดยใน Template ของ pmกำหนดให้ รับค่าเมื่อมีการ click ใน comment ของ pm และไปทำงานที่ function editComment โดยส่ง id ของ comment ไปด้วย

จากนั้นทำการเพิ่ม script ดังนี้

```

<script>
import CommentsService from '@/services/CommentsService'

export default {
  props:['comment', 'users', 'user', 'editable'],
  data () {
    return {
      // userName: null,
      compUser:[]
    }
  },
  created () {
    this.getUser()
  },
  methods: {
    getUser () {
      this.users.forEach(user => {
        if (user.id == this.comment.userId) {
          // this.userName = user.name
          this.compUser = user
        }
      })
      // console.log('users')
      // console.log(this.users)
    },
    editComment (status, commentId, compId, userId) {
      // check permission first
      if (this.user != null) {
        if(compId == userId) {
          if (status) {
            this.$emit('editable-update', commentId)
          } else {
        }
      }
    }
  }
}

```

```

        this.$emit('editable-close')
    }
}
},
async updateComment () {
try {
    await CommentsService.put(this.comment)
    this.editable = false
    let updateResult = "updated"
    this.$emit('editable-close')
    this.$emit('comment-part', updateResult)
} catch (err) {
    console.log(err)
    let updateResult = "error"
    this.$emit('comment-part', deleteResult)
}
},
async deleteComment (comment) {

let result = confirm("Want to delete?")
if (result) {
    try {
        await CommentsService.delete(comment)
        let deleteResult = "deleted"
        this.$emit('comment-part', deleteResult)
    } catch (err) {
        console.log(err)
        let deleteResult = "error"
        this.$emit('comment-part', deleteResult)
    }
}

let deleteResult = "deleted"
this.$emit('comment-part', deleteResult)
}
},
computed: {
updateEditable () {
    return this.editable
}
}
}
</script>

```

โดย script ของ pm ในส่วนนี้จะมีส่วนที่หน้าสนใจ คือ

```
props: ['comment', 'users', 'user', 'editable'],
```

เป็นการรับค่าผ่าน props เข้ามา จะสังเกตเห็นว่า pm รับ editable ของมาจัดการสถานะของการแสดงผล form edit comment ของเรา นั่นคือ เราจัดการสถานะจากภายนอก component นั้นเอง

แล้วเมื่อมีการ click เกิดขึ้น ที่

```
editComment (status, commentId) {
  // check permission first
  if (this.user != null) {
    if (status) {
      this.$emit('editable-update', commentId)
    } else {
      this.$emit('editable-close')
    }
  }
},
```

เราจะส่งกลับไปบอก component แม่ หรือ component ที่เรียกใช้งานของเราด้วย

```
this.$emit('editable-update', commentId)
```

ซึ่งใน component แม่เวลาเรียกใช้เราจะเรียกใช้ดังนี้

```
<edit-comment
  :editable="editState.find(status      =>      status.id      ===
comment.id).status"
  :comment="comment"
  v-on:comment-part="updatedResult"
  v-on:editable-update="updateEditStatus"
  v-on:editable-close="closeEditor"
  :user="user"
  :users="users"
></edit-comment>
```

สังเกตที่คำว่า 'editable-update' นะครับ มันคือ tag ที่เราใช้ส่งกลับมาแล้ววิ่งกลับไปทำ function ไหนนั่นเอง ในที่นี่คือ "updateEditStatus" ของเราเอง

โดย pm เขียน function ไว้ดังนี้

```
updateEditStatus (commentId) {
  console.log("state update: " + commentId)
  this.editState.map((mState) => {
    if(mState.id === commentId) {
```

```

        mState.status = true
    } else {
        mState.status = false
    }
}
},

```

เพื่อจัดการ การตอบกลับ function ของเรา รวมถึงถ้าดูดี ๆ เราจะมีส่วนที่ตรวจสอบว่า id ของ user ที่ list มากันนั้น ใช้ id ของเราหรือไม่ เพื่อทำการยินยอมให้แก่ไข comment ของเราได้นั่นเอง

ทำการเพิ่ม style เพื่อแยก user เรากลางจาก user คนอื่นดังนี้

```

<style scoped>
.comment-user {
    background:#888;
    color:white;
    border-radius:4px;
}
h4{
    padding: 4px;
}
</style>

```

code ในส่วน Comment.vue ที่ผ่านมาเป็นเพียงการอธิบายนะครับ

เรามาแก้ code ของเราในไฟล์ Comment.vue ตั้งนี้ครับ เริ่มจาก Code ในส่วนของ <template> ก่อนครับ

```

<template>
    <div id="comment">
        <div class="comments-wrapper">
            <div class="comment-form-wrapper">
                <h4>Comments</h4>
                <form v-on:submit.prevent="sendComment">
                    <p><textarea rows="5" class="form-control" v-model="comment"></textarea></p>
                    <p v-if="user == null">Login / Register for commented.</p>
                    <p v-else ><button type="submit" class="btn btn-primary"><i class="fas fa-comment"></i> Send Comment</button></p>

```

```

        </form>
    </div>
    <transition-group tag="ul" name="fade" class="comment-list">
        <li v-for="comment in comments" :key="comment.id">
            <edit-comment
                :editable="editState.find(status => status.id === comment.id).status"
                :comment="comment"
                v-on:comment-part="updatedResult"
                v-on:editable-update="updateEditStatus"
                v-on:editable-close="closeEditor"
                :user="user"
                :users="users"
            ></edit-comment>
        </li>
    </transition-group>
</div>
<transition name="fade">
    <div v-if="resultUpdated != ''" class="popup-msg">
        <p>{{ resultUpdated }}</p>
    </div>
</transition>
</div>
</template>

```

ลังเกตว่ามีการเรียกใช้ component ย่อๆออกໄປ พร้อมส่งค่าໄປให้ props ของมัน ด้วย เราเขียน script ดังนี้ครับ

```

<script>
import CommentsService from '@/services/CommentsService'
import EditComment from '@/components/Fronts/EditComment'
import UsersService from '@/services/UsersService'

export default {
    props:['blogid', 'user'],
    data () {
        return {
            comment: null,
            comments: '',
            resultUpdated: '',
            editState: []
        }
    },
    components: {

```

```

    EditComment
},
methods: {
  closeEditor(){
    this.editState.map((mState) => {mState.status = false})
  },
  updateEditStatus (commentId) {
    console.log("state update: " + commentId)
    this.editState.map((mState) => {
      if(mState.id === commentId) {
        mState.status = true
      } else {
        mState.status = false
      }
    })
  },
  async reloadComment () {
    try {
      this.comments = (await CommentsService.blog(this.blogid)).data
      this.comments.map((comment) =>
{this.editState.push({id:comment.id, status:false})})
    } catch (error) {
      console.log (error)
    }
  },
  async sendComment () {
    // console.log(`comment: ${this.comment}`)
    try {
      let comment = {
        blogId:this.blogid,
        comment:this.comment,
        userId:this.user.id
      }

      console.log(comment)
      await CommentsService.post(comment)
      this.comment = ''
      this.resultUpdated = "We are received"
      setTimeout(() => this.resultUpdated = '', 3000)
      this.reloadComment()
    } catch (err) {
      console.log(err)
    }
  },
}

```

```

updatedResult (result) {
    // console.log('Hello result:')
    // console.log(result)
    // console.log(result)

    if (result === "updated") {
        this.resultUpdated = "Updated successful."
        this.reloadComment()
    } else if (result === "deleted") {
        this.resultUpdated = "Deleted successful."
        this.reloadComment()
    } else {
        this.resultUpdated = "System have some error."
    }

    setTimeout(() => this.resultUpdated = '', 3000)
},
},
async created () {
// get all users
try {
    this.users = (await UsersService.getFront()).data
    // console.log(this.users)
} catch (error) {
    console.log (error)
}

this.reloadComment()
}
}
</script>

```

ใน script ของเราจะมีส่วนการตอบรับ สถานะการ edit form ของเรามาตามที่ได้ อธิบายไปแล้ว โดยผูกกำหนดสถานะเริ่มของ editor ของเราจาก

```

async reloadComment () {
    try {
        this.comments = (await CommentsService.blog(this.blogid)).data
        this.comments.map((comment) =>
{this.editState.push({id:comment.id, status:false})})
    } catch (error) {
        console.log (error)
    }
},

```

โดย .map จะทำงานเหมือน forEach นั้นเองครับ

และการส่งค่าไป พร้อมกับการเรียกใช้ component นั้นผมก็ใช้ .map เช่นกันในการค้นหาและส่งค่าสถานะที่ตรงกับ id เข้าไปครับ

```
:editable="editState.find(status => status.id === comment.id).status"
```

ในทางกลับกัน เมื่อ Edit Comment Component ของผม emit กลับมา ผมก็ไปจัดการทุก Edit State ให้กลายเป็น false เพื่อปิดการแก้ไขครับ

จัดการแก้ไข style ของเราเป็นดังนี้

```
<style scoped>
.blog-wrapper {
  margin-top: 80px;
}

/* thumbnail */
.thumbnail-pic img{
  width: 200px;
  margin-bottom: 10px;
}

/* display uploaded pic */
ul.pictures {
  list-style: none;
  padding: 0;
  margin: 0;
  float: left;
  padding-top: 10px;
  padding-bottom: 10px;
}

ul.pictures li {
  float: left;
}

ul.pictures li img {
  max-width: 180px;
  margin-right: 20px;
}

.clearfix {
  clear: both;
}
```

```

/* uplaod */
.dropbox {
    outline: 2px dashed grey; /* the dash box */
    outline-offset: -10px;
    background:lemonchiffon;
    color: dimgray;
    padding: 10px 10px;
    min-height: 200px; /* minimum height */
    position: relative;
    cursor: pointer;
}

.input-file {
    opacity: 0; /* invisible but it's there! */
    width: 100%;
    height: 200px;
    position: absolute;
    cursor: pointer;
}

.dropbox:hover {
    background:khaki; /* when mouse over to the drop zone, change color */
}

.dropbox p {
    font-size: 1.2em;
    text-align: center;
    padding: 50px 0;
}
</style>

```

มาถึงตอนนี้เรารสามารถ edit และ delete ของเราได้อย่างดีแล้วนะครับ เว็บบล็อกของเรามี function ต่าง ๆ สมบูรณ์หมดเกือบหมดแล้ว

บทที่ 17 เก็บงานทุกอย่างหลังบ้าน

ทำการเปิด src/components/Comments/Index.vue ที่ front end ของเรารีบบ์มาแล้วทำการแก้ไข ปุ่มดู blog ของเราให้วิ่งไปที่ blog ของเราดังนี้

```
<button class="btn btn-sm btn-info" v-on:click="navigateTo('/front/read/' + comment.blogId)">ดูล็อกที่ Comment</button>
```

มาถึงตอนนี้จะเห็นว่า จากนั้นให้ src/components/Fronts/EditComment ขึ้นมา แล้วเพิ่ม code ต่อจาก ปุ่ม delete ของเราเข้าไป

```
<p v-if="user.type === 'admin'"><button v-on:click="deleteComment(comment)" class="btn btn-xs btn-danger"><i class="fas fa-trash-alt"></i> Delete</button></p>
```

เพื่อให้ admin สามารถลบได้ ทุก comment นั่นเอง

จัดการสถานะ pause และ active ผู้ใช้งาน

ทำการเปิดไฟล์ src/components/Users/Index.vue หรือ User Index Component ของเราขึ้นมา แล้วทำการเพิ่มปุ่มสำหรับ pause และ active user ของเราเข้าไปใน <template></template>

```
<p>
    <a class="btn btn-danger btn-sm" href="#" v-on:click.prevent="pauseUser(user.id)"><i class="fas fa-pause"></i> Pause</a>&nbsp;
    <a class="btn btn-success btn-sm" href="#" v-on:click.prevent="activeUser(user.id)"><i class="fas fa-check"></i> Active</a>&nbsp;
</p>
```

จากนั้นใส่ function ของเราเข้าไปใน script

```
async pauseUser (userId) {
  let user = {
    "id": userId,
    "status": "pause"
```

```

    }

    try {
        await UsersService.put(user)
        this.refreshData()
    } catch (error) {
        console.log(error)
    }
},
async activeUser (userId) {
    let user = {
        "id": userId,
        "status":"active"
    }

    try {
        await UsersService.put(user)
        this.refreshData()
    } catch (error) {
        console.log(error)
    }
},

```

ง่ายมากเลย คือ การ update status นั้นเองครับ จากนั้นเปิด UserAuthenController ใน server ของเราขึ้นมาแล้วทำการใส่ code ทั้งดังนี้ ทั้ง function login() และ clientLogin() เพื่อทำการตอบกลับเป็น error สำหรับ user ที่ถูกระงับการใช้งาน

```

if(user.status != "active") {
    return res.status(403).send({
        error: 'Your account suspend.'
    })
}

```

ทำการทดสอบ จะเห็นว่า user ที่เราทำการ pause ไว้ จะไม่สามารถทำการ login เข้าระบบได้นั่นเอง

ชื่อ-นามสกุล: user1 - nakrub
email: user1@gmail.com
สร้างเมื่อ: 2019-03-17T07:35:05.241Z

ระดับการใช้งาน: user
สถานะ: pause

View User Edit User Delete

Pause Active

ชื่อ-นามสกุล: user3 - user3lastname

testing comment

Delete

use

Hello

Delete

kor

Delete

Client Login

Email:

Password:

Login Close

Your account suspend.

testing comment

Delete

ทำการเพิ่มการกำหนดสถานะการเผยแพร่ของ blog

ในส่วนนี้เราจะทำการสร้างสถานะการเผยแพร่กันนะครับ เพราะเนื่องจากเมื่อเราสร้างหรือเขียน blog ของเราขึ้นมา อาจจะเขียนยังไม่เสร็จ หรือต้องการปรับปรุงยังไม่พร้อมเผยแพร่ เราสามารถจัดการได้ดังนี้ครับ

ทำการเปิด src/components/Blogs/Index.vue ของเรารวมมาแล้วทำการใส่ปุ่มใน <template></template> เพื่อจัดการการเผยแพร่ blog ของเราดังนี้

```
<p>
    <a class="btn btn-danger btn-sm" href="#" v-on:click.prevent="suspend(blog.id)"><i class="fas fa-pause"></i> Suspend</a>&nbsp;
    <a class="btn btn-success btn-sm" href="#" v-on:click.prevent="publish(blog.id)"><i class="fas fa-check"></i> Published</a>&nbsp;
</p>
```

จากนั้นทำการเพิ่ม function ใน methods ของเราดังนี้ครับ

```
async suspend (blogId) {
  let user = {
    "id": blogId,
    "status": "saved"
  }

  try {
    await BlogsService.put(user)
    this.refreshData()
  } catch (error) {
    console.log(error)
  }
},
async publish (blogId) {
  let user = {
    "id": blogId,
    "status": "published"
  }

  try {
    await BlogsService.put(user)
    this.refreshData()
  } catch (error) {
    console.log(error)
  }
}
```

```
        }
    },
```

ตอนนี้เรารสามารถกำหนดสถานะการเผยแพร่ของ blog ที่เราเขียนขึ้นได้แล้ว แต่ส่วน back end ของเราที่ยังไม่ได้ทำการ filter blog ของเราสู่ client โดย เราต้องทำการเพิ่ม function ที่ Blog Controller ใน Server ของเราดังนี้

```
async frontIndex (req, res) {
  try {
    let blogs = null
    const search = req.query.search
    console.log('-----> search key: ' + search)

    if (search) {
      blogs = await Blog.findAll({
        where: {
          $or: [
            'title', 'content', 'category'
          ].map(key => ({
            [key]: {
              $like: `%${search}%`,
            }
          })),
          $and: [
            {
              status: {
                $eq: "published"
              }
            },
            {
              order: [['createdAt', 'DESC']]
            }
          ]
        },
        else {
          blogs = await Blog.findAll({
            where: {
              'status': 'published'
            },
            order: [['createdAt', 'DESC']]
          })
        }
      })
      res.send(blogs)
    } catch (err) {
```

```
        res.status(500).send({
            error: 'an error has occurred trying to fetch the blogs'
        })
    },
},
```

แล้วทำการ route mapping ให้ api ของเรา เรียกใช้ function frontIndex() ที่เราสร้างขึ้นดังนี้

เปิด routes.js ขึ้นมาแล้ว เพิ่ม route data เข้าไปดังนี้ครับ

```
app.get('/blogs/front',
    BlogController.frontIndex
)
```

จะสังเกตว่าเพิ่ม frontIndex เข้าไป แทนที่จะแก้ไขจาก index function เดิมที่มีอยู่ เพราะว่าหากเราแก้ไข function เดิมที่มีอยู่แล้วนั้น จะทำให้มีการ suspend blog ที่เราเขียน จะทำให้ Back Office ของเราไม่สามารถดำเนินการได้ เราจึงต้องทำ function แยกอีกมาให้ Front Office ของเราเรียกใช้ จากนั้นทำการเพิ่ม frontIndex ใน Blog Service ของเราดังนี้

```
frontIndex (search) {
    return Api().get('blogs/front', {
        params: {
            search: search
        }
    },
},
```

จากนั้นทำการเปิด src/components/Fronts/Index.vue ขึ้นมาแล้วการเปลี่ยนไปใช้ frontIndex แทน

```
this.results = (await BlogsService.frontIndex(value)).data
```

ลองทำการทดสอบดู จะเห็นว่าเมื่อเราทำการ suspend blog ของเราแล้วจะไม่ไปแสดงที่หน้า Front Office นั่นเอง

ส่วนจัดการบล็อก

Search

จำนวน blog: 7



Blog 7 ของเรา

Category: html

Create: 2019-03-17T18:48:32.394Z
Status: saved

localhost:8080/front

Home Login Register

Blog 6 ของเรา

Category: vuejs

Create: 2019-03-17T18:46:53.343Z

Blog 5 ของเรา

Category: nodejs

Create: 2019-03-17T18:45:10.805Z

จัดการ Format ของการแสดงวันที่

เราจะใช้ package ที่ชื่อ moment ในการจัดการเปลี่ยน format ของวันที่เป็น แบบ DD-MM-YYYY กันครับ โดยการติดตั้ง moment ด้วย

```
npm install --save moment
```

เวลาเราจะใช้งานเราก็เปิด Vue Component ที่เราจะใช้ และทำการ ใส่ Import package เพื่อใช้งาน และ สร้าง filter ดังนี้

```
import moment from 'moment'
```

```
filters: {
    formatedDate (value) {
        return moment(String(value)).format('DD-MM-YYYY')
    },
},
```

เวลาที่เราเรียกใช้ใน <template></template> ของเรา เราสามารถเรียกใช้ได้ ดังนี้ครับ

```
{{ blog.createdAt | formatedDate }}
```

ตรงไหนที่เป็นวันที่ เราก็ใช้เทคโนโลยีได้ทั้งหมดเลยครับ ลองกลับไปเก็บส่วนที่เกี่ยวกับ วันที่ทั้งหมดดูครับ

จัดการลำดับการเรียงสถานะของแต่ละหมวด

การดึงข้อมูลจาก api ของเราในแต่ละ controller นั้น ผมใช้เป็น dsec จาก updatedAt ทั้งหมด ซึ่งส่งผลให้มีเวลา updated จะเปลี่ยนแปลงทันที ทำให้สับสน ในการใช้งานได้ง่าย โดยผมจะจัดใหม่ โดยเปิด UserController.js ใน Server ขึ้นมา และทำการแก้ไข index function ของเราใหม่ดังนี้

```
if (search) {
    users = await User.findAll({
        where: {
            $or: [
```

```

        'name', 'lastname', 'email'
    ].map(key => ({
        [key]: {
            $like: `%${search}%`,
        }
    })),
},
order: [['createdAt', 'ASC']]
})
} else {
users = await User.findAll({
    order: [['createdAt', 'ASC']]
})
}

```

โดยสังเกตว่า ผู้อ้างอิงจากวันที่สร้างและเรียงจาก ก่อนไปหาหลังนั้นเอง

ทำการเปิด BlogController และ CommentController ขึ้นมา และทำการแก้ไขให้ เรียงจากความใหม่ในการสร้างดังนี้

```

if (search) {
blogs = await Blog.findAll({
where: {
$or: [
'title', 'content', 'category'
].map(key => ({
[key]: {
$like: `%${search}%`,
}
})),
},
order: [['createdAt', 'DESC']]
})
} else {
blogs = await Blog.findAll({
order: [['createdAt', 'DESC']]
})
}

```

ตอนนี้ระบบการจัดเรียงของเราไม่สับสนแล้วครับ

ทำการปิด Permission ที่หลังบ้านทั้งหมด โดยเปิด routes.js ที่ server แล้วทำการใส่ code เพื่อทำการปิด การเข้าใช้งานแบบ public

```
// get all user
app.get('/users',
  isAuthenController,
  UserController.index
)
```

โดยเช็ค list ที่ผู้มีอำนาจได้กำหนดนี้คับ

จาก sqlite สู่ mysql

```
app.post("/upload",
app.post('/upload/delete',
app.post('/blog',
app.put('/blog/:blogId',
app.delete('/blog/:blogId',
app.get('/blogs',
app.delete('/user/:userId',
app.get('/users',
```

ดังนั้น path ใน check list ของผู้มีอำนาจจะไม่สามารถเข้าถึงแบบ public ได้ ต้องทำการ login ก่อนเท่านั้น

จัดการ Theme Login Form โดยผู้มีอำนาจทำการใส่ Style ให้กับ Form Login ของผู้มีอำนาจ เปิด src/components/Login.Vue ขึ้นมาแล้วทำการแก้ไข source code ทั้งหมดดังนี้

```
<template>
<div>
  <div class="user-wrapper">
    <div class="container">
      <div class="row">
        <div class="con-md-12">
          <div class="login-wrapper">
            <h1>Login</h1>
```

```

        <form      v-on:submit.prevent="onLogin"      class="form-
horizontal">
            <div class="form-group">
                <label for="" class="control-label col-md-3">Username:</label>
                <div class="col-md-8">
                    <input type="text" v-model="email" class="form-
control" />
                </div>
            </div>
            <div class="form-group">
                <label for="" class="control-label col-md-3">Password:</label>
                <div class="col-md-8">
                    <input      type="password"      v-model="password"
class="form-control">
                </div>
            </div>
            <div class="form-group">
                <div class="col-md-offset-3 col-md-8">
                    <button class="btn btn-default" type="submit"><i
class="fas fa-sign-in-alt"></i> Sign In</button>
                    <p class="error" v-if="error">{{error}}</p>
                </div>
            </div>
            <!-- <p>username: <input type="text" v-model="email"
/></p>
            <p>password:      <input      type="password"      v-
model="password"></p> -->
            <!-- <p><button type="submit">Sign In</button></p> -->
        </form>
        <!-- <div class="error" v-if="error">{{error}}</div> -->
    </div>
    </div>
</div>
</div>
</template>
<script>

import AuthService from '@/services/AuthService'

export default {

```

```

data () {
  return {
    email: '',
    password: '',
    error: null
  }
},
methods: {
  async onLogin () {
    try {
      const response = await AuthenService.login({
        email: this.email,
        password: this.password
      })

      this.$store.dispatch('setToken', response.data.token)
      this.$store.dispatch('setUser', response.data.user)

      this.$router.push({
        name: 'blogs'
      })
      console.log(response.data)

    } catch (error) {
      console.log(error)
      this.error = error.response.data.error
      this.email = ''
      this.password = ''
    }
  }
}
</script>
<style scoped>
.error {
  color:red;
  margin-top: 10px;
}

.login-wrapper {
  /* border: solid 1px red; */
  width: 400px;
  margin-left: auto;
  margin-right: auto;
}

```

```

margin-top: 80px;
box-shadow: 0 2px 4px 0 rgba(0, 0, 0, 0.30);
border: 1px solid #f3f3f3;
border-radius: 3px;
padding: 20px;
}

.login-wrapper h1 {
  text-align: center;
  padding-bottom: 20px;
  margin-top: 0px;
  font-family: Kanit;
}

```

ซึ่งจริงๆ แล้วไม่มีอะไรเปลี่ยนแปลงนอกจากเรียกใช้งาน bootstrap และเพิ่ม style ในการจัดการหน้าตาของมันให้สวยงามขึ้นเท่านั้นเองครับ

มาถึงตอนนี้ผมเพิ่งสังเกตว่า BackHeader Component ของผมยังมีปัญหาเรื่อง การ login/logout อญี่ ผมเลยทำการแก้ไข code เป็นดังนี้ครับ

```
<li    v-if="isUserLoggedIn"      role="presentation"><a    href="#"    v-
on:click.prevent="logout">Logout</a></li>
```

จากนั้นทำการ ใช้ vuex ในการ mapState เพื่อใช้งาน user ของเรา

```
import {mapState} from 'vuex'
```

```

computed: {
  ...mapState([
    'isUserLoggedIn',
    'user'
  ]),
}
```

ทำการตรวจสอบ User Login พร้อมแสดงชื่อผู้ใช้งาน

```
<transition name="fade">
<li          v-if="isUserLoggedIn"      role="presentation"><a
href="#">{{user.name}}</a></li>
</transition>
```

มาถึงตอนนี้ หากเราไม่ทำการ login จะไม่สามารถโหลดข้อมูลทั้ง user และ blog ได้แล้ว เราทำการ logout แล้ว เรียกไปที่หน้า <http://localhost:8080/blogs> จะเห็นว่าไม่มีข้อมูลส่งมา

The screenshot shows a web browser window with the URL localhost:8080/blogs. The page title is "ส่วนจัดการบล็อก". It features a search bar, a green "Create blog" button, and a red "Clear" button. Below these buttons, a message in an orange box says "*** ไม่มีข้อมูล ***". At the bottom of the page, there is a developer tools Network tab. The Network tab shows a timeline from 10000 ms to 70000 ms. Under the "Name" column, there are several entries: "websocket", "favicon.ico", "992a26c13c8bf6070ad4.hot-update.json", "0.992a26c13c8bf6070ad4.hot-update.js", "blogs", and "blogs". The "Preview" column for the "blogs" entry shows the JSON response: {error: "you do not have access to this resource"} and error: "you do not have access to this resource".

รวมถึงหน้า users ก็จะไม่มีข้อมูลส่งมาเช่นกัน

localhost:8080/blogs

Blogs Users Comments Login

ส่วนจัดการบล็อก

Search

จำนวน blog: 0

*** ไม่มีข้อมูล ***

Network

Name	Headers	Preview	Response	Timing
websocket				
favicon.ico				
992a26c13c8bf6070ad4.hot-update.json				
0.992a26c13c8bf6070ad4.hot-update.js				
blogs			▼ {error: "you do not have access to this resource"} error: "you do not have access to this resource"	
blogs				

ตอนนี้หลังบ้านของเรากูกปิดในส่วนที่สำคัญหมวดแล้วนั่นเอง
ต่อมาเราจะจัดการปิดการใช้งานในส่วนของ Front End ทั้งหมดกันนะครับ
เราทำการปิดการใช้งานด้วย code ชุดนี้ครับ

```
import {mapState} from 'vuex'
```

```
mounted () {
  if (!this.isUserLoggedIn) {
    this.$router.push({
      name: 'login'
    })
  }
},
```

```
computed: {
  ...mapState([
    'isUserLoggedIn',
    'user'
  ]),
}
```

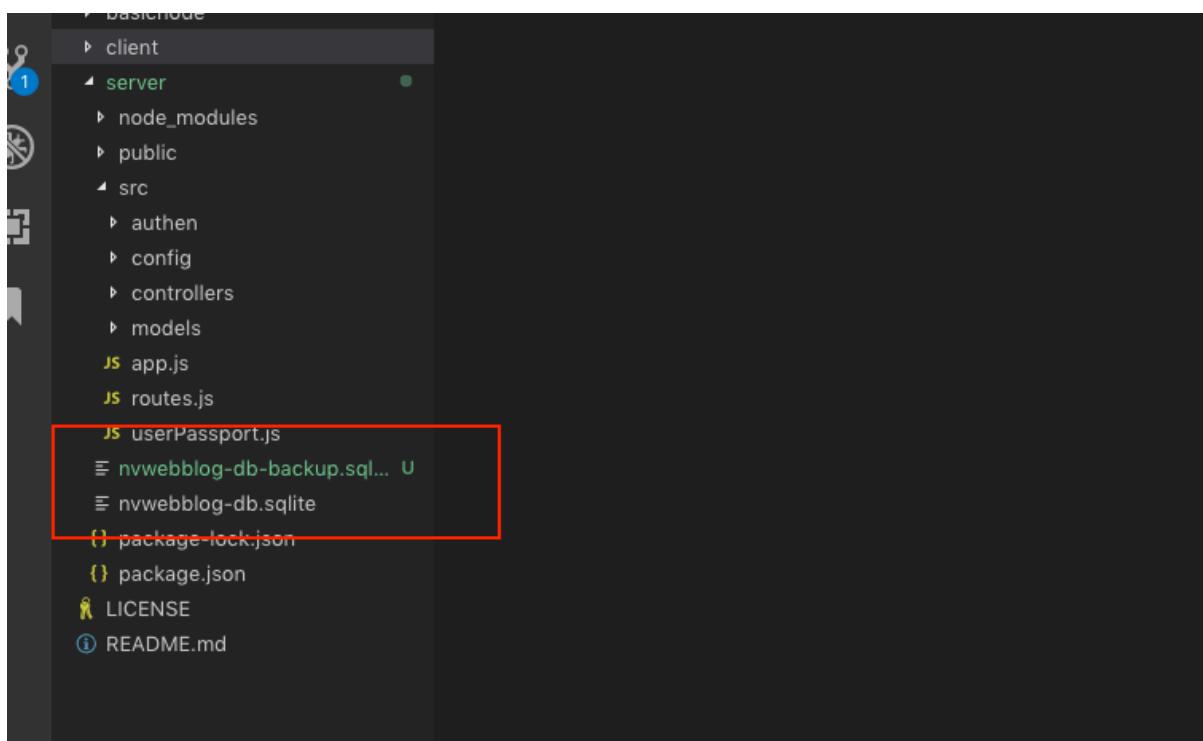
นี่คือเมื่อไหร่ที่ทำงานในส่วน cycle ของ mounted จะทำการตรวจสอบสถานะว่า login ผ่านมาหรือยัง ถ้ายังให้วิ่งไปที่หน้า login นั่นเอง

ทำการแปะ code ใน vue component ในส่วน Back Office ทั้งหมด หลังจากแปะ code ในการตรวจสอบแล้ว ตอนนี้ระบบของเราจะปลอดภัยทั้งหน้าบ้านและหลังบ้านแล้วครับ มาถึงตอนนี้ เว็บบล็อกของเราสำเร็จแล้วนั่นเองครับ ส่วน option หรือลูกเล่นอื่น ๆ ผมจะไม่กล่าวถึงนะครับ เพราะมาถึงตรงนี้ ผมเชื่อว่าทุกท่านสามารถทำเองได้เกือบหมดแล้ว หากติดขัดตรงไหน สามารถสอบถามกับผมได้ในกลุ่มเลยครับ

บทที่ 18 จาก SQLite สู่ MySQL

มาถึงตอนนี้เราจะใช้งานฐานข้อมูล MySQL กันนะครับ อาจจะไม่สอนติดตั้งนะครับ เพราะน่าจะติดตั้งกันเป็นทุกคนอยู่แล้ว ถ้าไม่เป็น ค้นใน google มีข้อมูลมากมาย ครบทุก platform เลยครับ

เขามาเป็นว่าตอนนี้เรามี MySQL ในเครื่องของเราแล้วนะครับ อย่างแรกจะทำอะไร ก็ backup กันให้ดีเสียก่อน ผมทำการ backup โดยการ copy ไฟล์ database ของ เราไปวางไว้ที่อื่นอีกไฟล์นึงก่อนครับ



จากนั้นทำการไปที่ website นี้ครับ <https://www.rebasedata.com/convert-sqlite-to-mysql-online> และทำการ browse ไฟล์ database ของเราขึ้นไป จากนั้น มีงโก้ ได้แล้ว MySQL Format ของเราแล้ว download ลงมา และทำการ Import เข้า MySQL ของเรา และเปิดดูด้วย MySQL Workbench ของเรา หรือ phpmyadmin ครับใช้ตัวอื่นก็ว่ากันไปครับ ผมใช้ตัวนี้ครับ

TABLE INFORMATION

- created: 19/3/2562 BE
- updated: 19/3/2562 BE
- engine: InnoDB
- rows: 7
- size: 48.0 KiB
- encoding: utf8mb4

7 rows in table; 1 row selected

มาแล้วครับ ข้อมูลของเรา จากนั้นทำการติดตั้ง package ที่ชื่อว่า mysql2 ที่ server ของเรา ด้วยคำสั่ง

```
npm install --save mysql2
```

จากนั้นเปิด config/config.js ใน server ของเราขึ้นมา ทำการแก้ connection data ให้ตรงกับ mysql ของเรา ของผมเป็นดังนี้ครับ

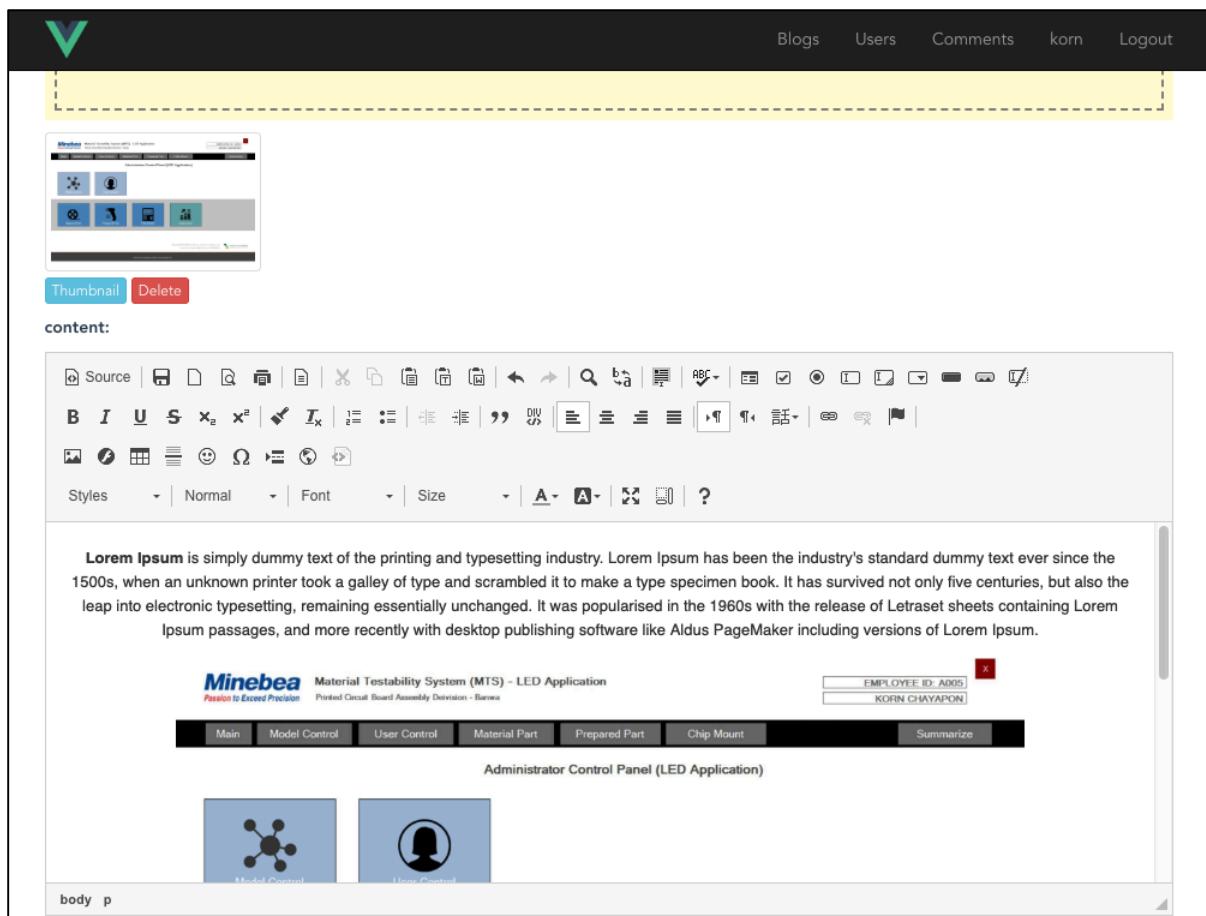
```
db: {
  database: process.env.DB_NAME || 'nvwebblog',
  user: process.env.DB_User || 'root',
  password: process.env.DB_PASS || '',
  options: {
    dialect: process.env.DIALECT || 'mysql',
    // storage: './nvwebblog-db.sqlite'
    host: process.env.HOST || 'localhost',
  },
},
```

ทำการเปิดหน้า blog create ของเรารีบบันมาทำการสร้าง blog เพื่อทดสอบดูครับว่าเราติดต่อกับ mysql ของเราได้หรือไม่ อย่างไร

จากทดสอบพบเจอบัญหาจากการแปลงฐานข้อมูลของเรา คือ field ของเราไม่ได้ทำเป็น primary key มาให้ ผມจึงแก้ไขโครงสร้างด้วย MySQL Workbench ของผมดังนี้ หากท่านใช้ตัวอื่น สามารถดูที่ structure และ add Primary Key เป็น Auto Increment ได้เลย

The screenshot shows the MySQL Workbench interface with the database 'nvwebblog' selected. The 'Structure' tab is active, displaying the 'Blogs' table. The 'Fields' section lists the columns: id, title, thumbnail, pictures, content, category, status, createdAt, and updatedAt. The 'id' column is highlighted with a red box. Its properties are: Type: INT, Length: 11, Unsigned: checked, Zerofill: unchecked, Binary: unchecked, Allow Null: unchecked, Key: PRI (Primary Key), Default: NULL, Extra: auto_incre..., Encoding: UTF-8, Collation: utf8mb4_..., and Comment: . The 'Indexes' section shows one index: Non_unique: 0, Key_name: PRIMARY, Seq_in_index: 1, Column_name: id, Collation: A, Cardinality: 4, Sub_part: NULL, Packed: NULL, and Comment: .

จากนั้นทำการทดสอบเพิ่ม blog อีกครั้ง



บังโก้ ผมเปลี่ยนจาก SQLite มาใช้งาน MySQL ได้แล้วครับ ง่ายมากเลย แก้ไม่กี่บรรทัดเอง ข้อดีของ SQLite ผมเคยกล่าวไปแล้ว คือ ความไม่เป็น server ของมันทำงานได้รวดเร็วติดตั้งง่าย ใช้งานง่าย backup ง่าย ส่วนเรื่องความปลอดภัยนั้น ถูกปกป้องไว้ด้วย node server ของเราอยู่แล้วไม่ต้องกังวล และไม่ต้องไปวิ่งหา database server ให้เสียเวลาครับ ส่วน MySQL น่าจะเป็น เพราะว่า เว็บในยุคกลางยังใช้กันอยู่ เราเลยทิ้งมันไปไม่ได้นั่งเองครับ

บทที่ 19 เริ่มต้นใส่ส่วนขายหนังสือกัน

เริ่มต้นทำ api เพื่อใช้งาน Back End ของเราในส่วนร้านหนังสือกัน

ในบทนี้เราจะทำการส่วนขายหนังสือกันครับ แต่เราจะไปกันแบบเร็วหน่อยนะครับ เพราะน่าจะเก่งกันแล้วทุกคน ที่ผ่านมาถึงบทนี้ได้ เราเริ่มต้นทำหลังบ้านกันก่อนครับ โดยผมเริ่มจากทำ Book Model ก่อนเลยครับ ผมสร้าง Model ชื่อว่า Book.js ไว้ใน src/models/Book.js ของเราครับ โดยมีรายละเอียดดังนี้

```
module.exports = (sequelize, DataTypes) => {
  const Book = sequelize.define('Book', {
    title: DataTypes.STRING,
    thumbnail: DataTypes.STRING,
    pictures: DataTypes.STRING,
    content: DataTypes.TEXT,
    category: DataTypes.STRING,
    status: DataTypes.STRING,
    prices: DataTypes.STRING
  })

  return Book
}
```

พอเรา save ปุ๊บ มันโปรแกรมที่เราเขียนขึ้นก็จะไปสร้างตารางให้เราใน MySQL ทันทีครับ

MySQL 5.7.18 | 127.0.0.1/nvwebblog/Books

Tables

	Field	Type	Length	Unsigned	Zerofill	Binary	Allow Null	Key	Default	Extra	Encoding	Collation	Comment
Blogs	id	INT	11					PRI	auto_incr...	auto_incr...	utf8	utf8_general_ci	
Books	title	VARCHAR	255						NULL	None	utf8	utf8_general_ci	
Comments	thumbnail	VARCHAR	255						NULL	None	utf8	utf8_general_ci	
sqlite_sequence	pictures	VARCHAR	255						NULL	None	utf8	utf8_general_ci	
Users	content	TEXT							NULL	None	utf8	utf8_general_ci	
	category	VARCHAR	255						NULL	None	utf8	utf8_general_ci	
	status	VARCHAR	255						NULL	None	utf8	utf8_general_ci	
	prices	VARCHAR	255						NULL	None	utf8	utf8_general_ci	
	createdAt	DATETIME								None			
	updatedAt	DATETIME								None			

Indexes

Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Comment
0	PRIMARY	1	id	A	0	NULL	NULL	

Table Information

- created: 20/3/2562 BE
- engine: InnoDB
- rows: 0
- size: 16.0 KiB
- encoding: utf8
- auto_increment: 1

ເຫັນມີຄວບ ຂລາດນາກເລຍ ໂປຣແກຣມທີ່ເຊື່ອນຈຶ່ນ

ຈາກນັ້ນເຮັດໄປສ້າງ BookController ຈຶ່ນມາ ພມໃຊ້ວິທີທີ່ເທັນເຄົາໃຊ້ກັນຄືອ copy ໄປ
ເປີ່ມຍື່ນຊື່ອເລຍຄວບ ນັ້ນ ຄືອ ເຮັດວຽກ BookController.js ເພີ້ນຮ້ອຍແລ້ວນັ້ນເອງ route
mapping ໃຫ້ api ຂອງເຮັດວຽກໃຊ້ Controller ຂອງເຮົາໄດ້ດັ່ງນີ້ຄວບ

```
const BookController = require('./controllers/BookController')
```

```
// book route
// create book
app.post('/book',
  isAuthenController,
  BookController.create
)

// edit book, suspend, active
app.put('/book/:bookId',
  isAuthenController,
  BookController.put
)

// delete book
app.delete('/book/:bookId',
  isAuthenController,
  BookController.remove
```

```

    )

    // get book by id
    app.get('/book/:bookId',
        BookController.show
    )

    // get all book
    app.get('/books',
        isAuthenController,
        BookController.index
    )

    app.get('/books/front',
        BookController.frontIndex
    )

```

เรียนรู้อยครับ api ของเรารวมใช้งาน ทดสอบด้วย POSTMAN กันเองนะครับ แต่ใครที่มือถึงและมั่นใจแล้ว ผ่านเลยครับ ไปจดหน้าบ้านกันเลย error เดี๋ยวค่อยมาเช็คครับ จะได้ไม่เสียเวลา

เริ่มต้นทำ Front End Service สำหรับจัดการสินค้า (Back Office)

ทำการ focus ไปที่ client ครับ เพราะเราจะทำ front end กัน เราทำการสร้าง Book Service ขึ้นมาที่ src/services/BooksService.js จากนั้นเขียน code ดังนี้ครับ

```

import Api from '@/services/Api'

export default {
    index (search) {
        return Api().get('books', {
            params: {
                search: search
            }
        })
    },
    frontIndex (search) {
        return Api().get('books/front', {
            params: {

```

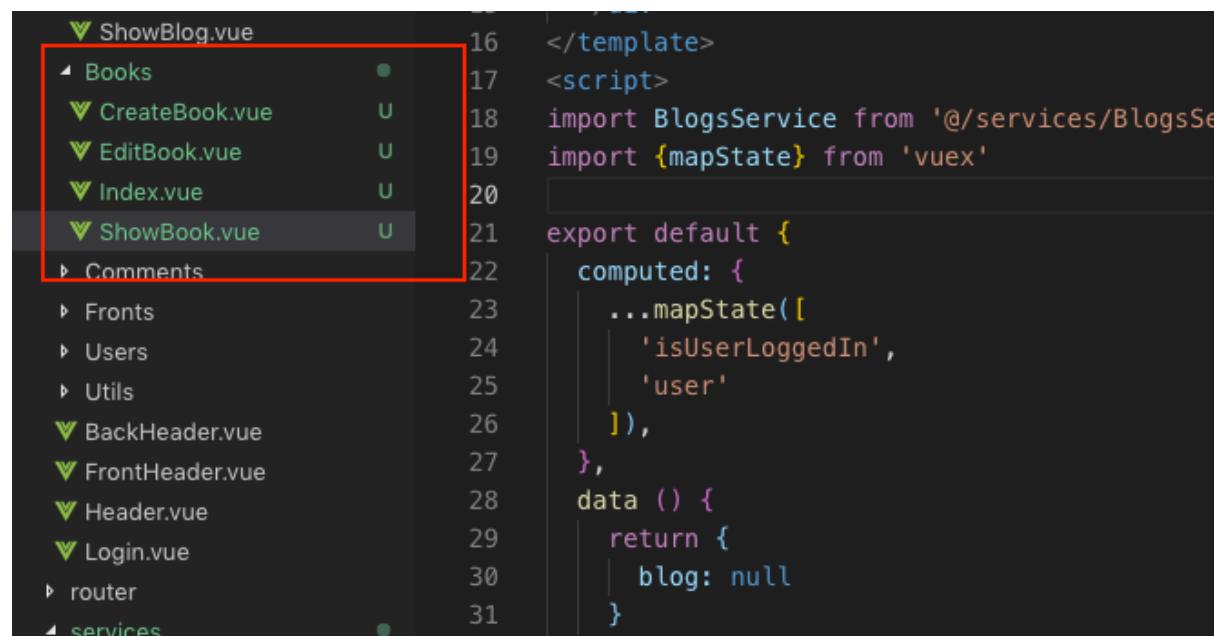
```

        search: search
    }
})
},
,

show (bookId) {
    return Api().get('book/'+bookId)
},
post (book) {
    return Api().post('book', book)
},
put (book) {
    return Api().put('book/'+book.id, book)
},
delete (book) {
    return Api().delete('book/'+book.id, book)
},
}

```

จากนั้นสร้างโฟล์เดอร์ Books ใน components และเพิ่มไฟล์ต่าง ๆ ดังนี้



```

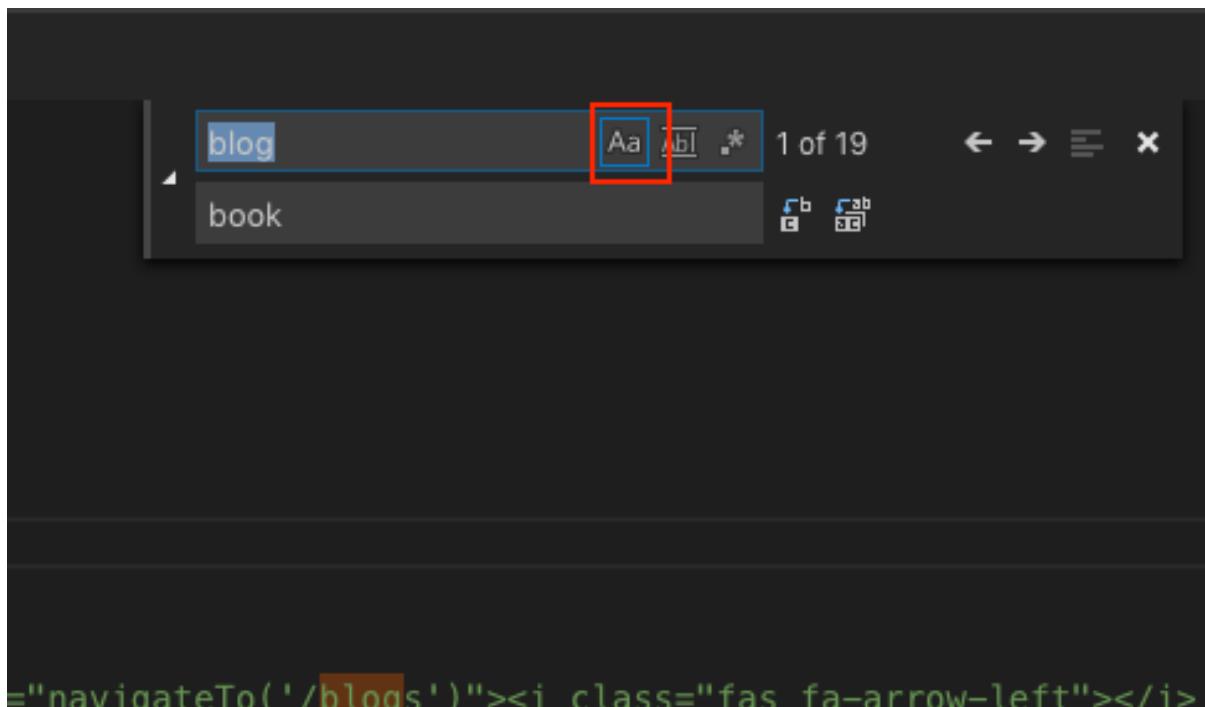
▼ ShowBlog.vue
  ▾ Books
    ▾ CreateBook.vue
    ▾ EditBook.vue
    ▾ Index.vue
    ▾ ShowBook.vue
    ▾ Comments
      ▾ Fronts
      ▾ Users
      ▾ Utils
    ▾ BackHeader.vue
    ▾ FrontHeader.vue
    ▾ Header.vue
    ▾ Login.vue
  ▾ router
  ▾ services

```

```

16  </template>
17  <script>
18  import BlogsService from '@/services/BlogsSe
19  import {mapState} from 'vuex'
20
21  export default {
22    computed: {
23      ...mapState([
24        'isUserLoggedIn',
25        'user'
26      ]),
27    },
28    data () {
29      return {
30        blog: null
31      }
32    }
33  }
34
35  
```

ซึ่งจริง ๆ แล้ว ผม copy มาวางทั้ง Blog โฟล์เดอร์และทำการเปลี่ยนชื่อไฟล์เป็นที่เห็น จากนั้นทำการค้นหาและแทนที่ blog เป็น book และ Blog เป็น Book ในทุก ๆ ไฟล์ โดยการกด **ctrl+alt+f** จากนั้นเลือกว่า ให้สันใจตัวเล็กตัวใหญ่ด้วย



ตอนนี้เรารายได้ Book Component ครบหมดแล้วนั่นเองครับ จากนั้นเราจะไปทำ route mapping กัน เพื่อให้สามารถเข้าถึง url ของเราได้ ทำการเปิด router/index.js ขึ้นมา และเพิ่ม code เข้าไปดังนี้

```
// Books
import BookIndex from '@/components/Books/Index'
import BookCreate from '@/components/Books/CreateBook'
import BookEdit from '@/components/Books/EditBook'
import BookShow from '@/components/Books>ShowBook'
```

```
// books
{
  path: '/books',
  name: 'books',
  component: BookIndex
},
{
  path: '/book/create',
  name: 'books-edit',
  component: BookCreate
},
{
  path: '/book/edit/:bookId',
```

```

        name: 'book-edit',
        component: BookEdit
    },
{
    path: '/book/:bookId',
    name: 'book',
    component: BookShow
},

```

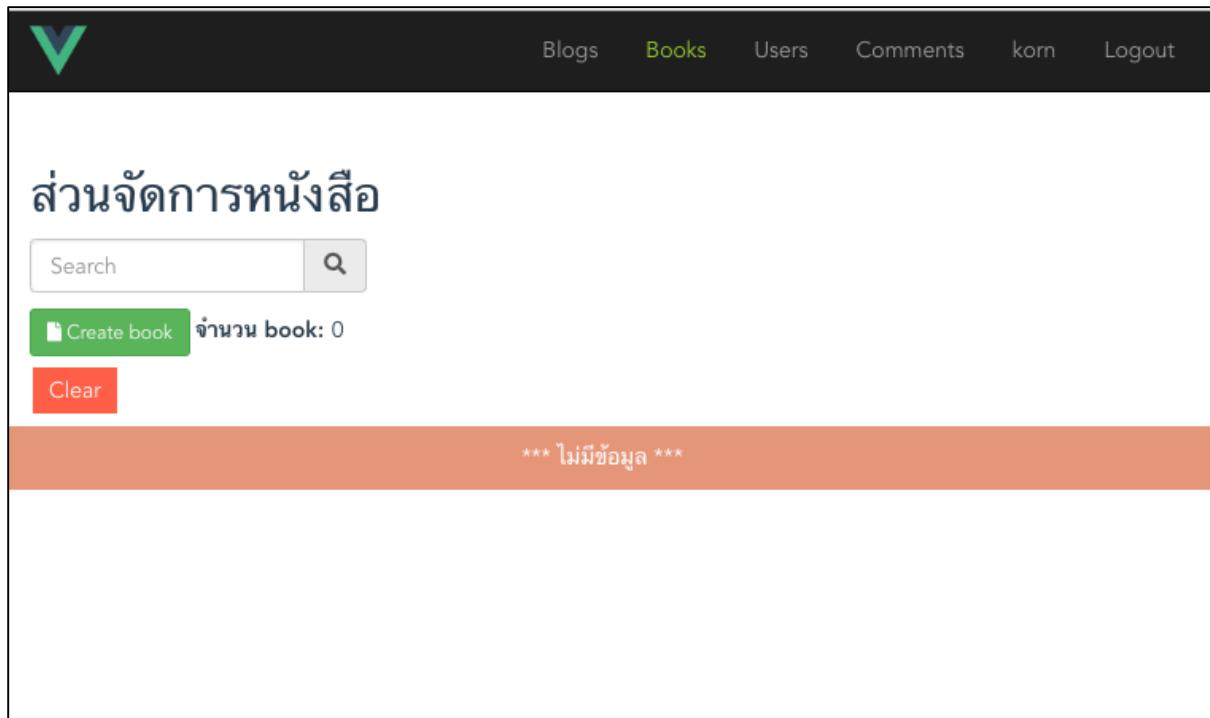
ตอนนี้ link และ component ต่าง ๆ เกี่ยวกับ Book ของเราในหน้าบ้านน่าจะพร้อมใช้งานกันแล้ว เราทำการเพิ่ม link ของไปที่ BackHeader ของเราอีกอันหนึ่งซึ่งชื่อว่า Book ครับ

```

<li      role="presentation"><router-link      :to="{name:      'books'}">Books</router-link></li>

```

ลองเปิดเว็บบล็อกของเราทดสอบดูครับ



บังゴ้ รวดเร็วเหมือนเสกมาเลยครับ ได้ส่วนจัดการหนังสือมาแล้วครับ
 เปิด src/components/Books/CreateBook.vue และ src/components/Books/CreateBook.vue ขึ้นมาครับ แล้วทำการ เพิ่มตัวแปร prices ใน data () และเพิ่มฟิลด์สำหรับใส่ราคาให้ Form ของเราครับ

```
prices: ''
```

```
<p>
  <label class="control-label">Prices :</label>
  <input type="text" v-model="book.prices" class="form-control">
</p>
<p>
```

จากนั้นไปเพิ่ม code แสดงราคาให้กับหนังสือของเรากัน ที่ส่วนจัดการหลัก คือ src/components/Books/Index.vue ดังนี้

```
<p><strong>Prices:</strong> {{ book.prices }} บาท</p>
```

แต่การแสดงผลแต่ตัวเลขนั้นทำให้ นำงดหigid เล็กน้อย ผิดจะเพิ่ม code ชุดนี้เข้าไปที่ filters ของเรา

```
getNumberWithCommas(x) {
  return x.toString().replace(/\B(?=(\d{3})+)(?!\d))/g, ",");
},
```

ซึ่งเป็นการใช้ Regular Expression มาช่วยนั่นเอง เรื่องนี้เป็นเรื่องที่ค่อนข้างยาก copy ไปใช้ได้เลยครับ หรือใครอยากรู้ก็สามารถ goole ได้เลยครับ

จากนั้นผิดจะแก้ การแสดงราคาใน <template></template> ของเราเป็นดังนี้ ครับ

```
<p><strong>Prices:</strong> {{ book.prices | getNumberWithCommas }} บาท
</p>
```

จากนั้น ราคานั้นจะมีตัวหนังสือของเราระบุรวม คือม่า ทำให้ดูแล้วไม่น่าดูแล้ว ครับ

ทำการเพิ่มหนังสือที่เราขายเข้าไปในระบบสัก 5-10 เล่มกันดูนะครับ แล้วทดสอบว่าแต่ไง และหยุดการขายได้ด้วยหรือไม่อีกต่อไปครับ

ตอนนี้เราทำหลังบ้านหรือ Back Office ในส่วนเพิ่มสินค้าของเราระบบแล้วนั้นเอง ครับ แต่เรายังไม่ได้ขายอะไรมาก่อนเลย ซึ่งในการขายนั้น เราจะต้องมีส่วนจัดการการขายแบบง่าย ๆ ซึ่งเก็บข้อมูลว่า ใครเป็นคนซื้อ รวมถึงผู้ซื้อก็จะต้องดูได้ว่า เราส่งของหรือยัง ซึ่งมีรายละเอียดอยู่พอสมควร โดยผมจะทำแบบง่าย ๆ เพื่อเป็น guide line ให้ดูนะครับ ส่วนรายละเอียดลึก ๆ ผมว่าทุกคนทำได้อยู่แล้ว เพียงแต่ต้องใช้เวลาเท่านั้นเอง

ทำส่วนจัดการ การสั่งซื้อ

เริ่มต้นทำส่วน Back End สำหรับจัดการการสั่งซื้อ

ทำการสร้าง Buy Model ขึ้นมาเพื่อจัดการ การสั่งซื้อ ebook ของเรา โดยสร้างไฟล์ src/models/Buy.js ขึ้นมา แล้วทำการเขียน code ดังนี้

```
module.exports = (sequelize, DataTypes) => {
```

```

const Buy = sequelize.define('Buy', {
  bookid: DataTypes.STRING,
  userid: DataTypes.STRING,
  qty: DataTypes.STRING,
  status: DataTypes.STRING,
  booktitle: DataTypes.STRING,
  username: DataTypes.STRING,
  userlastname: DataTypes.STRING,
})

return Buy
}

```

สังเกตว่าผมจะใส่ booktitle, username, userlastname เอาไว้ด้วย เพื่อจะได้ง่ายในการค้นหา การสั่งซื้อของเรา จะได้ไม่ต้องเอา id ไปค้นจาก database อีกครึ่งหนึ่งแล้ว

จากนั้นทำการสร้าง Buy Controller ขึ้นมาที่ src/controllers/BuyController.js โดยมีรายละเอียดดังนี้

```

const {Buy} = require('../models')

module.exports = {
  // index with serach buy
  async index (req, res) {
    try {
      let buys = null
      const search = req.query.search
      console.log('-----> search key: ' + search)

      if (search) {
        buys = await Buy.findAll({
          where: {
            $or: [
              'booktitle', 'username', 'userlastname'
            ].map(key => ({
              [key]: {
                $like: `%${search}%`,
              }
            })),
        },
      }
    }
  }
}

```

```

        order: [['createdAt', 'DESC']]
    })
} else {
    buys = await Buy.findAll({
        order: [['createdAt', 'DESC']]
    })
}
res.send(buys)
} catch (err) {
    res.status(500).send({
        error: 'an error has occured trying to fetch the buys'
    })
}
},
// create buy
async create (req, res) {
    // res.send(JSON.stringify(req.body))
    try {
        const buy = await Buy.create(req.body)
        res.send(buy.toJSON())
    } catch (err) {
        res.status(500).send({
            error: 'Create buy incorrect'
        })
    }
},
// edit buy, suspend, active
async put (req, res) {
    try {
        await Buy.update(req.body, {
            where: {
                id: req.params.buyId
            }
        })
        res.send(req.body)
    } catch (err) {
        res.status(500).send({
            error: 'Update buy incorrect'
        })
    }
}
}

```

จากนั้นเราทำการ route mapping เพื่อเรียกใช้ BuyController ของเราดังนี้

```
const BuyController = require('./controllers/BuyController')
```

```
// buy route
// create buy
app.post('/buy',
  isAuthenController,
  BuyController.create
)

// edit buy, suspend, active
app.put('/buy/:buyId',
  isAuthenController,
  BuyController.put
)

// get all buy
app.get('/buys',
  isAuthenController,
  BuyController.index
)
```

ตอนนี้ Back End ของเราพร้อมแล้ว สำหรับการสั่งซื้อหนังสือจาก Client ของเรา

เริ่มต้นทำ Front End ส่วนซื้อหนังสือ

ผมเริ่มต้นโดยการ ทำการสร้าง src/components/Fronts/Books.vue หรือ Book Component และนำส่วนของ src/components/Fronts/Index.vue มาทำการแก้ไข โดยปรับเปลี่ยน Hero เข้าไป ดังนี้

```
<div class="hero">
  
    <h1>ร้านหนังสือออนไลน์ สั่งซื้อได้เลยครับ</h1>
    <p>By Gooddev.ME</p>
</div>
```

แล้วทำการเพิ่ม Style ดังนี้

```
.hero {  
    margin-top: 80px;  
    border-radius: 5px;  
    background: lightslategray;  
    height: 250px;  
    color: white;  
    padding: 20px;  
}  
  
.hero h1 {  
    margin-top: 30px;  
}  
.logo {  
    padding-right: 20px;  
    max-width: 200px;  
}
```

และแก้ไขจากการใช้งาน BlogService ต่าง ๆ มาใช้งาน BooksService ในการดึงข้อมูลจาก Server แทน

มาถึงตอนนี้ผู้สอนทำการเพิ่มตัวแปรใน data () ของเรา

```
newBooks: [] ,
```

เพื่อจะทำการโหลดหนังสือมาใหม่ จากนั้นผู้สอนเพิ่ม script ในการโหลดหนังสือมาใหม่ดังนี้ครับ

```
async created () {  
    let allBooks = (await BooksService.frontIndex()).data  
    this.newBooks = allBooks.slice(0,4)  
},
```

แล้วทำการแสดงผลหนังสือ มาใหม่ของผู้สอนดังนี้

```
<div class="container new-book">  
    <h2>หนังสือมาใหม่</h2>  
    <div class="row">  
        <div class="col-md-3" v-for="book in newBooks" v-bind:key="book.id" >  
            <div v-if="book.thumbnail != 'null'">  
                
```

```
</div>
<h4>{{ book.title }}</h4>
<div v-html="book.content.slice(0,200) + '...'></div>
<p><strong>ราคา: </strong>{{ book.prices | getNumberWithCommas
}}</p>
<div>
    <button v-on:click.prevent="addCart(book)" class="btn btn-
sm btn-danger"><i class="fas fa-shopping-cart"></i> เพิ่มลงตะกร้า</button>
    <button class="btn btn-sm btn-info" v-
on:click="navigateTo('/front-view-book/' + book.id)"><i class="fab fa-
readme"></i> อ่านเพิ่ม</button>
</div>
</div>
</div>
```

แล้วทำการเพิ่ม style ดังนี้ครับ

```
.new-book h2 {
    font-family: kanit;
    padding-bottom: 20px;
    margin-top: 30px;
}
```

ตอนนี้เรารสามารถโหลดหนังสือมาใหม่ได้แล้วครับ โดยผมจะโหลดแค่ 4 เล่มพอนะครับ

จะเห็นว่าผมได้เพิ่มปุ่มสำหรับ เพิ่มสินค้าในตระกร้าไว้เรียบร้อยแล้วในการแสดงรายการหนังสือของผมทั้งหมด

ผมจะทำการสร้างตัวแปรเพื่อจัดการตระกร้าสินค้าของผมใน data () ดังนี้

```
carts: [],
total: 0
```

และเพิ่ม function ในการเพิ่มสินค้าในตระกร้าสินค้าของผมดังนี้

```
addCart (book) {
  if (this.user == null) {
    alert('Please, Register or Login before.\n\nThank you.')
  } else {
    this.total += parseInt(book.prices)

    let found = false
    this.carts.map((cart) => {
      if(cart.bookid == book.id) {
        cart.qty++
        found = true
      }
    })

    if (!found) {
```

```

let cart = {
    bookid:book.id,
    userid:this.user.id,
    qty:1,
    booktitle:book.title,
    username:this.user.name,
    userlastname:this.user.lastname,
    prices:book.prices
}
this.carts.push(cart)
}

// console.log('carts length: ' + this.carts.length)
},

```

การเพิ่มตระกร้าสินค้าของผู้รับເອົາ ສິນທີຈະເພີ່ມເຂົ້າມາຕົວແປຣ book ຈາກນັ້ນ ໃຊ້
ເໜີກກ່ອນວ່າ user ທຳການ login ອີຍ້ງ ຄ້າຍັງເຮັບັງຄັບໃຫ້ user login ກ່ອນ ຄ້າ login
ແລ້ວໃຫ້ທຳເພີ່ມ ຂໍອມຸລໃນ carts ຂອງເຮົານັ້ນເອງ ໂດຍກໍານົດວ່າຄ້າມີໜັງສືອເລີມເຕີມໃນ
cart ໃຫ້ເພີ່ມ qty ປື້ນ ອີກໜຶ່ງ

ຕອນນີ້ເຮົາມີຂໍອມຸລແລ້ວ ແຕ່ເຮັຍັງໄມ້ມີສ່ວນແສດງຜລຕະກົມາສິນຄ້າ ເຮົາຈຶ່ງຕ້ອງທຳການ
ເພີ່ມສ່ວນ ແສດງຜລໃນ <template></template> ຂອງເຮົາດັ່ງນີ້

```

!-- my carts -->
<transition name="fade">
<div class="popup-cart" v-if="carts.length">
<h3>ຕະກົມາສິນຄ້າ</h3>
<ul class="cart">
<li v-for="cart in carts" >
<div>
    {{ cart.booktitle }} ຈຳນວນ{{ cart.qty}} x {{ cart.prices }}
</div>
<div>
    <button v-on:click.prevent="inc(cart)">+</button>
    <button v-on:click.prevent="dec(cart)">-</button>
</div>
</li>
</ul>
<hr>
<p>ຮວມທີ່ງສູນ: {{total | getNumberWithCommas}} ພາສາ </p>

```

```

        <button class="btn btn-xs btn-danger" v-
on:click.prevent="sendBuy" ><i class="fas fa-check-square"></i> ทำการสั่งซื้อ
</button>
    </div>

```

ซึ่งผู้กำหนดแสดงผลเมื่อมี สินค้าในตระกร้า หรือ cart ของเรานั้นเอง รวมถึงเพิ่ม บุ้ม +/- จำนวนสินค้าที่เลือกซื้อ ดังนั้นผมต้องเพิ่ม function ใน methods ของผม เพื่อจัดการมันด้วยดังนี้ครับ

```

inc (item) {
    item.qty++
    this.total += parseInt(item.prices)
},
dec (item) {
    item.qty--
    this.total -= parseInt(item.prices)

    if(item.qty <= 0) {
        let i = this.carts.indexOf(item)
        this.carts.splice(i, 1)
        // this.total = 0
    }
},

```

แล้วทำการเพิ่ม style เพื่อแสดงผล รายการสินค้าของเราดังนี้ครับ

```

.popup-cart {
    box-shadow: 0 2px 4px 0 rgba(0,0,0,.2);
    border: solid 1px #ddd;
    background: #fff;
    width:360px;
    padding: 10px;
    position:fixed;
    bottom:0;
    right:0;
    border-radius: 5px;
    margin-bottom: 5px;
    margin-right: 5px;
}

```

ตอนนี้ตระกร้าสินค้าของเราเสร็จเรียบร้อยแล้วแล้ว เตรียมที่ทำการสั่งซื้อได้ละครับ
แต่มันได้ถูกเรียกใช้งานได้

เราต้องไปทำ route mapping เพื่อเรียกใช้ Book Component ของเรา ที่ router/index.js

```
import FrontBooks from '@/components/Fronts/Books'
```

```
{
  path: '/front-books',
  name: 'front-books',
  component: FrontBooks
},
```

จากนั้นทำการเพิ่ม menu book ใน FrontHeader ของเรา เพื่อใช้งานผ่าน navbar ของเรา

```
<li role="presentation"><router-link :to="{name: 'front-books'}" ><i class="fas fa-book-open"></i> Books</router-link></li>
```

ทำการสั่งซื้อหนังสือ

ในการแสดงตระกร้าสินค้าของเราเมื่อปุ่มทำการสั่งซื้อเรียบร้อยแล้ว เราจะทำการ
เพิ่ม function ในการสั่งซื้อสินค้าดังนี้ครับ

```
sendBuy () {
  this.carts.forEach(async (cart) => {
    console.log("cart: " + JSON.stringify(cart))

    try {
      await BuysService.post(cart)
      this.$router.push({
        name: 'cartlist'
      })
    } catch (err) {
```

```

        console.log(err)
    }
})
},

```

ซึ่งเป็นการ Post data ไปเพื่อบันทึกข้อมูลนั้นเอง จากนั้นผมทำการเพิ่มหน้ารายการสั่งซื้อสินค้าของ user ดังนี้

```

<template>
  <div id="carlist">
    <main-header navsel="front"></main-header>
    <div class="user-info">
      <h1>แสดงข้อมูลผู้ใช้งาน</h1>
      <div class="form-wrapper" v-if="user != null">
        <div class="form-horizontal">
          <div class="form-group">
            <label for="" class="control-label col-md-2">Name: </label>
            <div class="col-md-8">
              <input class="form-control" type="text" v-
model="user.name" disabled>
            </div>
          </div>

          <div class="form-group">
            <label for="" class="control-label col-md-2">Lastname:</label>
            <div class="col-md-8">
              <input class="form-control" type="text" v-
model="user.name" disabled>
            </div>
          </div>

          <div class="form-group">
            <label for="" class="control-label col-md-2">Email: </label>
            <div class="col-md-8">
              <input class="form-control" type="email" v-
model="user.email" disabled>
            </div>
          </div>
        </div>
      </div>
    </div>
  </div>
  <!-- cart info -->

```

```

<div class="cart-info container">
  <div class="rows">
    <div class="col-md-12">
      <div class="transection-wrapper">
        <h4>ຮາຍລະບະເນື້ອດກາຮສ່ວ່ນ</h4>
        <ul class="trasaction-list">
          <li v-for="transection in transections" v-bind:key="transection.id">
            <h4>{{ transection.booktitle }} ທຳນານ {{ transection.qty }}  

              × {{ transection.prices }}</h4>
            <p><strong>ຮາງຮມ :</strong> {{ transection.qty }} *  

              transection.prices | getNumberWithCommas }> ນາມ</p>
            <p><strong>ສຄານະຄູກຄ້າ:</strong> {{ transection.clientStatus }}</p>
            <p><strong>ເປົ້າໃຈ: </strong> {{ transection.clientStatus }}</p>
            <p><strong>ວັນທີ:</strong> {{ transection.createdAt }} |  

              formatedDate }</p>
            </li>
          </ul>
        </div>
      </div>
    </div>
  </div>
</div>
</template>
<script>
import {mapState} from 'vuex'
import BuysService from '@/services/BuysService'
import moment from 'moment'

export default {
  filters: {
    formatedDate (value) {
      return moment(String(value)).format('DD-MM-YYYY')
    },
    getNumberWithCommas(x) {
      return x.toString().replace(/\B(?=(\d{3})+(?!\d))/g, ",");
    },
  },
  computed: {

```

```
...mapState([
  'isUserLoggedIn',
  'user'
])
},
data () {
  return {
    transections:[]
  }
},
async created () {
  this.transections = (await BuysService.user(this.user.id)).data
  console.log(this.transections)
}
}

</script>
<style scoped>
.user-info h1{
  text-align: center;
}

.trasection-list {
  list-style: none;
  padding: 0;
  margin: 0;
}

.trasection-list li {
  border:solid 1px #dfdfdf;
  margin-bottom: 10px;
  padding: 20px;
  box-shadow: 0 2px 4px 0 rgba(0,0,0,.1);
}

.cart-info {
  margin-top: 50px;
}
</style>
```

ซึ่งเป็นการดึงข้อมูลจาก Buy Model ของเราขึ้นมาตาม userid ในส่วนนี้ถ้าใครยังไม่ได้เพิ่ม api ในการ get buy model by user ให้ทำการเขียนเพิ่มด้วยนะครับ ผมจะไม่ทำให้ดู เพราะน่าจะทำเองกันได้ทุกคนแล้วตอนนี้

ในการขาย ebook ผมของผมเป็นการขายแบบ online ลูกค้าสามารถติดต่อกับเว็บไซต์เพื่อแจ้งการชำระเงินและกดยืนยันการชำระเงินได้เองเมื่อชำระเงินแล้ว โดยผมเพิ่มบุ่มยืนยันการชำระเงินดังนี้

```
<p><button v-on:click.prevent="sendPaid(transection.id)" class="btn btn-xs btn-success">ยืนยันการชำระเงิน</button></p>
```

```
async sendPaid (id) {
  let transection = {
    "id": id,
    "clientStatus": "ชำระแล้ว"
  }

  try {
    await BuysService.put(transection)
    this.reloadData()
  } catch (error) {
    console.log(error)
  }
},
```

พร้อมกลับ refresh data ของผมดังนี้

```
async reloadData() {
  this.transections = (await BuysService.user(this.user.id)).data
}
```

ตอนนี้เราจะจัดการส่วนของ Front Office เพื่อทำการสั่งซื้อสินค้าเสร็จแล้ว

ทำส่วนของ Back Office เพื่อดูคำสั่งซื้อและกำหนดสถานะการจัดสั่ง

จากนี้เราจะทำ Back Office เพิ่มเติม เพื่อดูรายการการสั่งซื้อที่ลูกค้าสั่งซื้อข้อมาจากนั้น โดยผมทำการสร้าง src/components/Books/CartList.vue

แล้วทำการเขียน code ดังนี้

```

<template>
  <div>
    <main-header navsel="back"></main-header>
    <div class="container">
      <div class="cartlist-warpper">
        <!-- cart info -->
        <div v-if="transections.length" class="cart-info container">
          <div class="rows">
            <div class="col-md-12">
              <div class="transection-wrapper">
                <h4>รายละเอียดการสั่งซื้อ</h4>
                <ul class="trasection-list">
                  <li v-for="transection in transections" v-bind:key="transection.id">
                    <h4>{{ transection.booktitle }}</h4>
                    {{ transection.qty}} x {{ transection.prices }}</h4>
                    <h4>email: {{ transection.email }}</h4>
                    <h4>ชื่อ: {{ transection.username }}</h4>
                    <h4>นามสกุล: {{ transection.userlastname }}</h4>
                    <p><strong>ราคารวม:</strong> {{ transection.qty * transection.prices | getNumberWithCommas }} บาท</p>
                    <p><strong>สถานะ:</strong> {{ transection.clientStatus }}</p>
                    <p><strong>สถานะร้านค้า:</strong> {{ transection.shopStatus }}</p>
                    <p><button v-on:click.prevent="sendSent(transection.id)" class="btn btn-xs btn-success">ยืนยันการสั่ง</button></p>
                    <p><strong>วันที่:</strong> {{ transection.createdAt | formatedDate }}</p>
                  </li>
                </ul>
              </div>
            </div>
          </div>
        <div v-else class="container">
          <div class="trasection-null">
            ไม่มีรายการสั่งซื้อข้อมูลนี้
          </div>
        </div>
      </div>
    </div>
  </div>

```

```

</template>
<script>
import {mapState} from 'vuex'
import BuysService from '@/services/BuysService'
import moment from 'moment'

export default {
  filters: {
    formatedDate (value) {
      return moment(String(value)).format('DD-MM-YYYY')
    },
    getNumberWithCommas(x) {
      return x.toString().replace(/\B(?=(\d{3})+)(?!\d))/g, ",");
    },
  },
  data () {
    return {
      transections: []
    }
  },
  async created () {
    this.refreshData()
  },
  methods: {
    async refreshData() {
      this.transections = (await BuysService.index()).data
    },
    async sendSent (id) {
      let transection = {
        "id": id,
        "shopStatus": "ส่งแล้ว"
      }

      try {
        await BuysService.put(transection)
        this.refreshData()
      } catch (error) {
        console.log(error)
      }
    },
  }
}
</script>

```

```

<style scoped>
.cartlist-warpper {
  margin-top: 80px;
}

.trasection-null {
  border:solid 1px #dfdfdf;
  margin-bottom: 10px;
  padding: 20px;
  box-shadow: 0 2px 4px 0 rgba(0,0,0,.1);
  margin-top: 30px;
}

.user-info h1{
  text-align: center;
}

.trasection-list {
  list-style: none;
  padding: 0;
  margin: 0;
}

.trasection-list li {
  border:solid 1px #dfdfdf;
  margin-bottom: 10px;
  padding: 20px;
  box-shadow: 0 2px 4px 0 rgba(0,0,0,.1);
}

.cart-info {
  margin-top: 50px;
  font-family: kanit;
}
</style>

```

ซึ่งจริง ๆ แล้ว code ชุดนี้ไม่มีอะไรมากนอกจาก Query All ใน Buy Model ขึ้นมาแสดง และทำการกำหนดสถานการณ์จัดส่งให้ ผู้ใช้สามารถตรวจสอบ email ได้นั่นว่า ว่าได้รับหรือยัง โดยส่วน cart list ของ Front End จะมีหน้าตาดังนี้

The screenshot shows a web application interface. At the top, there is a dark header bar with a green logo on the left and navigation links for "Home", "Books", "user", and "Logout" on the right. Below the header, the main content area has a title "แสดงข้อมูลผู้ใช้งาน" (Display User Information). There are three input fields for "Name" (user), "Lastname" (user), and "Email" (user@gmail.com). Below these fields, a section titled "รายละเอียดการสั่งซื้อ" (Order Details) contains two items:

1. บิ๊กบันทึ้งห้อง จำพวก 2 x 150

ราคาหน่วย : 300 บาท
สถานะลูกค้า : รอชำระ
วันที่สั่ง : 20-03-2019

2. Image Processing จำพวก 2 x 5000

ราคาหน่วย : 10,000 บาท
สถานะลูกค้า : รอชำระ
วันที่สั่ง : 20-03-2019

และส่วน cartlist ในส่วนของ Back End ของผมจะเป็นดังนี้



รายละเอียดการสั่งซื้อ

บิลกานต์ห้อง จํานวน 1 x 150
email: korn.chayapon@gmail.com
ชื่อ: korn chayapon
ราคารวม : 150 บาท
สถานะลูกค้า : ยังไม่ได้
สถานะร้านค้า : ส่งแล้ว
[ยืนยันการสั่ง](#)
วันที่ : 20-03-2019

กํา front end ด้วย vuejs จํานวน 1 x 550
email: korn.chayapon@gmail.com
ชื่อ: korn chayapon
ราคารวม : 550 บาท
สถานะลูกค้า : ยังไม่ได้
สถานะร้านค้า : ส่งแล้ว
[ยืนยันการสั่ง](#)
วันที่ : 20-03-2019

Image Processing จํานวน 1 x 5000
email: korn.chayapon@gmail.com
ชื่อ: korn chayapon
ราคารวม : 5,000 บาท
สถานะลูกค้า : ยังไม่ได้
สถานะร้านค้า : รอสั่ง
[ยืนยันการสั่ง](#)
วันที่ : 20-03-2019

บิลกานต์ห้อง จํานวน 2 x 150
email: user@gmail.com
ชื่อ: user nakrub
ราคารวม : 300 บาท
สถานะลูกค้า : ยังไม่ได้
สถานะร้านค้า : รอสั่ง
[ยืนยันการสั่ง](#)
วันที่ : 20-03-2019

Image Processing จํานวน 2 x 5000
email: user@gmail.com
ชื่อ: user nakrub
ราคารวม : 10,000 บาท
สถานะลูกค้า : ยังไม่ได้
สถานะร้านค้า : รอสั่ง
[ยืนยันการสั่ง](#)
วันที่ : 20-03-2019

ซึ่งในการทำ ebook shop ของผมเป็นเพียงการประยุกต์ใช้งานแบบง่าย ๆ basic หรือเป็น guide line เพียงเท่านั้น ซึ่งผมได้นำเสนอแนวทางในส่วนที่วุ่นวาย คือ shopping cart ให้หมดแล้ว แต่จากการอ่านหนังสือเล่มนี้มาจนถึงตอนนี้ ผมเชื่อว่า

ทุกท่านจะสามารถทำเพิ่มเติมในส่วนรายละเอียดต่าง ๆ ที่ขาดหายไป ได้เอง ทั้งหมด เพียงอาศัยเวลาและการทบทวน ลงมือทำบ่อย ๆ

บทที่ 20 ติดตั้งขึ้น server ec2

เตรียม Back End เพื่อทำการติดตั้ง

ในบทนี้ผมจะทำการติดตั้งใช้งานบน Server จริง ๆ กัน หรือเรียกว่า Deploy ขึ้น Server AWS EC2 เนื่องจากผมไม่มี MySQL Database เพื่อทดสอบ而已 และไม่่อยากติดตั้งใหม่บน EC2 ผมเลยจะกลับมาใช้ SQLite เมื่อนอนเดิมจะได้ติดตั้งง่ายขึ้น

โดยผมทำการแก้ config/config.js ของเราเป็นดังนี้

```
module.exports = {
  port: 8081,
  db: {
    database: process.env.DB_NAME || 'nvwebblog',
    user: process.env.DB_User || 'root',
    password: process.env.DB_PASS || '',
    options: {
      dialect: process.env.DIALECT || 'sqlite',
      storage: './nvwebblog-db.sqlite'
      // host: process.env.HOST || 'localhost',
    },
  },
  authentication: {
    jwtSecret: 'korn'
  }
}
```

จากนั้นทำการ login ในส่วนของ admin ค้างไว้ อย่า logout นะครับ

ไม่งั้นเรา clear database เราจะสร้าง user ไม่ได้นะครับ จากนั้นทำการเปลี่ยน force เป็น true และ restart server และ เปลี่ยน force เป็น false กลับมาถึงตอนนี้ database ของเราจะเป็น 0 คือ ว่างเปล่ามาถึงตอนนี้ กลับที่หน้า Back Office ที่เราเปิดค้างไว้ทำการสร้าง User ของ admin เข้าไปครับ

จากนั้นทำการ Login อีกครั้ง เราจะเข้ามาจัดการ Back Office ของเราได้เหมือนเดิม ตอนนี้เราได้ทำการเปลี่ยนมาใช้ SQLite เรียบร้อยแล้วครับ

นอกจากเปลี่ยน database ของเราแล้วผมยังทำการเปลี่ยน port ของผมเป็น 80 ด้วยคับ จากนั้นทำการ commit ขึ้น Server ไป เราจะไป pull บน serve จะริงกันครับ

เริ่มต้นใช้งาน EC2

การติดตั้งใช้งานบน Amazon AWS Server นั้นเราสามารถทำการสมัครได้เลยครับ เพียงมี email เท่านั้น เข้าไปที่

<https://aws.amazon.com/th/ec2/> ถ้าใครมีบัญชีอยู่แล้ว ก็ทำการ Login ได้เลยครับ

AWS Management Console

AWS services

Find Services
You can enter names, keywords or acronyms.

Example: Relational Database Service, database, RDS

▶ All services

Build a solution

Get started with simple wizards and automated workflows.

Launch a virtual machine With EC2 2-3 minutes 	Build a web app With Elastic Beanstalk 6 minutes 
Build using virtual servers With Lightsail 1-2 minutes 	Connect an IoT device With AWS IoT 5 minutes 

▶ See more

Learn to build

Learn to deploy your solutions through step-by-step guides, labs, and videos. [See all](#)

Websites and Web Apps 3 videos, 3 tutorials, 3 labs 	Storage 3 videos, 3 tutorials, 3 labs 
Databases 3 videos, 3 tutorials, 3 labs 	DevOps 3 videos, 3 tutorials, 3 labs 
Machine Learning 3 videos, 3 tutorials, 3 labs 	Big Data 3 videos, 1 lab 

[Build with SDKs](#)

Access resources on the go

 Access the Management Console using the AWS Console Mobile App. [Learn more](#)

Explore AWS

Amazon SageMaker
Machine learning for every developer and data scientist. [Learn more](#)

AWS Marketplace
Find, buy, and deploy popular software products that run on AWS. [Learn more](#)

Visit AWS around the world at a Summit
AWS Global Summits bring the cloud computing community together to connect, collaborate, and learn about AWS. [Learn more](#)

Run Serverless Containers with AWS Fargate
AWS Fargate runs and scales your containers without having to manage servers or clusters. [Learn more](#)

Have feedback?

 Submit feedback to tell us about your experience with the AWS Management Console.

สมัครหรือทำการ Login เสร็จแล้วจะมีหน้าตาประมาณนี้ครับ หรือใครมีหน้าตาแบบไปก็ไม่ต้องตกใจนะครับ

ผมเลือกเป็น Singapore นะครับ ใกล้บ้านเราน้อย จากนั้นเลือกเป็น EC2 ครับ

The screenshot shows the AWS Home Page. At the top, there's a search bar with the placeholder "Example: Relational Database Service, database, RDS". Below it, a button says "All services". The main area is titled "Build a solution" with the sub-instruction "Get started with simple wizards and automated workflows.". There are four main sections with icons and descriptions:

- Launch a virtual machine** (With EC2, 2-3 minutes, icon of a CPU)
- Build a web app** (With Elastic Beanstalk, 6 minutes, icon of a cloud with a key)
- Build using virtual servers** (With Lightsail, 1-2 minutes, icon of a speaker)
- Connect an IoT device** (With AWS IoT, 5 minutes, icon of a person with a cloud)

On the right side, there are additional links:

- Explore AWS**
- Amazon SageMaker**: Machine learning for every developer & scientist. [Learn more](#)
- AWS Marketplace**: Find, buy, and deploy popular software run on AWS. [Learn more](#)
- Visit AWS around the world at a [Summit](#)**: AWS Global Summits bring the cloud community together to connect, collaborate about AWS. [Learn more](#)
- Run Serverless Containers with AWS Lambda**: AWS Fargate runs and scales your containers without having to manage servers or clusters. [Learn more](#)

เลือกเป็น free tier Ubuntu นะครับ ใช้ง่ายดี แต่ถ้าใครชำนาญตัวอื่นก็ตามสบายครับ ไม่ว่ากัน

The screenshot shows the "Choose an Amazon Machine Image (AMI)" step of the AWS Launch Wizard. The top navigation bar includes steps: 1. AMI, 2. Choose Instance Type, 3. Configure Instance, 4. Add Storage, 5. Add Tags, 6. Configure Security Group, 7. Review. A "Cancel and Exit" button is also present.

The list of AMIs includes:

- Amazon Linux AMI 2018.03.0 (HVM), SSD Volume Type - ami-05b3bcf7f311194b3**
Free tier eligible. The Amazon Linux AMI is an EBS-backed, AWS-supported image. The default image includes AWS command line tools, Python, Ruby, Perl, and Java. The repositories include Docker, PHP, MySQL, PostgreSQL, and other packages.
Root device type: ebs Virtualization type: hvm [Select](#) 64-bit (x86)
- Red Hat Enterprise Linux 7.5 (HVM), SSD Volume Type - ami-76144b0a**
Free tier eligible. Red Hat Enterprise Linux version 7.5 (HVM), EBS General Purpose (SSD) Volume Type.
Root device type: ebs Virtualization type: hvm [Select](#) 64-bit (x86)
- SUSE Linux Enterprise Server 15 (HVM), SSD Volume Type - ami-0920c364049458e86**
Free tier eligible. SUSE Linux Enterprise Server 15 (HVM), EBS General Purpose (SSD) Volume Type. Public Cloud, Advanced Systems Management, Web and Scripting, and Legacy modules enabled.
Root device type: ebs Virtualization type: hvm [Select](#) 64-bit (x86)
- Ubuntu Server 18.04 LTS (HVM), SSD Volume Type - ami-0dad20bd1b9c8c004**
Free tier eligible. Ubuntu Server 18.04 LTS (HVM), EBS General Purpose (SSD) Volume Type. Support available from Canonical (<http://www.ubuntu.com/cloud/services>).
Root device type: ebs Virtualization type: hvm [Select](#) 64-bit (x86)

At the bottom, a note says: "Are you launching a database instance? Try Amazon RDS." with a "Hide" link.

เลือกเป็นตัวนีนั่นคือ ตัวอื่นเสียตั้งค่าจะครับ ใช้ตัวนี้ไปก่อนดีแล้วค่อย upgrade ได้ ครับ ครับ ครับ ไม่โปร อย่าเพิ่งกด Review and Launch ให้กด Next เพื่อ Config ไป ก่อน

Step 2: Choose an Instance Type

Family	Type	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance	IPv6 Support
General purpose	t2.nano	1	0.5	EBS only	-	Low to Moderate	Yes
General purpose	t2.micro	1	1	EBS only	-	Low to Moderate	Yes
General purpose	t2.small	1	2	EBS only	-	Low to Moderate	Yes
General purpose	t2.medium	2	4	EBS only	-	Low to Moderate	Yes
General purpose	t2.large	2	8	EBS only	-	Low to Moderate	Yes
General purpose	t2.xlarge	4	16	EBS only	-	Moderate	Yes
General purpose	t2.2xlarge	8	32	EBS only	-	Moderate	Yes
General purpose	t3.nano	2	0.5	EBS only	Yes	Up to 5 Gigabit	Yes
General purpose	t3.micro	2	1	EBS only	Yes	Up to 5 Gigabit	Yes
General purpose	t3.small	2	2	EBS only	Yes	Up to 5 Gigabit	Yes
General purpose	t3.medium	2	4	EBS only	Yes	Up to 5 Gigabit	Yes

Cancel **Previous** **Review and Launch** **Next: Configure Instance Details**

กด Next ต่อไปครับ หน้านี้จะแสดงรายละเอียดเกี่ยวกับ Server ที่เราจะใช้ ดูได้แต่ อย่ามือเร็วไปเปลี่ยน นะคับ ถ้าไม่มีความรู้

Step 3: Configure Instance Details

Subnet: No preference (default subnet in any Availability Zone) **Create new VPC**

Auto-assign Public IP: Use subnet setting (Enable)

Placement group: Add instance to placement group

Capacity Reservation: Open **Create new Capacity Reservation**

IAM role: None **Create new IAM role**

Shutdown behavior: Stop

Enable termination protection: Protect against accidental termination

Monitoring: Enable CloudWatch detailed monitoring
Additional charges apply.

Tenancy: Shared - Run a shared hardware instance
Additional charges will apply for dedicated tenancy.

T2/T3 Unlimited: Enable
Additional charges may apply

Advanced Details

Cancel **Previous** **Review and Launch** **Next: Add Storage**

ส่วนนี้จะแสดงถึงว่าเราเก็บไฟล์ไว้ที่ไหน ขนาดเท่าไหร่ เราใช้ฟรีได้ถึง 30GB แต่ 8GB เพียงพอล่ะครับ กด Next ไป

Step 4: Add Storage

Your instance will be launched with the following storage device settings. You can attach additional EBS volumes and instance store volumes to your instance, or edit the settings of the root volume. You can also attach additional EBS volumes after launching an instance, but not instance store volumes. [Learn more](#) about storage options in Amazon EC2.

Volume Type	Device	Snapshot	Size (GiB)	Volume Type	IOPS	Throughput (MB/s)	Delete on Termination	Encrypted
Root	/dev/sda1	snap-0de5fb7f53112ca1d	8	General Purpose SSD (gp2)	100 / 3000	N/A	<input checked="" type="checkbox"/>	Not Encrypted

Add New Volume

Free tier eligible customers can get up to 30 GB of EBS General Purpose (SSD) or Magnetic storage. [Learn more](#) about free usage tier eligibility and usage restrictions.

Cancel Previous Review and Launch Next: Add Tags

มาถึงตอนนี้ให้ทำการ Add Tag เพื่อ ตั้งชื่อ Instance ของเราที่เราสร้างขึ้นครับ

Step 5: Add Tags

A tag consists of a case-sensitive key-value pair. For example, you could define a tag with key = Name and value = Webserver. A copy of a tag can be applied to volumes, instances or both. Tags will be applied to all instances and volumes. [Learn more](#) about tagging your Amazon EC2 resources.

Key (127 characters maximum)	Value (255 characters maximum)	Instances	Volumes
weblog	weblog	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Add another tag (Up to 50 tags maximum)

จากนั้นทำการเพิ่มเงื่อนไขใน Firewall ของเราให้ port 22 และ port 8081 ของเราเป็น public หรือ anyware นั่นเอง

Search : sg-029712d701adbe7db Add filter

Edit inbound rules

Type	Protocol	Port Range	Source	Description
HTTP	TCP	80	Custom 0.0.0.0/0	e.g. SSH for Admin Desktop
HTTP	TCP	80	Custom ::/0	e.g. SSH for Admin Desktop
SSH	TCP	22	Custom 0.0.0.0/0	e.g. SSH for Admin Desktop
SSH	TCP	22	Custom ::/0	e.g. SSH for Admin Desktop
Custom TCP	TCP	8081	Custom 0.0.0.0/0	e.g. SSH for Admin Desktop
Custom TCP	TCP	8081	Custom ::/0	e.g. SSH for Admin Desktop

Add Rule

NOTE: Any edits made on existing rules will result in the edited rule being deleted and a new rule created with the new details. This will cause traffic that depends on that rule to be dropped for a very brief period of time until the new rule can be created.

Cancel Save

ตอนนี้ port ของเรารวมไปซึ่งงานแล้วครับ มาถึงตรงนี้ก็ด **Launch** ได้เลยครับ

1. Choose AMI 2. Choose Instance Type 3. Configure Instance 4. Add Storage 5. Add Tags 6. Configure Security Group 7. Review

Step 7: Review Instance Launch

Instance Type	ECUs	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance
t2.micro	Variable	1	1	EBS only	-	Low to Moderate

▼ Security Groups [Edit security groups](#)

Security group name	Description
launch-wizard-2	launch-wizard-2 created 2019-03-20T14:57:29.823+07:00

Type	Protocol	Port Range	Source	Description
SSH	TCP	22	0.0.0.0/0	
SSH	TCP	22	::/0	
HTTP	TCP	80	0.0.0.0/0	
HTTP	TCP	80	::/0	

▶ Instance Details [Edit instance details](#)

▶ Storage [Edit storage](#)

▶ Tags [Edit tags](#)

Cancel Previous Launch

AWS จะแสดงข้อมูลเกี่ยวกับ service ที่เราใช้รวมถึง Firewall config และข้อมูลต่างๆ ให้เราครบ ทำการกด **Launch** ได้เลยครับ

The screenshot shows the AWS Launch Status page. At the top, there's a header with the AWS logo, navigation links for Services, Resource Groups, and a user icon. Below the header, the main content area has a green banner stating "Your instances are now launching" with a link to "View launch log". A blue banner below it says "Get notified of estimated charges" with a link to "Create billing alerts".

Launch Status

Your instances are now launching
The following instance launches have been initiated: [i-0d801a58adff7851](#) [View launch log](#)

Get notified of estimated charges
[Create billing alerts](#) to get an email notification when estimated charges on your AWS bill exceed an amount you define (for example, if you exceed the free usage tier).

How to connect to your instances
Your instances are launching, and it may take a few minutes until they are in the **running** state, when they will be ready for you to use. Usage hours on your new instances will start immediately and continue to accrue until you stop or terminate your instances.

Click [View Instances](#) to monitor your instances' status. Once your instances are in the **running** state, you can [connect](#) to them from the Instances screen. [Find out](#) how to connect to your instances.

Here are some helpful resources to get you started

- [How to connect to your Linux instance](#)
- [Amazon EC2: User Guide](#)
- [Learn about AWS Free Usage Tier](#)
- [Amazon EC2: Discussion Forum](#)

While your instances are launching you can also

- [Create status check alarms](#) to be notified when these instances fail status checks. (Additional charges may apply)
- [Create and attach additional EBS volumes](#) (Additional charges may apply)

เรียนรู้อย Server ของเราร่วมใช้งานแล้วครับ กดเลือกไปที่ EC2 Dashboard ครับ

The screenshot shows the AWS EC2 Dashboard. On the left, there's a sidebar with navigation links for EC2 Dashboard, Events, Tags, Reports, Limits, Instances (selected), Launch Templates, Spot Requests, Reserved Instances, Dedicated Hosts, Capacity Reservations, Images, AMIs, Bundle Tasks, Elastic Block Store, Volumes, Snapshots, Lifecycle Manager, Network & Security, Security Groups, Elastic IPs, and Placement Groups. The main content area shows a table of running instances:

Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS
GoodDevDB...	i-0493f1390d46db0a9	t2.micro	ap-southeast-1b	running	2/2 checks ...	None	ec2-13-229-2-
webblog	i-0d801a58adff7851	t2.micro	ap-southeast-1b	running	2/2 checks ...	None	ec2-3-1-195-

Below the table, detailed information for the selected instance (i-0d801a58adff7851) is shown:

- Availability zone:** ap-southeast-1b
- Security groups:** [launch-wizard-2](#), [view inbound rules](#), [view outbound rules](#)
- Scheduled events:** No scheduled events
- AMI ID:** [ubuntu/images/hvm-ssd/ubuntu-bionic-18.04-amd64-server-20190212.1 \(ami-0dad20bd1b9c8c004\)](#)
- Platform:** -
- IAM role:** -
- Key pair name:** sshOnGooddevOnAWS
- Owner:** 184818151314
- Launch time:** March 20, 2019 at 3:30:14 PM UTC+7 (less than one hour)
- Termination protection:** False
- Lifecycle:** normal
- Monitoring:** basic
- Alarm status:** None
- Private IPs:** 172.31.22.223
- Secondary private IPs:** -
- VPC ID:** vpc-24250c43
- Subnet ID:** subnet-ddeefcba
- Network interfaces:** eth0
- Source/dest. check:** True
- T2/T3 Unlimited:** Disabled
- EBS-optimized:** False
- Root device type:** ebs
- Root device:** /dev/sda1
- Block devices:** /dev/sda1
- Elastic Graphics ID:** -

แล้วทำการเลือกไปที่ Running Instances

EC2 Dashboard

- Events
- Tags
- Reports
- Limits

INSTANCES

- Instances
- Launch Templates
- Spot Requests
- Reserved Instances
- Dedicated Hosts
- Capacity Reservations

IMAGES

- AMIs
- Bundle Tasks

ELASTIC BLOCK STORE

- Volumes
- Snapshots
- Lifecycle Manager

NETWORK & SECURITY

- Security Groups

Resources

You are using the following Amazon EC2 resources in the Asia Pacific (Singapore) region:

2 Running Instances	0 Elastic IPs
0 Dedicated Hosts	0 Snapshots
2 Volumes	0 Load Balancers
1 Key Pairs	3 Security Groups
0 Placement Groups	

Learn more about the latest in AWS Compute from AWS re:Invent by viewing the [EC2 Videos](#).

Create Instance

To start using Amazon EC2 you will want to launch a virtual server, known as an Amazon EC2 instance.

Service Health

Scheduled Events

AWS Marketplace

เลือกที่ weblog ของเราตอนเรา add tag ไว้ครับ ครับ และทำการกด connect instance ของเราที่จะใช้งานครับ

EC2 Dashboard

- Events
- Tags
- Reports
- Limits

INSTANCES

Instances

- Launch Templates
- Spot Requests
- Reserved Instances
- Dedicated Hosts
- Capacity Reservations

IMAGES

- AMIs
- Bundle Tasks

ELASTIC BLOCK STORE

- Volumes
- Snapshots
- Lifecycle Manager

NETWORK & SECURITY

- Security Groups
- Elastic IPs
- Placement Groups

Services ▾ **Resource Groups** ▾

Actions ▾

Connect

Filter by tags and attributes or search by keyword

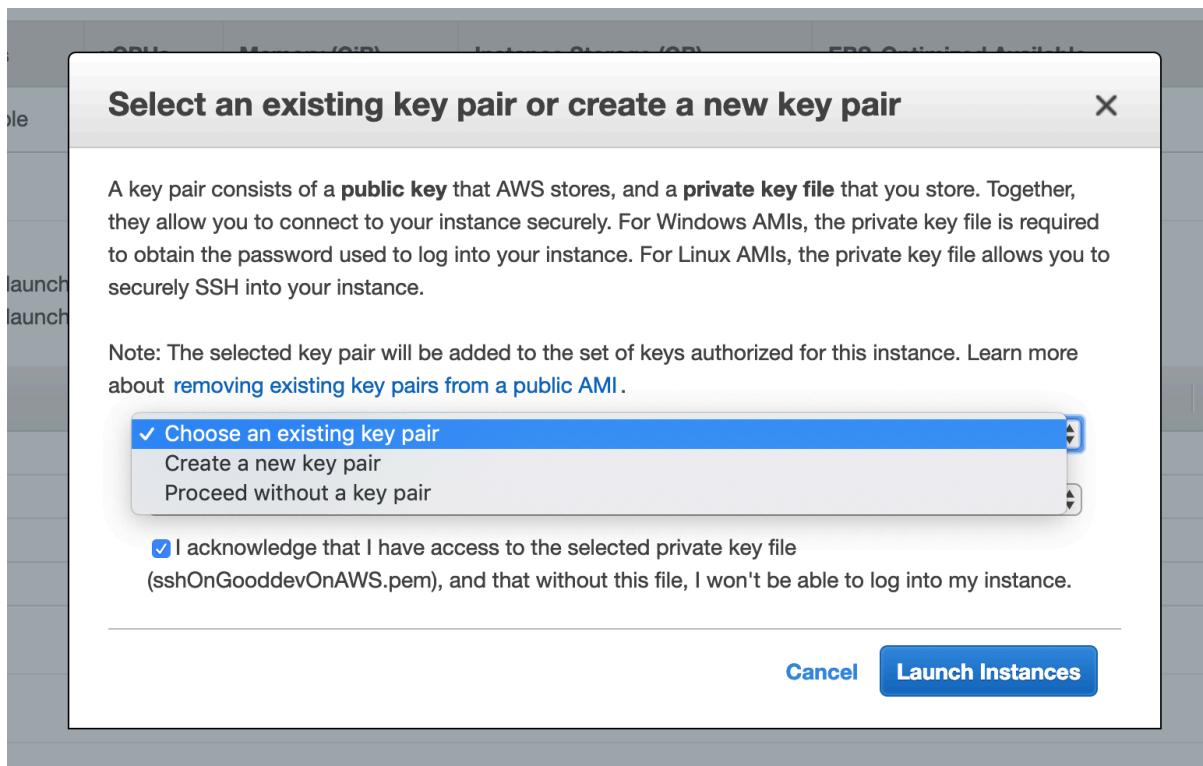
Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status	Public DNS (IPv4)
GoodDevDB...	i-0493f1390d46db0a9	t2.micro	ap-southeast-1b	running	2/2 checks ...	None	ec2-13-229-250-
webblog	i-0d801a58adff7851	t2.micro	ap-southeast-1b	running	2/2 checks ...	None	ec2-3-1-195-189-

Instance: i-0d801a58adff7851 (webblog) Public DNS: ec2-3-1-195-189.ap-southeast-1.compute.amazonaws.com

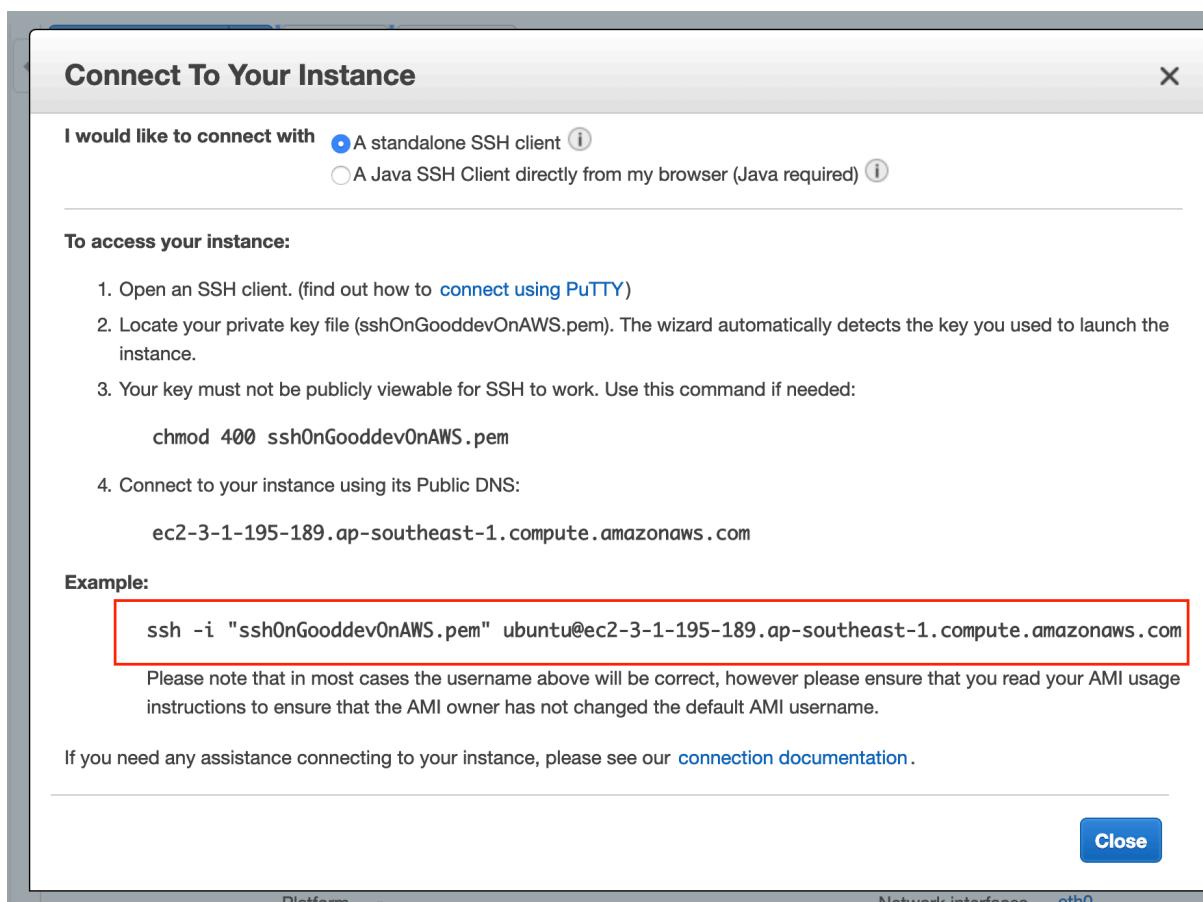
Description **Status Checks** **Monitoring** **Tags**

Instance ID	i-0d801a58adff7851	Public DNS (IPv4)	ec2-3-1-195-189.ap-southeast-1.compute.amazonaws.com
Instance state	running	IPv4 Public IP	3.1.195.189
Instance type	t2.micro	IPv6 IPs	-
Elastic IPs		Private DNS	ip-172-31-22-223.ap-southeast-1.compute.internal
Availability zone	ap-southeast-1b	Private IPs	172.31.22.223
Security groups	launch-wizard-2, view inbound rules, view outbound rules	VPC ID	vpc-24250c43
Scheduled events	No scheduled events	Subnet ID	subnet-ddeefcba
AMI ID	ubuntu/images/hvm-ssd/ubuntu-bionic-18.04-amd64-server-20190212.1 (ami-0dad20bd1b9c8c004)	Network interfaces	eth0
Platform	-		

ถ้าเรายังไม่เคยใช้งานมาก่อนเค้าจะให้เราสร้าง key เพื่อทำการ connect เข้าใช้งานระบบครับ ทำการสร้างแล้ว download มาไว้ในเครื่องเราครับ จากนั้นเอาไว้ใน drive d:/ หรือ c:/



ทำการกด connect แล้วทำการ copy ssh command มา ก่อนครับ



จากนั้นทำการเปิด Terminal ของเราขึ้นมาแล้วทำการ เปิดไปที่ตำแหน่งที่เรา .pem หรือ key file ที่เราทำการ download มาในขั้นตอนก่อนหน้านี้

จากนั้นทำการใส่ คำสั่ง

```
chmod 400 sshOnGooddevOnAWS.pem
```

หรือชื่อไฟล์อื่นตามที่ท่านได้ตั้งไว้ แล้วทำการ พิมพ์คำสั่ง เพื่อทำการ connect ไปที่ ec2 ดังนี้

```
ssh -i "sshOnGooddevOnAWS.pem" ubuntu@ec2-3-1-195-189.ap-southeast-1.compute.amazonaws.com
```

จากนั้นทำการ พิมพ์คำสั่ง

```
sudo apt-get update  
sudo apt install node  
sudo apt install npm
```

จากนั้นทำการตรวจสอบด้วย

```
npm -version  
node -version
```

ถ้าติดตั้งสำเร็จจะแสดง version ออกมาก

จากนั้น

git clone <https://github.com/kornchayapon/nvwebblog.git> มาไว้ที่ server ของเรา จากนั้นเข้าไปที่ server ของเรา และทำการ uninstall เอา sqlite3 และ sequelize ออกก่อน และทำการติดตั้งใหม่อีกครั้ง

```
npm uninstall sqlite3  
npm uninstall sequelize
```

```
npm install -save sqlite3  
npm install -save sequelize
```

จากนั้นทำการรัน node src/app.js ที่ server ของเรา มิ่งโก้ Back End ของเรา แล้วครับ

```
kornchayapon — ubuntu@ip-172-31-22-223: ~/nvwebblog/server — ssh -i sshOnGooddevOnAWS.pem ubuntu@ec2-3-1-195-189.ap-s...  
~/noderootbook/nvwebblog — bash ..ubuntu@ec2-3-1-195-189.ap-southeast-1.compute.amazonaws.com  
at Promise._settlePromise0 (/home/ubuntu/nvwebblog/server/node_modules/bluebird/js/release/promise.js:614:10)  
at Promise._settlePromises (/home/ubuntu/nvwebblog/server/node_modules/bluebird/js/release/promise.js:694:18)  
at _drainQueueStep (/home/ubuntu/nvwebblog/server/node_modules/bluebird/js/release/async.js:138:12)  
at _drainQueue (/home/ubuntu/nvwebblog/server/node_modules/bluebird/js/release/async.js:131:9)  
at Async._drainQueues (/home/ubuntu/nvwebblog/server/node_modules/bluebird/js/release/async.js:147:5)  
at Immediate.Async.drainQueues [as _onImmediate] (/home/ubuntu/nvwebblog/server/node_modules/bluebird/js/release/async.js:144:14)  
at runCallback (timers.js:794:20)  
at tryOnImmediate (timers.js:752:5)  
at processImmediate [as _immediateCallback] (timers.js:729:5)  
ubuntu@ip-172-31-22-223:~/nvwebblog/server$ ls  
node_modules nvwebblog-db-backup.sqlite nvwebblog-db.sqlite package-lock.json package.json public src  
ubuntu@ip-172-31-22-223:~/nvwebblog/server$ cd src/config  
ubuntu@ip-172-31-22-223:~/nvwebblog/server/src/config$ ls  
config.js  
ubuntu@ip-172-31-22-223:~/nvwebblog/server/src/config$ sudo nano config.js  
ubuntu@ip-172-31-22-223:~/nvwebblog/server/src/config$ cd ..  
ubuntu@ip-172-31-22-223:~/nvwebblog/server/src$ cd ..  
ubuntu@ip-172-31-22-223:~/nvwebblog/server$ node src/app.js  
sequelize deprecated String based operators are now deprecated. Please use Symbol based operators for better security, read more at http://docs.sequelizejs.com/manual/tutorial/querying.html#operators node_modules/sequelize/lib/sequelize.js:242:13  
Executing (default): CREATE TABLE IF NOT EXISTS `Blogs` (`id` INTEGER PRIMARY KEY AUTOINCREMENT, `title` VARCHAR(255), `thumbnail` VARCHAR(255), `pictures` VARCHAR(255), `content` TEXT, `category` VARCHAR(255), `status` VARCHAR(255), `createdAt` DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL);  
Executing (default): PRAGMA INDEX_LIST(`Blogs`)  
Executing (default): CREATE TABLE IF NOT EXISTS `Books` (`id` INTEGER PRIMARY KEY AUTOINCREMENT, `title` VARCHAR(255), `thumbnail` VARCHAR(255), `pictures` VARCHAR(255), `content` TEXT, `category` VARCHAR(255), `status` VARCHAR(255), `prices` VARCHAR(255), `createdAt` DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL);  
Executing (default): PRAGMA INDEX_LIST(`Books`)  
Executing (default): CREATE TABLE IF NOT EXISTS `Buys` (`id` INTEGER PRIMARY KEY AUTOINCREMENT, `bookid` VARCHAR(255), `userid` VARCHAR(255), `qty` VARCHAR(255), `email` VARCHAR(255), `clientStatus` VARCHAR(255), `shopStatus` VARCHAR(255), `booktitle` VARCHAR(255), `username` VARCHAR(255), `userlastname` VARCHAR(255), `prices` VARCHAR(255), `createdAt` DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL);  
Executing (default): PRAGMA INDEX_LIST(`Buys`)  
Executing (default): CREATE TABLE IF NOT EXISTS `Comments` (`id` INTEGER PRIMARY KEY AUTOINCREMENT, `blogId` VARCHAR(255), `content` TEXT, `userId` VARCHAR(255), `createdAt` DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL);  
Executing (default): PRAGMA INDEX_LIST(`Comments`)  
Executing (default): CREATE TABLE IF NOT EXISTS `Users` (`id` INTEGER PRIMARY KEY AUTOINCREMENT, `email` VARCHAR(255) UNIQUE, `password` VARCHAR(255), `name` VARCHAR(255), `lastname` VARCHAR(255), `status` VARCHAR(255), `type` VARCHAR(255), `address` TEXT, `createdAt` DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL);  
Executing (default): PRAGMA INDEX_LIST(`Users`)  
Executing (default): PRAGMA INDEX_INFO(`sqlite_autoindex_Users_1`)  
Server running on 8081  
-----> search key: undefined  
Executing (default): SELECT `id`, `blogId`, `comment`, `userId`, `createdAt`, `updatedAt` FROM `Comments` AS `Comment` ORDER BY `Comment`.`createdAt` DESC;
```

จากนั้นเราจะทำการเปิดไปที่ url ของเรากันครับ

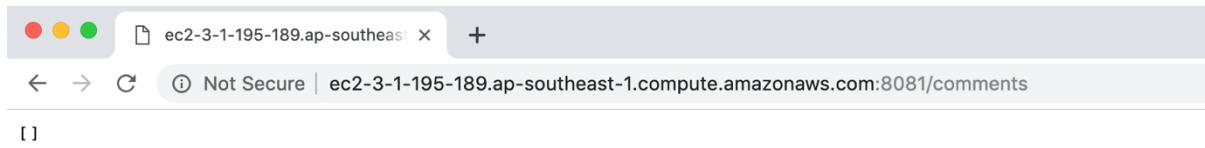
Instance: i-0d801a58adfff7851 (webblog) Public DNS: ec2-3-1-195-189.ap-southeast-1.compute.amazonaws.com

Description	Status Checks	Monitoring	Tags
Instance ID	i-0d801a58adfff7851		Public DNS (IPv4) ec2-3-1-195-189.ap-southeast-1.compute.amazonaws.com
Instance state	running		IPv4 Public IP 3.1.195.189
Instance type	t2.micro		IPv6 IPs -
Elastic IPs			Private DNS ip-172-31-22-223.ap-southeast-1.compute.internal
Availability zone	ap-southeast-1b		Private IPs 172.31.22.223
Security groups	launch-wizard-2. view inbound rules. view outbound rules		Secondary private IPs
Scheduled events	No scheduled events		VPC ID vpc-24250c43
AMI ID	ubuntu/images/hvm-ssd/ubuntu-bionic-18.04-amd64-server-20190212.1 (ami-0dad20bd1b9c8c004)		Subnet ID subnet-ddeefcba
Platform	-		Network interfaces eth0

อยู่ตรงนี้ url ของเรา ผ่านทำการทดสอบเปิดที่ link ของผม คือ

<http://ec2-3-1-195-189.ap-southeast-1.compute.amazonaws.com:8081/comments>

เห็นมั้ยครับ คล้ายในเครื่องเราเลยครับ ถ้าเราจะ domain ตัวเองก็สามารถศึกษาเกี่ยวกับ amazon aws เพิ่มเติมเองได้ครับ สังเกตว่า server ของเราต้องกลับมาเป็นค่า array ว่าง ๆ เพราะเรายังไม่มีข้อมูลนั้นเองครับ แสดงว่าส่วน backend นั้น work และเพราะถ้าไม่ work จะ error เป็น 503 ครับ



กด ctrl เพื่อหยุดจากรัน node server ของเรา จากนั้นทำการ ใช้ nohup มาช่วยเพื่อรันแบบ backend process จะได้อ览 shell ของเราไปรัน client

ทำการ start server ใหม่ด้วยคำสั่งดังนี้ครับ

```
nohup node src/app.js
```

แล้วกด ctrl + c สั่งเกตว่า server ของเราอยู่เช่นเดิมครับ

ทำการรัน Front End กันครับ

เราเริ่มทำการ เข้าไป folder client ของเราแล้วใช้ คำสั่ง

```
npm run build
```

เพื่อทำการ build โปรเจคของเรากันครับ จากนั้นทำการ commit ขึ้น server และที่ server ec2 ของเรา กันแล้วใช้คำสั่ง git pull เพื่อดึงไฟล์ของเรา

ทำการติดตั้ง serve เพื่อทำการรัน production vuejs ของเรา

```
sudo npm install -g serve
```

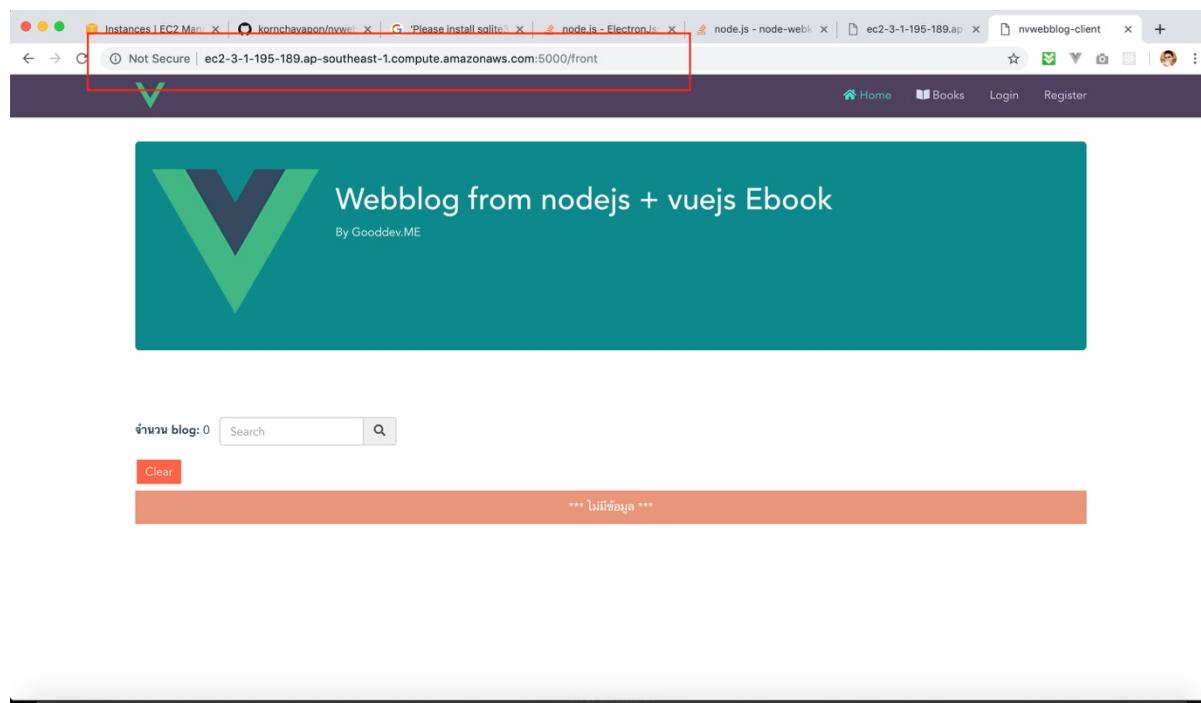
จากนั้นทำการรันด้วยคำสั่ง

```
nohup serve -s dist
```

จากนั้นเรียกไปที่

<http://ec2-3-1-195-189.ap-southeast-1.compute.amazonaws.com:5000>

บังゴก เว็บบล็อกของเราไปรันอยู่ server เรียบร้อยแล้วครับ ทำการเพิ่มข้อมูลและทำการทดสอบกันดูนะครับ ผ่านทดสอบแล้วใช้งานได้ดีเลยครับ เพียงแต่หน้าแรก capture มาลงรูปที่มีอยู่แล้วเท่านั้นครับ



มาถึงตอนนี้ทุกคนน่าจะสามารถเขียนเว็บไซต์ แบบเว็บบล็อก พร้อมทั้งมีไอเดีย และเข้าใจ concept ของ vuejs และการใช้งาน javascript framework กัน หมวดแล้วนะครับ ขอให้พยายามนำความรู้ที่ได้ไปใช้เขียนบ่อย ทำงานเยอะ ๆ และจะเก่งขึ้นตามลำดับหากติดปัญหา ในหนังสือส่วนไหน สามารถสอบถามได้ทันที

จบเรื่องนี้