SE-ED COMPUTER BOOK developer

พัฒนา Application ด้วย React และ React Native

สร้าง Web App และ Mobile App (iOS/Android)
โดยใช้ JavaScript เป็นหลักเพียงภาษาเดียว

สามารถทดสอบแอป
บนระบบ Windows
ได้ทั้ง iOS และ Android

- React สำหรับพัฒนา Web App ด้วย JavaScript และ JSX
- React Native สำหรับพัฒนา Mobile App ทั้ง iOS และ Android
- REST API สำหรับรับส่งข้อมูลระหว่างฝั่ง Local และ Server

DOWNLOAD
ดาวน์โหลดโค้ดหนังสือเล่มนี้ได้ที่
http://www.developerthai.com

บัญชา ปะสีละเตสัง
ผู้เขียนหนังสือขายดีระดับ Best Seller
ด้าน Programming

ซีเอ็ด

พัฒนา Application ด้วย React และ React Native

โดย บัญชา ปะสีละเตสัง

ข้อมูลทางบรรณานุกรมของหอสมุดแห่งชาติ

บัญชา ปะสีละเตสัง.

พัฒนา Application ด้วย React และ React Native.--กรุงเทพฯ : ซีเอ็ดยูเคชั่น, 2565.
404 หน้า.
1. แอพพลิเคชั่น. 2. ซอฟต์แวร์--การพัฒนา.
I. ชื่อเรื่อง.

005.268

Barcode (e-book) : 9786160843237

ผลิตและจัดจำหน่ายโดย

บริษัท ซีเอ็ดยูเคชั่น จำกัด (มหาชน)
SE-EDUCATION PUBLIC COMPANY LIMITED

เลขที่ 1858/87-90 ถนนเทพรัตน แขวงบางนาใต้ เขตบางนา กรุงเทพฯ 10260
โทรศัพท์ 0-2826-8000

[หากมีคำแนะนำหรือติชม สามารถติดต่อได้ที่ comment@se-ed.com]

คำนำ

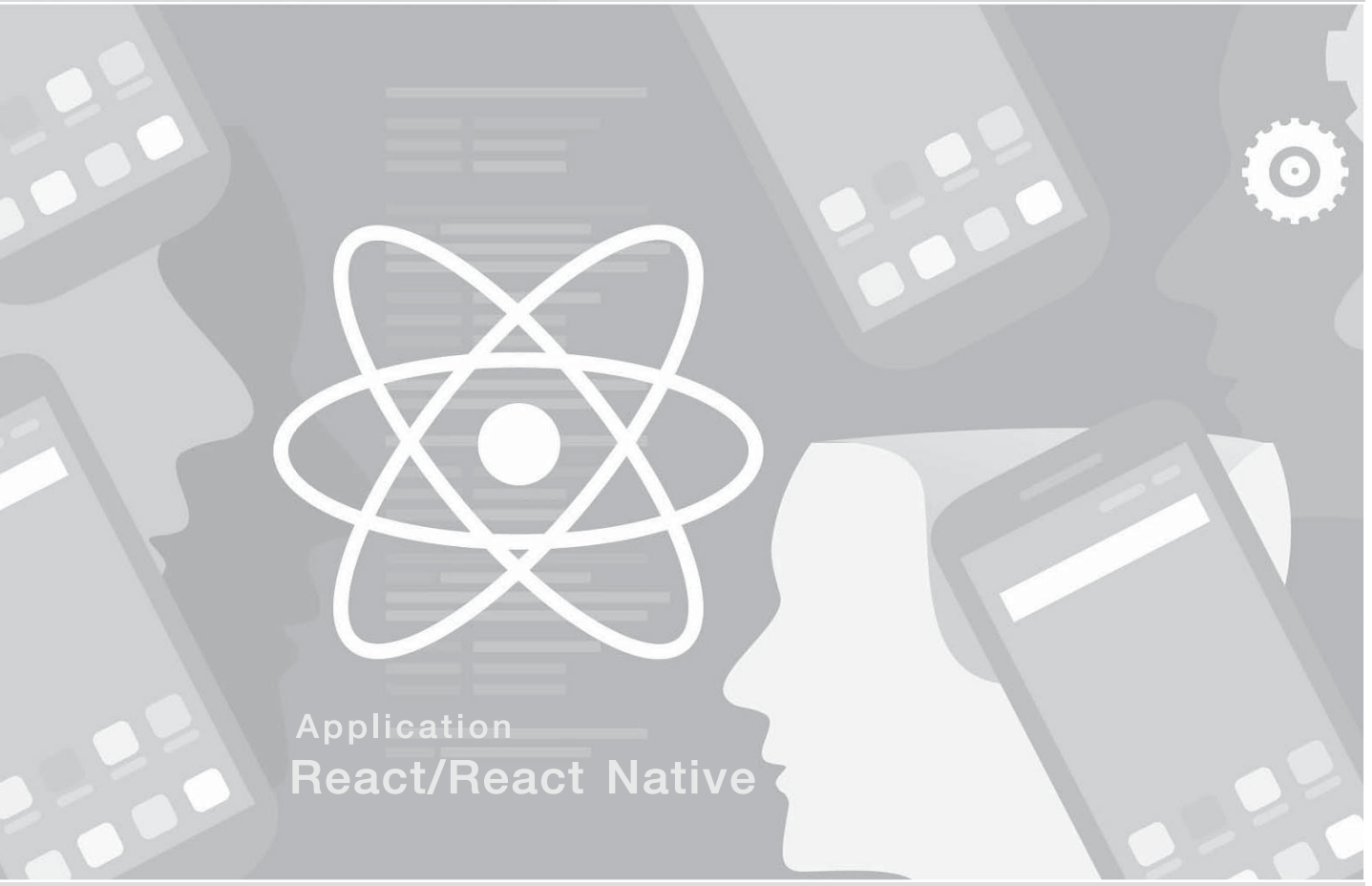
ในปัจจุบันนี้ มีอุปกรณ์ไอทีให้เลือกใช้งานในหลายรูปแบบไม่ว่าจะเป็น โน้ตบุ๊ก สมาร์ตโฟน หรือแท็บเล็ต ซึ่งอุปกรณ์เหล่านี้ อาจทำงานบนระบบปฏิบัติการหรือแพลตฟอร์ม (Platform) ที่แตกต่างกัน ดังนั้น เราจึงต้องพัฒนาแอปสำหรับใช้งานบนแพลตฟอร์มนั้นโดยตรง เช่น สร้าง Web App เพื่อเปิดด้วยเว็บเบราว์เซอร์ หรือพัฒนา Mobile App สำหรับ iOS และ Android เป็นต้น แต่เนื่องจากวิธีการสร้างแอปในแต่ละแพลตฟอร์มจะต้องใช้เครื่องมือและภาษาในการเขียนโค้ดที่แตกต่างกัน จึงสร้างความยุ่งยากให้แก่นักพัฒนาที่จะเรียนรู้ให้ครบทั้งหมดจนถึงระดับที่สามารถนำไปใช้งานจริงได้ รวมถึงการสร้างแอปแบบแยกแต่ละแพลตฟอร์มก็อาจต้องใช้เวลาอีกด้วย

จากเหตุผลที่กล่าวมา ทำให้นักพัฒนาหันมาให้ความสนใจเครื่องมือในการสร้างแอปแบบผสมผสาน (Hybrid) กันมากขึ้น โดยจะสร้างแอปขึ้นมาเพียงชุดเดียว แล้วสามารถแปลงไปเป็น Web App และ Mobile App สำหรับทั้งระบบ iOS และ Android ได้ทันที ซึ่งเครื่องมือการพัฒนาแอปในลักษณะดังกล่าวที่ได้รับความนิยมสูงสุดก็คือ React/React Native ที่มีข้อได้เปรียบคือ ใช้ JavaScript เป็นหลักเพียงภาษาเดียวในการเขียนโค้ด ซึ่งนักพัฒนาส่วนใหญ่ต่างก็คุ้นเคยกันดีอยู่แล้ว ดังนั้น จึงสามารถเรียนรู้ได้ในระยะเวลาอันสั้น รวมถึงมีไลบรารีที่สนับสนุนการทำงานอย่างมากมาย และแอปที่ได้ก็สามารถทำงานได้อย่างมีประสิทธิภาพในทุกแพลตฟอร์ม

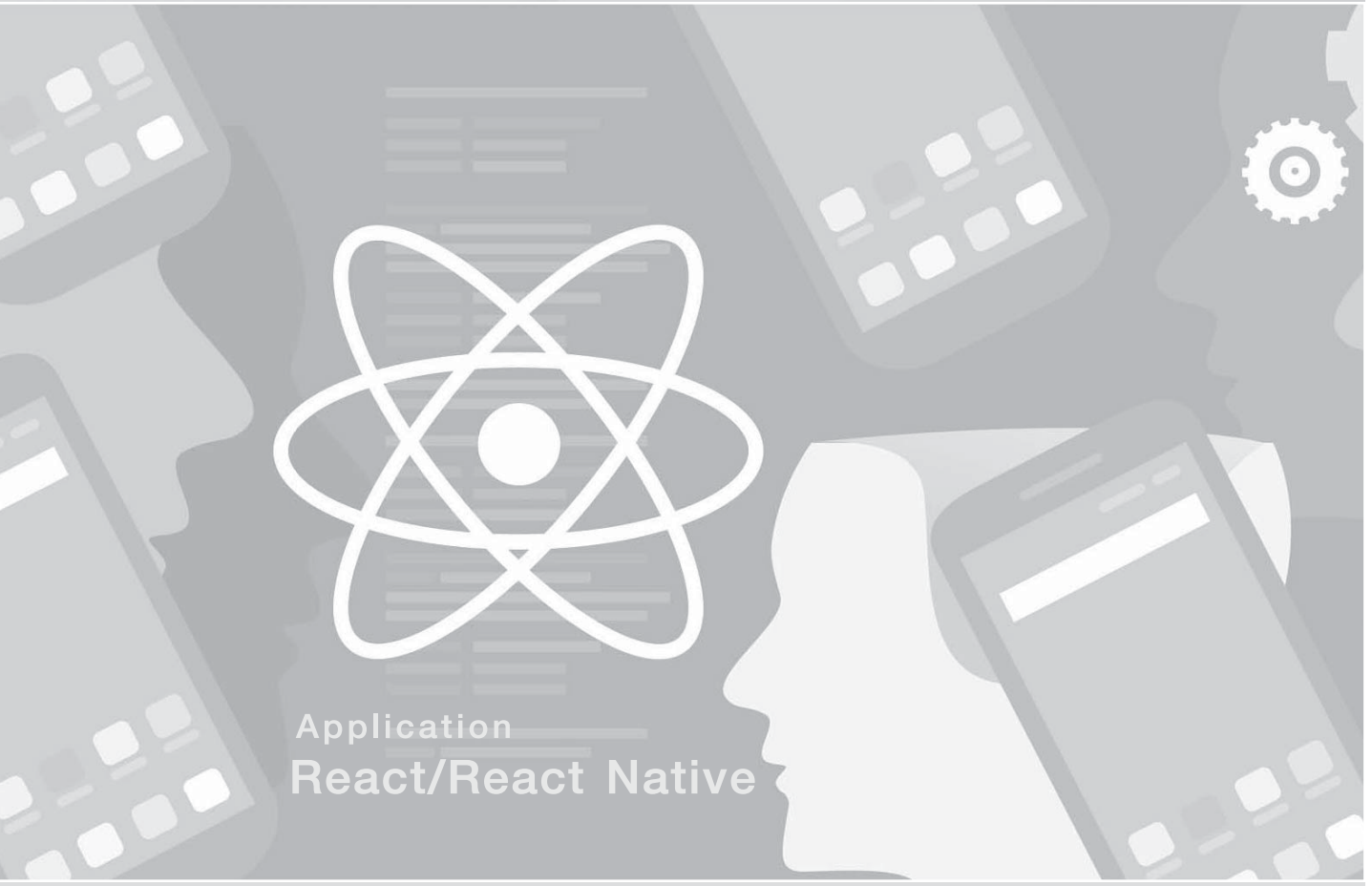
หนังสือเล่มนี้ได้จัดทำขึ้นเพื่อรวบรวมเนื้อหาเกี่ยวกับการพัฒนา Web App และ Mobile App สำหรับทั้ง iOS และ Android ด้วย React/React Native ตั้งแต่ขั้นพื้นฐานจนถึงระดับการใช้งานจริง ดังนั้น จึงหวังว่าผู้อ่านจะได้รับความรู้ที่ครบสมบูรณ์ พร้อมที่จะนำไปพัฒนาแอปตามต้องการ

บัญชา ปะสีละเตสัง

banchar.pa@yahoo.com
facebook.com/developerthai
http://www.developerthai.com


Application
React/React Native

สารบัญ


Application
React/React Native

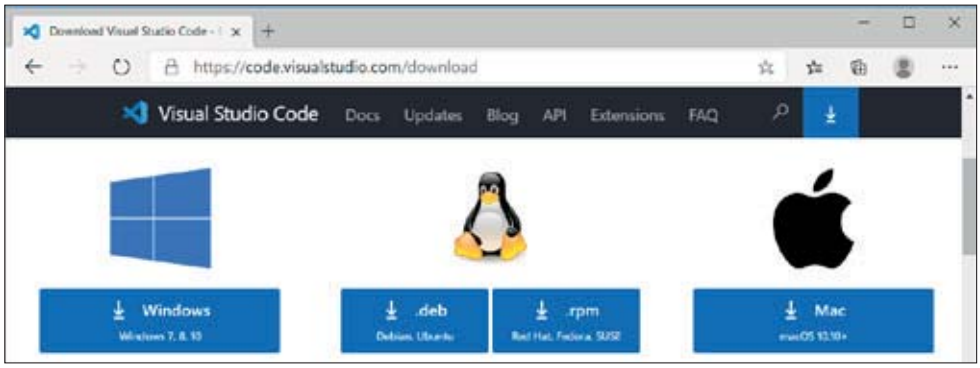
การติดตั้งและพื้นฐานของ React

1

React เป็นไลบรารีหรือเฟรมเวิร์กของภาษา JavaScript ที่ได้รับความนิยมสูงสุดในปัจจุบัน โดยเนื้อหาในบทนี้ เราจะกล่าวถึงการติดตั้งเครื่องมือที่จำเป็นทั้งหมดสำหรับการใช้ React จากนั้น ก็จะเริ่มต้นเรียนรู้วิธีการเขียนโค้ด HTML และ CSS ตามแนวทางของ React หรือที่เรียกว่า JSX รวมถึงลักษณะปลีกย่อยอื่นๆ ที่เราควรรู้จักในเบื้องต้น เพื่อเป็นพื้นฐานสำหรับการพัฒนา Web Application ที่จะกล่าวถึงในบทต่อๆ ไป

การใช้ VS Code

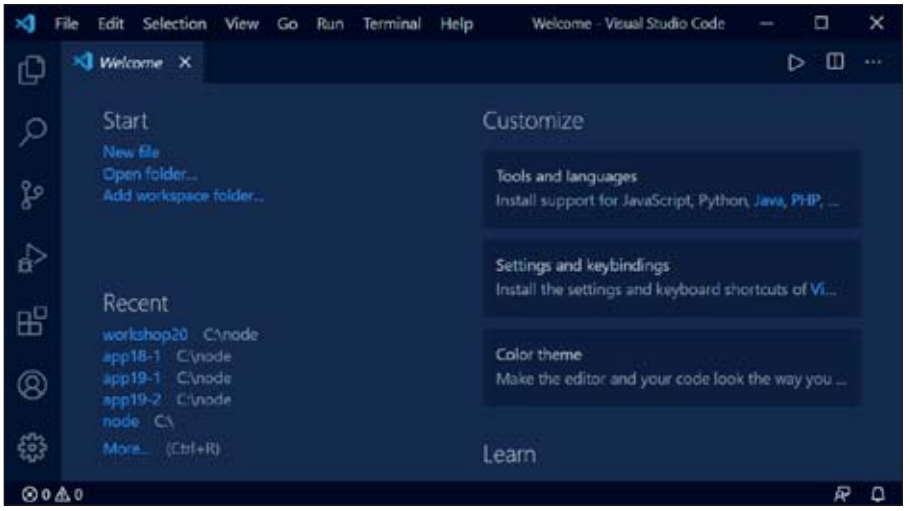
เครื่องมือในการเขียนและทดสอบโค้ดของ React ที่ได้รับความนิยมสูงสุดในปัจจุบันก็คือ Visual Studio Code (VS Code) ซึ่งเป็นโปรแกรมประเภท IDE (Integrated Development Environment) ที่มีลักษณะโดดเด่นหลายประการ เช่น ให้ใช้งานฟรี รองรับการเขียนโค้ดได้หลายภาษา และที่สำคัญคือ มีส่วนเสริมการทำงานให้เลือกใช้มากมาย สำหรับการติดตั้ง VS Code เราสามารถดาวน์โหลดชุดติดตั้งได้จากเว็บไซต์ https://code.visualstudio.com/download



หลังดาวน์โหลดเสร็จ ก็ดับเบิลคลิกไฟล์ติดตั้ง ซึ่งขั้นตอนต่างๆ ก็ทำเหมือนกับโปรแกรมทั่วไป และเนื่องจาก VS Code เป็น IDE สำหรับการเขียนโค้ดทั่วไปที่ไม่เจาะจงว่าเราต้องใช้ภาษาใด ดังนั้น เครื่องมือและตัวช่วยต่างๆ ก็เป็นสิ่งที่สามารถนำไปใช้ได้กับภาษาคอมพิวเตอร์เกือบทั้งหมด โดยลักษณะที่น่าสนใจซึ่งเราอาจได้ใช้งานในบางครั้ง มีดังนี้

หน้าจอ Welcome

ปกติแล้ว เมื่อเราเปิดเข้าสู่ VS Code จะปรากฏหน้าจอ Welcome โดยอัตโนมัติ ดังภาพ



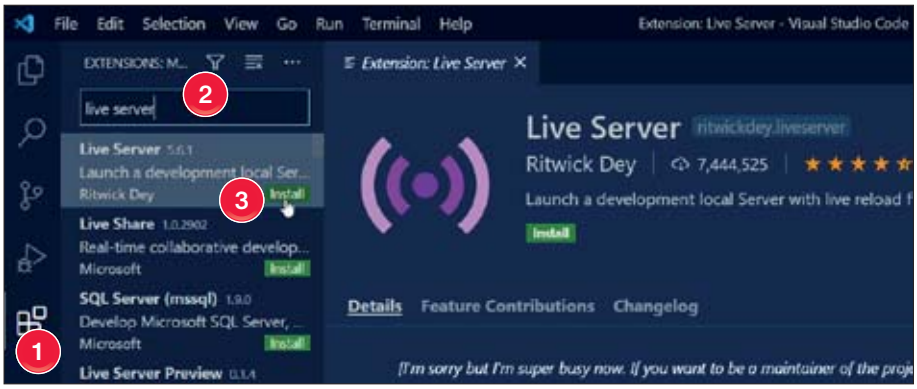
สิ่งที่น่าสนใจของหน้าจอ Welcome เช่น ทางลัดในการเปิดโฟลเดอร์เป้าหมายได้รวดเร็วขึ้น หรือการแจ้งข้อมูลข่าวสาร รวมถึงการอัปเดตต่างๆ แต่หากหน้าจอ Welcome ไม่ปรากฏ เช่น เราอาจคลิกปิดไปแล้ว หรือตั้งค่าไม่ให้แสดงอีก แต่ต้องการแสดงอีกครั้ง ก็เลือกที่เมนู **Help > Welcome**

การติดตั้งส่วนขยายของ VS Code

การเขียนโค้ดในบางกรณี เราอาจต้องใช้ส่วนขยาย (Extension) สำหรับการทำงานร่วมกับภาษานั้นๆ ซึ่งเราสามารถตรวจสอบและดาวน์โหลดส่วนขยายมาใช้งานได้ดังนี้

1. ในขณะนั้น ต้องเชื่อมต่อกับอินเทอร์เน็ต แล้วคลิกที่ไอคอน Extensions ที่ขอบด้านซ้าย (ดูภาพถัดไป) หรือเลือกที่เมนู **View > Extensions**
2. ใส่คีย์เวิร์ดลงไป หลังจากนั้นจะปรากฏรายชื่อส่วนขยายที่อาจเกี่ยวข้องกับคีย์เวิร์ด

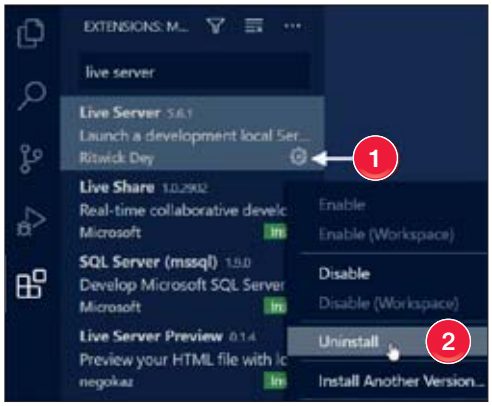
3. ถ้าเราต้องการติดตั้งส่วนขยายใด ก็คลิกปุ่ม Install บนรายการนั้น หรือหากต้องการดูรายละเอียดเพิ่มเติมเกี่ยวกับส่วนขยายใด ก็คลิกที่รายการดังกล่าว ดังภาพ



4. ส่วนขยายใดที่ถูกติดตั้งเอาไว้แล้ว จะมีรูปไอคอน Manage ที่มุมล่างขวาของรายการนั้น ทั้งนี้หากเราต้องการลบส่วนขยายนั้นทิ้งไป ก็ทำดังนี้

   (1) คลิกที่รูปไอคอนดังกล่าว

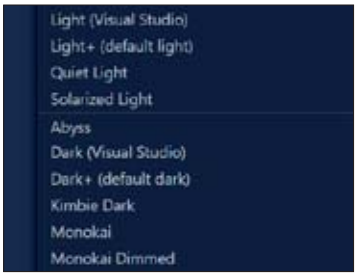
   (2) เลือกที่ Uninstall



การเปลี่ยนรูปแบบสีของ VS Code

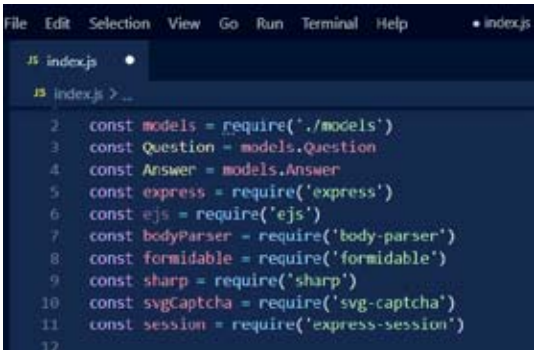
รูปแบบสีหรือ Color Theme ที่ VS Code เลือกไว้ล่วงหน้าจะเป็นสีดำ (Dark) แต่หากเราไม่ถูกใจสิ่งนี้ VS Code ก็มีรูปแบบสีมาให้เลือกบางส่วน ซึ่งขั้นตอนคือ

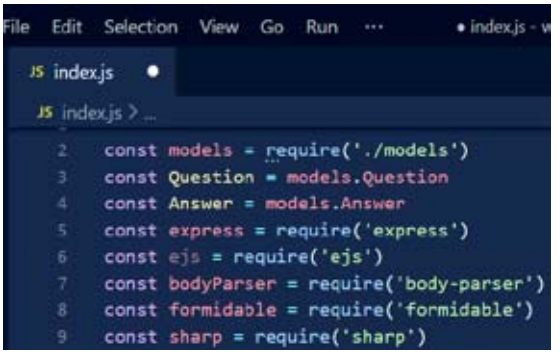
1. คลิกที่ไอคอน Manage ตรงมุมล่างซ้าย
2. เลือกเมนู Color Theme แล้วเลือกรูปแบบสีที่ต้องการ (กรณีของผู้เขียนจะเลือก Tomorrow Night Blue)



การย่อและขยายขนาด (Zoom In/Out)

ขนาดปกติของฟอนต์และไอคอนที่ VS Code กำหนดไว้ล่วงหน้า จะเป็นดังภาพถัดไป (ซ้าย)





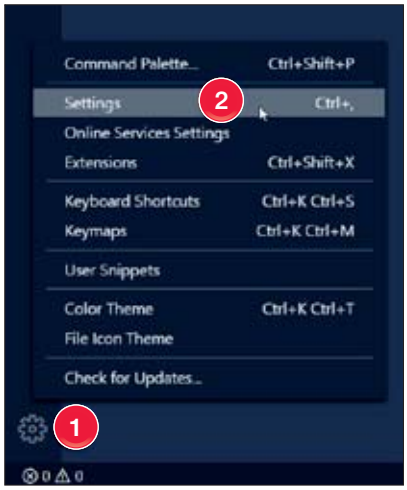
หากเราต้องการให้ฟอนต์และไอคอนมีขนาดใหญ่อีก 1 ระดับ ก็เลือกที่เมนู **View > Appearance > Zoom In** หรือกด <Ctrl => และถ้าทำเช่นนี้อีก ขนาดจะเพิ่มขึ้นอีก 1 ระดับ ดังภาพที่แล้วด้านขวามือ แต่หากเราต้องการให้ขนาดของฟอนต์และไอคอนเล็กลง 1 ระดับ ก็เลือกที่เมนู **View > Appearance > Zoom Out** หรือกด <Ctrl -> และถ้าทำเช่นนี้อีก ขนาดจะลดลงอีก 1 ระดับ

การกำหนดรูปแบบฟอนต์ของ Code Editor

หากเราต้องการปรับเปลี่ยนฟอนต์ของ Code Editor ก็ทำตามขั้นตอนดังนี้

1. คลิกที่ไอคอน Manage ตรงมุมล่างซ้าย
2. เลือกที่เมนู Settings
3. เลือกแท็บ User
4. คลิกที่ Text Editor
5. คลิกเมนู Font

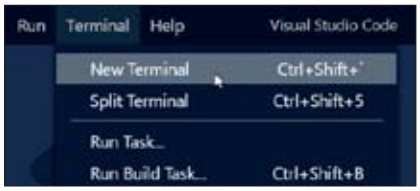
จากนั้นก็กำหนดรูปแบบฟอนต์ตามที่เราต้องการ เช่น Font Family (ชนิดฟอนต์), Font Size และ Font Weight เป็นต้น



การใช้ Terminal ของ VS Code

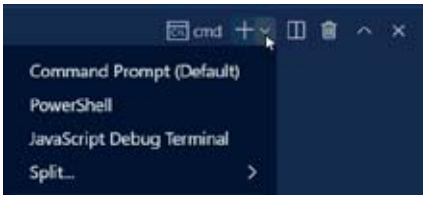
การทดสอบ React บน VS Code มีบางกรณีที่เราต้องพิมพ์คำสั่งผ่าน Command Line หรือ Terminal เพื่อดำเนินการต่างๆ เช่น การสร้างแอปของ React การเข้าและออกจากไดเร็กทอรี การติดตั้งโมดูล การรันทดสอบแอป เป็นต้น และยังมีกรณีอื่นๆ อีกมากมายที่เราอาจต้องพิมพ์คำสั่งผ่าน Terminal ดังนั้น จึงควรมีพื้นฐานการเขียนคำสั่งผ่าน Terminal เอาไว้บ้าง ดังแนวทางต่อไปนี้

ปกติแล้ว Terminal จะยังไม่ปรากฏ ยกเว้นเราจะเคยเปิดเอาไว้เมื่อใช้งาน VS Code คราวที่แล้ว ทั้งนี้ หากเราต้องการเปิดเข้าสู่ Terminal ให้เลือกที่เมนู **Terminal > New Terminal**



หลังจากนั้น เมื่อ Terminal ปรากฏขึ้น ซึ่งตามปกติจะแบ่งเป็นหลายแท็บ หากเราต้องการเขียนคำสั่งก็ให้เลือกแท็บ Terminal

VS Code นั้นมี Terminal ให้เลือกหลายแบบ เช่น Command Prompt (cmd) หรือ PowerShell เป็นต้น ซึ่งเราจะใช้งานแบบไหนก็ได้ โดยสามารถเลือกรูปแบบที่ต้องการได้ดังนี้



1. เปิด Terminal ขึ้นมาแสดงก่อน
2. คลิกที่ไอคอนลูกศรดังภาพถัดไป แล้วเลือกรูปแบบ Terminal ตามต้องการ
3. หลังจากนั้น Terminal แบบใหม่ที่เราเลือกจะถูกเพิ่มเข้ามา ส่วน Terminal อันเดิม ถ้ายังปรากฏอยู่ก็สามารถลบทิ้งไปโดยการคลิกที่ไอคอนถังขยะ

หลังจากที่เราใช้งาน Terminal เสร็จแล้ว หากไม่ต้องการใช้งานอีก อาจลบวินโดว์ Terminal ทิ้งไป (สร้างใหม่ได้) โดยคลิกที่ไอคอนรูปถังขยะบนแถบทูลบาร์ของมัน และหากต้องการใช้งานอีกก็ต้องสร้างขึ้นมาใหม่ โดยเลือกที่เมนู **Terminal > New Terminal** เช่นเคย หรือถ้าต้องการซ่อน Terminal ชั่วคราว เช่น กรณีที่ยังไม่จำเป็นต้องใช้งานขณะนั้น แต่อาจจะกลับมาใช้งานในภายหลัง (ต้องไม่ปิด VS Code) ก็ให้คลิกไอคอน X ที่มุมขวาบนของแถบทูลบาร์ของมัน และหากจะกลับมาใช้งานอีก ให้เลือกที่เมนู **View > Terminal**

การใช้ Terminal เราต้องพิมพ์คำสั่งสำหรับดำเนินการในกรณีนั้นๆ ลงไป อย่างไรก็ตาม คำสั่งของ Terminal นั้นมีเป็นจำนวนมาก แต่มีเพียงไม่กี่อันที่เราจะได้ใช้งานอยู่บ่อยๆ ดังนั้น จึงขอนำมากล่าวเฉพาะคำสั่งที่น่าสนใจ ซึ่งวิธีการใช้งานคือ ให้เปิดเข้าสู่ Terminal แล้วพิมพ์คำสั่งที่ต้องการลงไป โดยแนวทางการใช้คำสั่งที่สำคัญมีดังนี้

- คำสั่ง **cd** (หรือ **chdir**) สำหรับเปลี่ยน (change) ไปยังไดเร็กทอรีที่ระบุ โดยกำหนดไดเร็กทอรีเป้าหมายตามหลังคำสั่งนี้ เช่น

C:\Users\MY>cd d:\ebooks\javascript	ไปยังไดเร็กทอรี d:\ebooks\javascript
C:\Users\MY>cd documents\node	ไปยังไดเร็กทอรีที่อยู่ถัดจากตำแหน่งที่แสดง prompt คำสั่งในขณะนั้น ซึ่งกรณีนี้คือ C:\Users\MY ดังนั้น คำสั่งดังกล่าวจะไปที่ C:\Users\MY\Documents\node
C:\Users\MY>cd ..	(2 จุด) ไปยังไดเร็กทอรีก่อนนี้ 1 ระดับ นั่นคือ C:\Users
C:\Users\MY>cd \	ไปยังไดเร็กทอรีเริ่มต้นที่แสดง prompt คำสั่งในขณะนั้น ซึ่งกรณีนี้จะไปที่ C:\

- คำสั่ง **md** (หรือ **mkdir**) สำหรับสร้าง (make) ไดเร็กทอรีใหม่ โดยระบุชื่อไดเร็กทอรีที่จะสร้างใหม่ตามหลังคำสั่งนี้ เช่น

C:\Users\MY>mkdir js	สร้างไดเร็กทอรี (โฟลเดอร์) ที่ชื่อ js ซ้อนไว้ในตำแหน่งที่แสดง prompt คำสั่งในขณะนั้น ซึ่งกรณีนี้จะสร้างไว้ที่ C:\Users\MY\js
C:\Users\MY>mkdir js\node	ถ้ายังไม่มีไดเร็กทอรี js ก็จะสร้างขึ้นใหม่ แล้วสร้างไดเร็กทอรี node ซ้อนไว้ใน js
C:\Users\MY>md d:\javascript\code	สร้างไดเร็กทอรี javascript\code ไว้ที่ d:\

- คำสั่ง **rm** (หรือ **rmdir**) สำหรับลบ (remove) ไดเร็กทอรี (ลบได้เฉพาะไดเร็กทอรีว่าง) โดยระบุชื่อไดเร็กทอรีที่ต้องการลบตามหลังคำสั่งนี้ แต่หากไดเร็กทอรีนั้นไม่ว่าง เช่น มีไดเร็กทอรีย่อย ๆ และ/หรือ ไฟล์อยู่ข้างใน จะลบไม่ได้

C:\Users\MY>rmdir test	ลบไดเร็กทอรี (โฟลเดอร์) ที่ชื่อ test ซึ่งซ้อนอยู่ในตำแหน่งที่แสดง prompt คำสั่งในขณะนั้น สำหรับกรณีนี้คือ C:\Users\MY\test
C:\Users\MY>rmdir d:\bad_dir	ลบไดเร็กทอรี (โฟลเดอร์) ในตำแหน่งที่ระบุ (ถ้าว่าง)

- คำสั่ง **del** (หรือ **erase**) สำหรับการลบไฟล์ โดยระบุตำแหน่งไฟล์ดังกล่าวตามหลังคำสั่งนี้ แต่หากกำหนดเป็นชื่อไดเร็กทอรี จะหมายถึงการลบไฟล์ทั้งหมดในไดเร็กทอรีนั้น

C:\Users\MY>del temp.docx	ลบไฟล์ temp.docx ออกจาก c:\Users\MY
C:\Users\MY>del d:\test\badfile.txt	ลบไฟล์ badfile.txt ออกจาก d:\test
C:\Users\MY>erase test1\test2	ลบไฟล์ทั้งหมดออกจากไดเร็กทอรี c:\Users\MY\test1\test2 (แต่ไม่ลบไดเร็กทอรี) และจะมี prompt ให้เรายืนยันด้วย

- คำสั่ง **ren** หรือ **rename** สำหรับเปลี่ยนชื่อไฟล์หรือไดเร็กทอรี โดยระบุชื่อไฟล์หรือไดเร็กทอรีเดิมแล้วตามด้วยชื่อใหม่ เช่น

C:\Users\MY>ren test.js demo.js	เปลี่ยนชื่อไฟล์ test.js ซึ่งอยู่ในตำแหน่งที่แสดง prompt คำสั่งในขณะนั้น สำหรับกรณีนี้คือ C:\Users\MY\test.js ไปเป็นชื่อ demo.js
C:\Users\MY>rename temp backup	เปลี่ยนชื่อไดเร็กทอรี ซึ่งซ้อนอยู่ในตำแหน่งที่แสดง prompt คำสั่งในขณะนั้น สำหรับกรณีนี้คือ C:\Users\MY\temp ไปเป็นชื่อ backup

- คำสั่ง **cls** สำหรับเคลียร์หน้าจอ (screen) โดยไม่จำเป็นต้องระบุพารามิเตอร์ใดๆ เช่น

C:\Users\MY>cls	เคลียร์ข้อความทั้งหมดบนหน้าจอของ Terminal

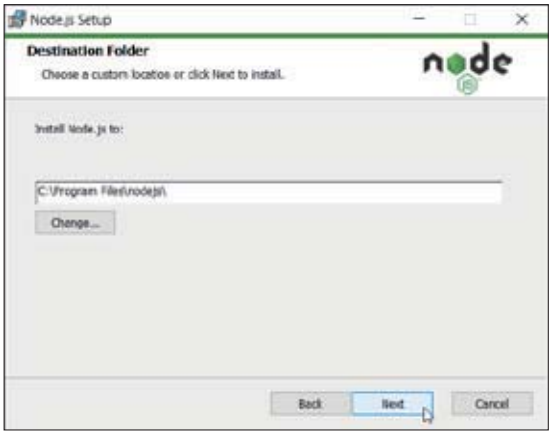
- คำสั่ง **dir** สำหรับแสดงรายชื่อไฟล์และไดเร็กทอรีย่อยๆ ที่อยู่ในไดเร็กทอรีของ prompt คำสั่งในขณะนั้น หรือที่ระบุตามหลังคำสั่งนี้ เช่น

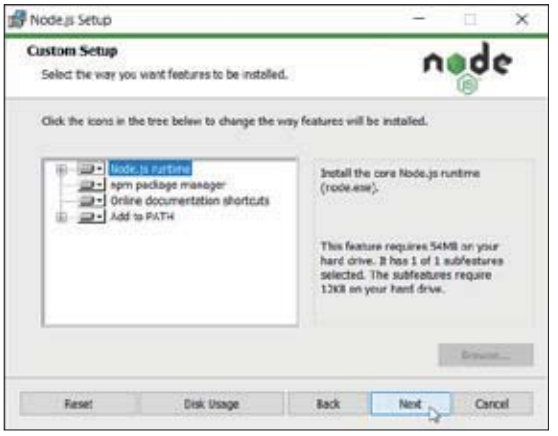
C:\Users\MY>dir c:\	แสดงรายชื่อไฟล์และไดเร็กทอรีใน C:\Users\MY>
C:\Users\MY>dir c:\	แสดงรายชื่อไฟล์และไดเร็กทอรีใน c:\
C:\Users\MY>dir documents\js	แสดงรายชื่อไฟล์และไดเร็กทอรีใน c:\Users\MY\documents\js
C:\Users\MY>dir d:\temp\download	แสดงรายชื่อไฟล์และไดเร็กทอรีใน d:\temp\download

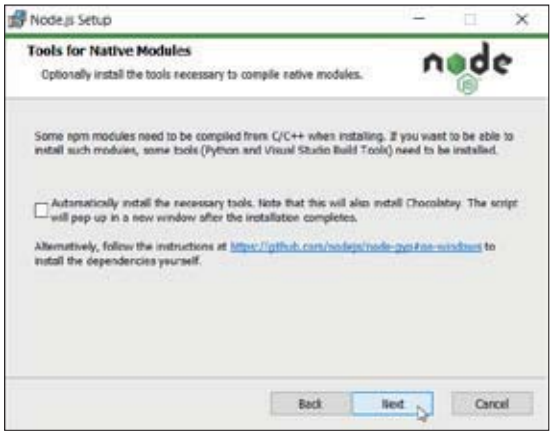
การติดตั้ง Node.js และคำสั่ง NPM

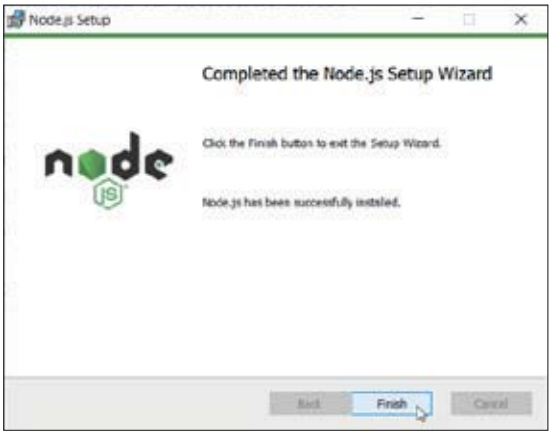
ถึงแม้ React จะเป็นการสร้าง Web Application สำหรับฝั่ง Frontend เป็นหลัก แต่โครงสร้างของแอปจะอยู่ในรูปแบบของ Node.js Application และจำเป็นต้องอาศัย Engine ของ Node.js ในการทำงาน ดังนั้น องค์ประกอบสำคัญอีกอย่างหนึ่งที่เราต้องจัดเตรียมไว้ล่วงหน้าก็คือ Node.js ซึ่งมีขั้นตอนการติดตั้งดังนี้

1. สามารถดาวน์โหลดชุดเครื่องมือของ Node.js ได้ฟรีที่เว็บไซต์ **https://nodejs.org/**
2. หลังจากดาวน์โหลด ก็ดับเบิลคลิกไฟล์ติดตั้ง ซึ่งขั้นตอนต่างๆ ก็เหมือนโปรแกรมทั่วไป









หลังการติดตั้ง เมื่อเปิดเข้าสู่ VS Code ก็สามารถใช้งาน Node.js ผ่าน Terminal ทันที ซึ่งโดยทั่วไปจะเป็นการพิมพ์ชุดคำสั่งที่เรียกว่า NPM (Node Package Manager) สำหรับคำสั่งที่เราควรรู้จักในเบื้องต้นมีดังนี้

- **npm install <ชื่อแพ็กเกจ>** สำหรับการติดตั้งแพ็กเกจตามชื่อที่ระบุ เช่น npm install react-wizard
- **npm install -g <ชื่อแพ็กเกจ>** เมื่อเราต้องการติดตั้งแพ็กเกจแบบ global ให้สามารถใช้งานระหว่างแอปได้ โดยไม่จำเป็นต้องติดตั้งแบบซ้ำๆ ในแต่ละแอปที่จะใช้งาน เช่น npm install -g hello-world
- **npm install <-g> <แพ็กเกจ1> <แพ็กเกจ2> <แพ็กเกจ3> ...** เมื่อต้องการติดตั้งหลายๆ แพ็กเกจพร้อมกัน (ไม่ต้องเสียเวลาติดตั้งทีละแพ็กเกจ) เช่น npm install reactive reaction reactor
- **npm uninstall <ชื่อแพ็กเกจ>** เมื่อต้องการลบหรือยกเลิกการติดตั้งแพ็กเกจตามชื่อระบุออกไปจากแอป เช่น npm uninstall chain-reaction

นอกจากนี้ ยังมีอีกหลายคำสั่งที่เราอาจได้ใช้งานในบางกรณี รวมถึงวิธีการนำคำสั่งที่กล่าวมานี้ไปใช้งานจริง เราจะได้เรียนรู้เพิ่มเติมอีกครั้งในหัวข้อต่อๆ ไป

การใช้โมดูลของ JavaScript

โมดูล (Module) ก็คือไฟล์ที่จัดเก็บโค้ดของจาวาสคริปต์ ซึ่งอาจประกอบด้วย ตัวแปร ฟังก์ชัน คลาส ออบเจ็กต์ หรืออื่นๆ เพื่อนำไปใช้งานยังส่วนอื่นๆ และเนื่องจากการพัฒนาเว็บด้วย React นั้น จะเกี่ยวข้องกับการใช้โมดูลไปตลอด ดังนั้น เราควรรู้จักพื้นฐานในการนำเข้า (import) และส่งออก (export) โมดูลเอาไว้ล่วงหน้า เพื่อให้ง่ายต่อการเรียนรู้ในหัวข้อต่อๆ ไป

การสร้างและส่งออกโมดูล

เนื่องจากโมดูลก็คือไฟล์จาวาสคริปต์ (.js) ดังนั้น หากจะสร้างโมดูล เราก็เริ่มจากการสร้างไฟล์พร้อมกำหนดชื่อให้กับมัน (ชื่อไฟล์จะเป็นชื่อโมดูล) และให้มีส่วนขยายเป็น .js เช่น ในการทดสอบนี้ ผู้เขียนสร้างไฟล์ชื่อ math.js แสดงว่าโมดูลนี้ก็จะชื่อ math เป็นต้น และสมาชิกที่อยู่ในโมดูลอาจเป็น ตัวแปร ฟังก์ชัน คลาส ออบเจ็กต์ ก็ได้ สุดท้ายก็ส่งออกไปด้วยคำสั่ง export เช่น

```
const PI = 3.141

function add(num1, num2) {
     return num1 + num2
}

class Random {
     static getInteger(min, max) {
          return min + Math.floor(Math.random() * (max - min))
     }
     static getBoolean() {
          return (this.getInteger(1, 10) <= 5) ? true : false
     }
}

export {PI, add, Random}//ระบุสิ่งที่จะส่งออกไว้ในวงเล็บ { }

//หรือเราอาจส่งออกในชื่อใหม่ โดยใช้คีย์เวิร์ด as เช่น
//export {PI, Random, add as plus}
//หมายถึง PI, Random ใช้ชื่อเดิมแต่ add เปลี่ยนเป็น plus
```

หรืออีกวิธีคือ นำคีย์เวิร์ด export มาวางไว้ด้านหน้าสมาชิกของโมดูลที่จะส่งออกไป ซึ่งเรียกว่า inline export เช่น

```
export const PI = 3.141

export function add(num1, num2) {
     ...
}

export class Random {
     ...
}
```

การนำเข้าโมดูล

การนำเข้าโมดูล จะใช้คำสั่ง import-form ดังรูปแบบต่อไปนี้

```
import {identifier} from <location>
```

- identifier ให้ระบุชื่อของสมาชิกที่ต้องการนำเข้า เช่น ชื่อตัวแปร ฟังก์ชัน คลาส หรือออบเจ็กต์ เป็นต้น โดยต้องเขียนไว้ในวงเล็บ { } เช่น {add, number, Random}
- location คือตำแหน่งของโมดูล โดยเทียบกับตำแหน่งไฟล์ที่นำโมดูลนั้นมาใช้งาน ซึ่งการระบุตำแหน่งมีหลักการดังนี้
  - หากไฟล์ของโมดูล และไฟล์ที่ต้องการใช้โมดูล อยู่ในไดเร็กทอรีเดียวกัน ให้แทนตำแหน่งด้วย './ชื่อไฟล์' เช่น './math' โดย . หมายถึงไดเร็กทอรีปัจจุบัน
  - แต่ถ้าไฟล์ของโมดูล อยู่ในไดเร็กทอรีย่อยๆ ลงไปอีก เราก็ต้องระบุตำแหน่งไปตามลำดับจนถึงไฟล์ของโมดูล เช่น import { ... } from './modules/js/math'

  เช่น จากโมดูล math ที่เราได้สร้างไว้ในหัวข้อที่แล้ว หากเราต้องการนำเข้าฟังก์ชัน add, คลาส Random จากโมดูล math มาใช้งานในไฟล์ test-module.js ซึ่งเก็บอยู่ในโฟลเดอร์เดียวกับไฟล์โมดูล ก็อาจกำหนดโค้ดดังนี้

```
import {add, Random} from './math'

alert(add(10, 20))
let r = Random.getInteger(1, 100)
```

- เราอาจตั้งชื่อใหม่ให้กับสมาชิกบางอันของโมดูลด้วย as แล้วเรียกใช้งานจากชื่อที่ตั้งขึ้นใหม่ เช่น

```
import {add as plus, Random} from './math'

let x = plus(1, 99)
```

- หากเราต้องการนำเข้าทุกอย่างที่มีในโมดูล สามารถแทนด้วยเครื่องหมาย * แล้วตั้งชื่อให้กับมันด้วย as ซึ่งกรณีนี้ไม่ต้องเขียนในวงเล็บ { } จากนั้นเราอ้างถึงสมาชิกในโมดูลผ่านชื่อที่ตั้งขึ้น

```
import * as math from './math'

let x = math.add(10, 90)
let r = math.Random.getInteger(1, 100)
```

การส่งออกแบบ default

ภายในโมดูล หากมีสมาชิกเพียงอันเดียว เช่น คลาสเดียว หรือออบเจ็กต์เดียว หรือฟังก์ชันเดียว เราสามารถส่งออกสมาชิกแบบดีฟอลต์ โดยมีหลักการดังนี้

- หากใช้วิธี inline ก็ระบุคำว่า export default ที่สมาชิกตัวนั้น เช่น

```
export default class Test {
     static hello() { }
}
```

- หรืออาจกำหนดสมาชิกขึ้นมาก่อน แล้วใช้คำสั่ง export ทีหลัง เช่น

```
class Test() {
     static hello() { }
}

export default Test                 //แบบที่ 1
//export {Test as default}          //แบบที่ 2
```

- ขั้นตอนการนำเข้า ไม่ต้องมีวงเล็บ { } ครอบชื่อสมาชิกที่ส่งออกแบบ default ส่วนหลักการอื่นๆ ก็เหมือนเดิม เช่น

```
import Test from './test'

Test.hello()
```

- ถ้าต้องการนำเข้าโมดูลที่ส่งออกแบบ default แล้วเปลี่ยนชื่อ ก็ใช้ as ตามเดิม แต่กรณีนี้ต้องเขียนในวงเล็บ { } เช่น

```
import {default as demo} from './test'

demo.hello()
```

อย่างไรก็ตาม ไม่จำเป็นต้องใช้ default กับโมดูลที่มีสมาชิกเพียงอันเดียว เราสามารถใช้กับโมดูลที่มีสมาชิกมากกว่า 1 อันได้ **แต่ในโมดูลเดียวกันจะมีสมาชิกแบบ default ได้เพียงอันเดียวเท่านั้น** เช่น

```
//แบบที่ 1
export function hi() { return 'Hi'  }
export default function hello() { return 'Hello' } //ส่งออกแบบ default
export function hey() { return 'Hey'}
```

```
//แบบที่ 2
function hi() { ... }
function hello() { ... }
function hey() { ... }

export {hi, hey, hello as default}                    //ส่งออก hello แบบ default
```

แล้วในขั้นตอนการนำเข้า ก็กำหนดโค้ดแบบใดแบบหนึ่งในลักษณะดังนี้

```
//เอาตัว default ขึ้นก่อน
import hello, {hi, hey} from './example'

//หรือแบบอื่นๆ เช่น
//import {default as hello, hi, hey} from './example'
//import hello, * as H from './example'
```

ทั้งหมดที่กล่าวคือ หลักการพื้นฐานทั่วๆ ไปตามข้อกำหนดของ JavaScript (ECMAScript) ซึ่งเมื่อนำไปใช้งานใน React Application อาจมีบางลักษณะที่แตกต่างไปจากนี้เล็กน้อย โดยส่วนใหญ่ก็ยังเป็นไปตามหลักการที่กล่าวมานี้

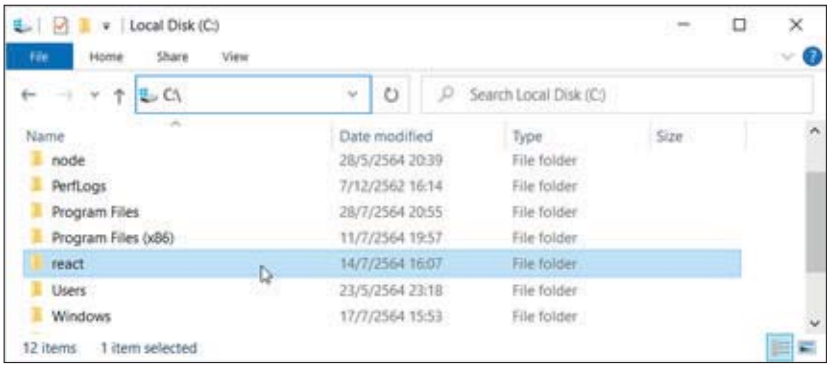
การสร้างและทดสอบ React Application

การสร้างเว็บด้วย React ต้องทำในแบบของ Web Application ที่ประกอบด้วยไฟล์หลายชนิดร่วมกัน เช่น HTML, CSS, JavaScript รวมถึงโมดูลของ React ซึ่งอยู่ในรูปแบบของ Node.js อย่างไรก็ตาม React ได้จัดเตรียมคำสั่ง npm ให้เราใช้ในการสร้างแอปเอาไว้แล้ว ซึ่งเราแค่พิมพ์คำสั่งลงใน Terminal แล้วองค์ประกอบ (ไฟล์) พื้นฐานทั้งหมดของ React App ก็จะถูกสร้างขึ้นหรือดาวน์โหลดมาติดตั้งให้โดยอัตโนมัติ ทั้งนี้หากผู้อ่านเคยสร้างแอปของ Node.js มาก่อน ในกรณีของ React ก็ใช้หลักการเดียวกัน โดยมีขั้นตอนปลีกย่อยดังต่อไปนี้

สร้างโฟลเดอร์สำหรับจัดเก็บแอปต่างๆ

หากเราต้องการสร้างหลายๆ แอปเพื่อทดสอบการทำงานในกรณีต่างๆ ก็ควรสร้างโฟลเดอร์เพื่อเก็บแอปเหล่านั้นให้เป็นสัดส่วนเฉพาะ สำหรับชื่อและตำแหน่งการจัดเก็บ ผู้อ่านสามารถกำหนดได้เองตามต้องการ แต่ควรจะเข้าถึงได้ง่าย เช่น กรณีที่จะนำเสนอประกอบการใช้งานในหนังสือเล่มนี้ ผู้เขียนจะสร้างโฟลเดอร์ชื่อ

react ไว้ที่ตำแหน่ง **c:\react** สำหรับเก็บแอปของ React ดังภาพถัดไป ซึ่งผู้อ่านอาจสร้างโฟลเดอร์ผ่าน File Explorer หรือใช้คำสั่ง mkdir บน Terminal ก็ได้ เช่น mkdir c:\react



การติดตั้งคำสั่ง create-react-app

เนื่องจากการสร้าง React App เราต้องพิมพ์คำสั่งในแบบของ npm แต่คำสั่งนี้ไม่ใช่องค์ประกอบพื้นฐานของ Node.js ดังนั้น เราจะต้องติดตั้งเพิ่มลงไปก่อน โดยใช้รูปแบบดังนี้

```
npm install -g create-react-app
```

```
C:\react>npm install -g create-react-app
```

กรณีนี้ เราควรระบุออปชั่น -g ลงไปด้วย เพื่อให้คำสั่งหรือแพ็กเกจนี้ถูกติดตั้งแบบ Global แล้วในการสร้างแอปครั้งต่อๆ ไป ก็ไม่จำเป็นต้องติดตั้งคำสั่งนี้ซ้ำอีก แต่สามารถใช้งานได้ตลอด

การสร้าง React App

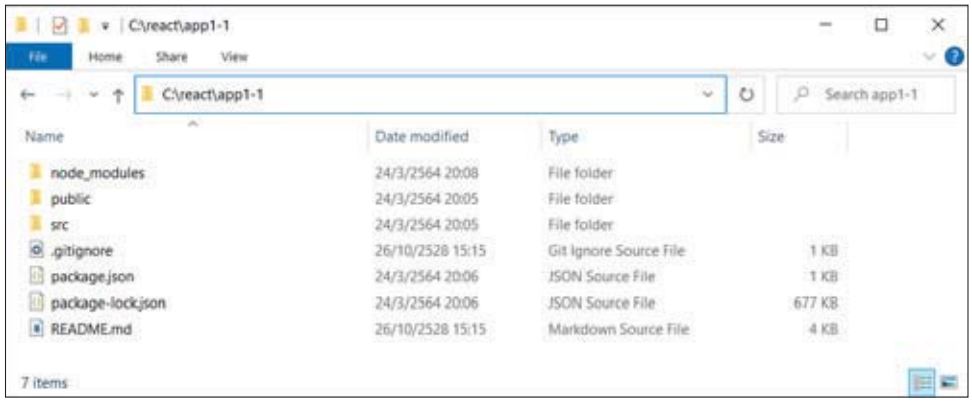
การสร้าง React App จะต้องใช้คำสั่ง create-react-app ตามที่เราได้ติดตั้งเอาไว้ในหัวข้อที่แล้ว ด้วยรูปแบบดังนี้

```
create-react-app <ชื่อแอป>
```

โดยเราต้องพิมพ์คำสั่งนี้ลงบน Terminal แต่เนื่องจากแอปจะถูกสร้างในโฟลเดอร์หรือตำแหน่งที่แสดง Prompt อยู่ในขณะนั้น ซึ่งหากไม่ใช่เป้าหมายที่เราต้องการ ก็ให้ใช้คำสั่ง cd เพื่อเปลี่ยนตำแหน่งไดเร็กทอรีก่อนแล้วค่อยสร้างแอป โดยตามข้อกำหนดในหนังสือเล่มนี้คือ จะสร้างแอปไว้ C:\react ดังนั้น แนวทางโดยรวมก็จะเป็นดังภาพ ซึ่งสมมติว่าต้องการสร้างแอปชื่อ app1-1

```
C:\Users\MY>cd c:\react

c:\react>create-react-app app1-1
```

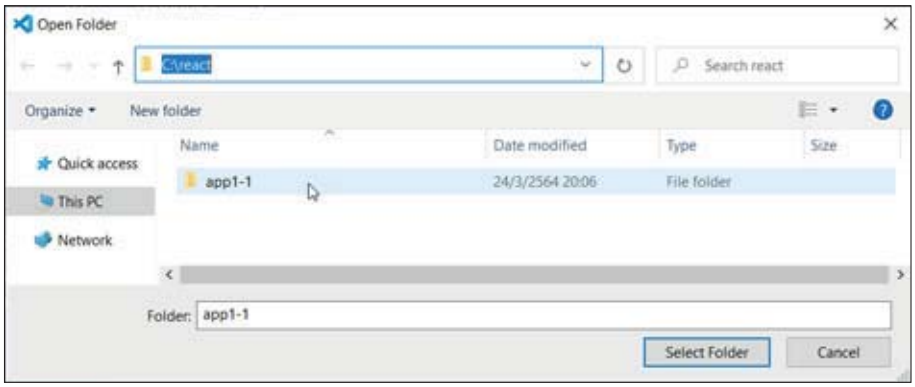
การสร้าง React App ดังที่กล่าวมา เราต้องเชื่อมอินเทอร์เน็ตและใช้เวลาพอสมควร กว่าไฟล์ทั้งหมดจะถูกดาวน์โหลดมาติดตั้งจนครบสมบูรณ์ โดยในขณะที่เขียนหนังสือเล่มนี้ React App จะมีขนาดเริ่มต้นประมาณ 157 MB ต่อ 1 แอป และหลังจากการสร้างแอปเสร็จสิ้น หากเราเปิด File Explorer เข้าไปดูที่ตำแหน่งการจัดเก็บแอป ก็จะปรากฏโฟลเดอร์และไฟล์ดังนี้



การเปิดเข้าสู่แอป

หลังจากที่เราสร้างแอปดังหัวข้อที่ผ่านมา หากต้องการเปิดเข้าสู่แอป ก็สามารถทำได้หลายวิธี ดังต่อไปนี้ โดยสมมติว่า เราต้องการเปิดแอปที่ชื่อ app1-1

- **วิธีที่ 1** เปิดจากเมนูของ VS Code โดยเลือกที่ **File > Open Folder** จากนั้นก็เลือกโฟลเดอร์ของแอปที่เราสร้างเอาไว้



- **วิธีที่ 2** โดยการพิมพ์คำสั่งผ่าน Terminal ของ VS Code ดังขั้นตอนต่อไปนี้

  1) ในขณะนั้น Prompt ต้องชี้อยู่ที่โฟลเดอร์ของแอป ถ้าอยู่ที่ตำแหน่งอื่น ให้ใช้คำสั่ง cd เพื่อเข้าไปยังโฟลเดอร์ของแอป เช่น cd c:\react\app1-1

  2) พิมพ์คำสั่งเพื่อเปิดโฟลเดอร์ดังนี้

     code . -r

     โดยที่ . (จุด) หมายถึงเปิดโฟลเดอร์ปัจจุบันที่แสดง Prompt อยู่ในขณะนั้นและ -r หมายถึงเปิดแทนที่แอปเดิมที่กำลังเปิดอยู่ (ถ้ามี) หรือเปิดในวินโดว์ปัจจุบันของ VS Code นั่นเอง

```
c:\react>cd app1-1

c:\react\app1-1>code . -r
```

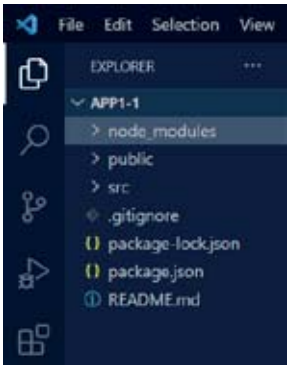
  นอกจากนี้ ยังมีอีกวิธีคือ การระบุตำแหน่งโฟลเดอร์ที่ต้องการเปิดลงไปโดยตรง เช่น code c:\react\app1-1 -r เป็นต้น

- **วิธีที่ 3** ถ้าเราเคยเปิดแอปนั้นมาก่อน เมื่อเข้าสู่ VS Code จะปรากฏรายชื่อบนหน้าจอ Welcome ในกลุ่ม Recent ซึ่งหากมีรายชื่อแอปที่เราต้องการ ก็สามารถเปิดผ่านตรงนี้ได้เลย (หากหน้าจอ Welcome ไม่ปรากฏ ให้เลือกที่เมนู Help > Welcome)

หลังจากเปิดแอปขึ้นมาแล้ว หากต้องการปิด ให้เลือกที่เมนู File > Close Folder หรือ File > Exit (วิธีนี้เมื่อเปิด VS Code ครั้งต่อไป จะเข้าสู่แอปที่เคยเปิดครั้งล่าสุดโดยอัตโนมัติ)

ลักษณะโครงสร้างของ React Application

หลังจากที่เราได้สร้าง React App ขึ้นมาแล้ว เมื่อเปิดเข้าสู่แอปดังวิธีการที่ได้มาแล้ว จะปรากฏโครงสร้างหรือองค์ประกอบภายในแอปดังภาพ สำหรับโฟลเดอร์และไฟล์ที่สำคัญภายในแอป ซึ่งเราควรรู้จักในเบื้องต้นมีดังนี้



- **node_modules** เป็นโฟลเดอร์สำหรับจัดเก็บโมดูลที่จำเป็นสำหรับการทำงานของ React โดยโมดูลเหล่านี้จะอยู่ในรูปแบบของ Node.js ทั้งนี้ หากเราติดตั้งโมดูลอื่นๆ เพิ่มเติมในภายหลังด้วยคำสั่ง npm ก็จะถูกนำมาเก็บไว้ในโฟลเดอร์นี้
- **public** เป็นโฟลเดอร์สำหรับจัดเก็บไฟล์ที่ต้องใช้งานร่วมกันระหว่างเพจหรือไฟล์อื่นๆ ซึ่งโดยส่วนใหญ่จะเป็นไฟล์แบบ Static หรือไฟล์ที่ไม่มีการเปลี่ยนเนื้อหานั่นเอง เช่น ไฟล์ HTML, รูปภาพ เป็นต้น
- **src** เป็นโฟลเดอร์สำหรับจัดเก็บไฟล์ Source Code หลักของแอป เช่น ซึ่งโดยส่วนใหญ่จะเป็น JavaScript แต่บางกรณีก็อาจเป็น CSS โดยไฟล์สำคัญภายในโฟลเดอร์นี้คือ

- **src\App.css** เป็นที่จัดเก็บโค้ด CSS ส่วนกลางสำหรับนำไปวางที่ส่วนอื่นๆ ในแบบ Placeholder CSS
- **src\App.js** เป็นไฟล์ของคอมโพเนนต์ระดับ Top-Level ของ React
- **src\index.js** คือไฟล์หลักสำหรับหน้าแรกของแอปพลิเคชัน หรือเทียบได้กับ Homepage ของเว็บไซต์นั่นเอง
- **src\index.css** เป็นไฟล์ CSS สำหรับหน้าแรก หรือใช้คู่กับไฟล์ index นั่นเอง

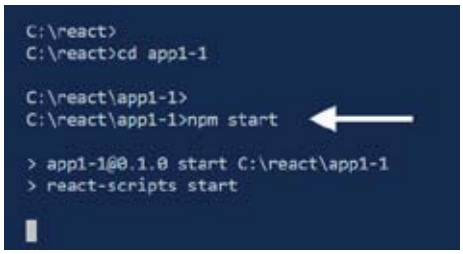
- **package.json** เป็นไฟล์ที่เก็บการตั้งค่าต่างๆ เกี่ยวกับแอปพลิเคชันนั้น
- **package-lock.json** ใช้จัดเก็บข้อมูลของโมดูลที่ติดตั้งในโฟลเดอร์ node_modules เพื่อใช้เป็นค่าอ้างอิงสำหรับคำสั่ง npm โดยทั่วไป เราไม่จำเป็นต้องจัดการใดๆ กับไฟล์นี้
- **.gitignore** รายชื่อไฟล์ที่ไม่ควรเพิ่มลงใน Git แต่ให้ใช้ภายในแอปนี้เท่านั้น
- **README.md** ส่วนขยาย .md คือไฟล์ประเภท Markdown และสำหรับไฟล์ README.md เป็นการแสดงคำสั่งและข้อมูลที่จำเป็นเมื่อเรานำไปแอปไปใช้บนแพลตฟอร์มอื่น เช่น GitHib

ทั้งหมดนี้เป็นเพียงองค์ประกอบพื้นฐานบางส่วนที่ React สร้างมาให้ นอกจากนี้ เราอาจต้องสร้างไฟล์เพิ่มเติมหรือนำมาจากที่อื่นๆ เพื่อให้แอปสามารถทำงานได้ตามต้องการ

การรันทดสอบแอป

หากเราต้องการรันทดสอบองค์ประกอบพื้นฐานที่ React สร้างมาให้ ก็ต้องเปิดเข้าสู่แอปนั้นก่อน ดังที่กล่าวไปในหัวข้อที่แล้ว จากนั้นก็ทำตามขั้นตอนดังนี้

1. ที่ Terminal ต้องเข้าไปในโฟลเดอร์ของแอปที่ต้องการรัน (ยกเว้น Prompt จะแสดงอยู่ที่ตำแหน่งนั้นอยู่แล้ว) เช่น สมมติว่าเราต้องการรันแอป app1-1 ก็ใช้คำสั่ง cd เพื่อเข้าไปยังโฟลเดอร์ดังกล่าว



2. พิมพ์คำสั่ง **npm start** ลงบน Terminal
3. ต่อจากนั้น ระบบเซิร์ฟเวอร์จำลองของ React จะเริ่มการทำงาน (อาจใช้เวลาเล็กน้อย เพราะแอปมีขนาดค่อนข้างใหญ่) หลังจากนั้น เว็บเบราว์เซอร์ที่ถูกตั้งค่าเป็นดีฟอลต์จะถูกเปิดขึ้นมาโดยอัตโนมัติ พร้อมกับแสดงผลที่ตำแหน่ง http://localhost:3000 (หรือเทียบเท่ากับ http://127.0.0.1:3000) โดย 3000 หมายเลขพอร์ต ดังภาพถัดไป ซึ่งเป็นหน้า Homepage ที่ React สร้างไว้ให้ล่วงหน้า

ระบบเซิร์ฟเวอร์จำลองจะทำงานต่อเนื่องไปเรื่อยๆ ซึ่งในระหว่างนั้น หากเราเปิดเบราว์เซอร์ตัวอื่นขึ้นมาแล้วกำหนด URL เป็น http://localhost:3000 (หรือ http://127.0.0.1:3000) ก็จะได้ผลในแบบเดียวกัน และถ้าเราต้องการหยุดการรัน ให้กลับไปที่ Terminal แล้วคลิกบนพื้นที่ว่างของมัน (เพื่อให้ Terminal อยู่ในสถานะ Active) แล้วกด **<Ctrl +C>** จากนั้นเมื่อมี Prompt ขึ้นมาถาม ก็ให้ตอบ Y (Yes) เพื่อยืนยัน แล้วเซิร์ฟเวอร์จำลองจะหยุดทำงาน ดังนั้น การสื่อสารระหว่างเบราว์เซอร์กับเซิร์ฟเวอร์จะสิ้นสุดลง และถ้าเราต้องการกลับไปรันอีกครั้ง ก็พิมพ์คำสั่ง npm start ตามเดิม แล้วหยุดรันด้วย <Ctrl + C> เป็นเช่นนี้ไปตลอด

หลักการของ React JSX

JSX (JavaScript XML) เป็นเทคนิคการเขียนแท็ก HTML ภายในโค้ดของ JavaScript โดยไม่ใช้วิธีการแบบสตริง ซึ่ง JSX ถือเป็นจุดเด่นอีกอย่างหนึ่งของ React แต่ก่อนอื่น เราลองมาดูวิธีการเขียน HTML ลงในโค้ด JavaScript แบบปกติทั่วไป ที่ต้องกำหนดขอบเขตด้วยเครื่องหมายคำพูด (Quotes หรือ Backticks) อันใดอันหนึ่ง ดังนี้ (ยังไม่ใช่ JSX)

```
const head = "<h2>React & React Native</h2>"
const text = '<div><b>Create Web & Mobile Apps with JavaScript</b></div>'
const table = `
     <table>
          <tr><th>Column 1</th><th>Column 2</th></tr>
          <tr><td>...</td><td>...</td></tr>
          ...
     </table>`
```

แต่การเขียนในแบบ JSX เราไม่ต้องเขียนเครื่องหมายคำพูดกำกับ สามารถเขียนโค้ด HTML ลงไปโดยตรงเหมือนกับในเว็บเพจ และถ้าสตริงนั้นยาวเกิน 1 บรรทัด ให้เขียนไว้ในวงเล็บ () เช่น

```
const head = <h2>React & React Native</h2>
const text = <div><b>Create Web & Mobile Apps with JavaScript</b></div>
const table = (
     <table>
          <tr><th>Product</th><th>Price</th></tr>
          <tr><td>React</td><td>320</td></tr>
          <tr><td>React Native</td><td>340</td></tr>
          ...
     </table>
)
```

จะเห็นได้ว่าการเขียนโค้ด HTML ในแบบ JSX นั้น ทำได้ง่ายกว่าและเป็นหนึ่งของ JavaScript โดยไม่ต้องเขียนเครื่องหมายคำพูดกำกับให้ดูยุ่งเหยิง อย่างไรก็ตาม โค้ดดังกล่าวไม่ตรงตามหลักการของ JavaScript ดังนั้น React จึงมีกลไกที่ช่วยแปลงจาก JSX ไปเป็นสตริงตามรูปแบบที่ถูกต้องของ JavaScript ดังนั้น จึงสามารถนำไปแสดงผลบนเบราว์เซอร์ได้ตามปกติ โดยไม่เกิดปัญหาใด ๆ

และนอกจากนี้ การใช้ JSX ยังมีข้อกำหนดและลักษณะที่เราควรรู้เพิ่มเติมอีกหลายประการ ดังรายละเอียดต่อไปนี้

- ไฟล์ JavaScript ที่เราจะเขียน JSX ได้ ต้องนำเข้า (import) คอมโพเนนต์ที่ชื่อ React จากโมดูล 'react' เป็นอันดับแรก เช่น

```
import React from 'react'
```

- แท็ก HTML ที่เขียนในแบบ JSX ต้องบรรจุอยู่ใน Parent หรือมี Container ที่มีแท็กเปิดและแท็กปิดเพื่อกำหนดขอบเขตแทนการใช้เครื่องหมายคำพูด เช่น

```
const t1 = <div><b>Hello React</b></div>          //มี div เป็น Parent

//หรือสามารถใช้แท็กว่างเป็น Parent ก็ได้ เช่น
cost t2 = <>This text is in empty parent</>
```

```
//แต่ถ้าเขียนแบบนี้จะเกิดข้อผิดพลาด เพราะทั้งหมดไม่อยู่ใน Parent
const t3 = <br/><span>Hi React</span>            //Error เพราะเริ่มต้นด้วย <br/>
const t4 = สร้าง Mobile App ด้วย React Native    //Error เพราะไม่มี Parent
```

- ถ้ามีหลายบรรทัด ให้เขียนไว้ในวงเล็บ () เช่น

```
const table = (
     <table>
          <tr><th>Product</th><th>Price</th></tr>
          <tr><td>React</td><td>1000</td></tr>
          ...
     </table>
)
```

- เนื่องจาก JSX ใช้หลักการของ XML ดังนั้น แท็ก HTML ทั้งหมดที่เขียนต้องมีทั้งแท็กเปิดและปิดคู่กันเสมอ คือเริ่มต้นด้วย Open Tag ในแบบ <tagname> และสิ้นสุดด้วย Close Tag ในแบบ </tagname> แต่กรณีที่เป็นแท็กเดี่ยว (Sigle Tag) ให้เขียนในแบบ <tagname/> เช่น

แท็กปกติของ HTML	แท็กในแบบ JSX
<hr>	
<input ...>	<input ... />
<img ...>	

- เราสามารถเขียน HTML ในแบบ JSX ให้กับตัวแปร ค่าคงที่ ฟังก์ชัน หรือกำหนดเป็นอาร์กิวเมนต์ของฟังก์ชัน และอื่นๆ อีกหลายแบบ เช่น

```
import React from 'react'

let title = <h2>React & React Native</h2>

function showFooter() {
     return <div>&copy;2000-2030 developerthai.com</div>
}
```

```
function test(html) {
     ...
}

test(<h3>Hello</h3>)     //สามารถกำหนด HTML ให้เป็นอาร์กิวเมนต์เมื่อเรียกฟังก์ชัน
```

- เราสามารถแทรกค่าตัวแปรหรือค่าจากฟังก์ชัน หรือ Expression ที่มีผลลัพธ์เกิดขึ้น ลงใน JSX ได้เช่นเดียวกับการแทรกลงใน Template String ของ JavaScript โดยระบุตัวแปรไว้ในวงเล็บ { } แต่กรณีนี้ไม่ต้องเขียนเครื่องหมาย $ ไว้หน้าวงเล็บเหมือนที่ต้องทำใน Template String ของ JavaScript เช่น

```
import React from 'react'

let r = 'React'
let rn = 'React Native'
let title = <span>Developing Web & Mobile Apps with {r} & {rn}</span>
let a = 10
let b = 20
let x = <div>{a} + {b} = {a + b}</div>
```

- หากข้อมูลที่นำมาแทรกเป็นแบบอาร์เรย์ สมาชิกทั้งหมดจะถูกเชื่อมต่อเป็นสตริงเดียวกัน เช่น

```
let a = ['red', 'green', 'blue']
let b = <div>{a}</div>       //b = redgreenblue
```

- หากตัวแปรที่เรานำมาแทรก มีแท็ก HTML รวมอยู่ด้วย (HTML String) เมื่อแสดงผล จะปรากฏเป็นแท็กในแบบสตริงตามเดิม เช่น Hello แต่หากเราต้องการให้แท็กถูกประมวล ต้องกำหนดด้วยแอตทริบิวต์ **dangerouslySetInnerHTML** ดังโค้ดต่อไปนี้

```
let r = '<b>React</b>'
let a = <span>{r}</span>   //a = <b>React</b>
let b = <span dangerouslySetInnerHTML={{__html: r}}/>  //React จะเป็นตัวหนา
```

- นอกจากการแทรกค่าตัวแปรแล้ว เรายังสามารถกำหนดเงื่อนไขแบบ Ternary ภายในวงเล็บ { } ดังแนวทางต่อไปนี้

```
let success = false
const msg = <div>{(success) ? 'Yes' : 'No'}</div>
//หมายความว่า หาก success = true จะแทรกคำว่า 'Yes' มิฉะนั้นจะแทรกคำว่า 'No'
```

```
let num = parseInt(Math.random()*10)
const oddEven = (
  <div>
      {num % 2 == 0) ? 'Even' : 'Odd'}
  </div>
)
//ถ้าหาร 2 ลงตัว จะแทรกคำว่า 'Even' มิฉะนั้น จะแทรกคำว่า 'Odd'
```

- เนื่องจาก HTML ที่เขียนในแบบ JSX ต้องถูกแปลงเป็นสตริงในรูปแบบโค้ดของ JavaScript ดังนั้น การเขียนแอตทริบิวต์ของ HTML ต้องตรงตามหลักการของ JavaScript ซึ่งสามารถสรุปประเด็นที่น่าสนใจได้ดังนี้
  - แอตทริบิวต์ที่ตรงกับ Reserved Word ของ JavaScript ได้แก่ class และ for ให้แทนด้วยค่าใหม่ ดังนี้

แอตทริบิวต์แบบ HTML	แอตทริบิวต์แบบ JSX
class	className
for (ปกติใช้กับ label)	htmlFor

```
<div className="container">...</div>
<label htmlFor="login">Login</label>
<input id="login" ... />
```

  - แอตทริบิวต์ที่เป็นคำเดียว ให้เขียนเป็นตัวพิมพ์เล็กทั้งหมดตามแบบเดิมของ HTML เช่น id, style, name, value, placeholder, src, disabled เป็นต้น
  - แอตทริบิวต์ที่เกิดจากการนำตั้งแต่ 2 คำขึ้นไปมารวมกัน ให้เขียนแบบ camelCase นั่นคือ คำแรกให้ขึ้นต้นด้วยตัวพิมพ์เล็ก แต่คำที่ 2 เป็นต้นไปให้ขึ้นต้นด้วยตัวพิมพ์ใหญ่ เช่น

แอตทริบิวต์แบบ HTML	แอตทริบิวต์แบบ JSX
maxlength	maxLength
tabindex	tabIndex
onclick	onClick
onmousedown	onMouseDown
...	...

```
const input =
     <input type="number" id="n1" className="num" maxLength="5" />
```

- หากต้องการเขียนคำอธิบาย (Comment) แทรกใน JSX ต้องเขียนในแบบ {/* คำอธิบาย */} เท่านั้น จะใช้รูปแบบของ HTML คือ <!-- --> ไม่ได้ เช่น

```
const table = (
     <table>
          <tr><th>Product</th><th>Price</th></tr>
          {/*
          <tr><td>React</td><td>1000</td></tr>
          */}
     </table>
)
```

หลักการเขียน HTML ในแบบ JSX ดังที่กล่าวมานี้ เป็นพื้นฐานเพียงส่วนหนึ่งเท่านั้น สำหรับลักษณะปลีกย่อยอื่นๆ รวมถึงแนวทางการนำไปใช้งาน ขอให้ดูร่วมกับหัวข้อต่อๆ ไป

ReactDOM และการแสดงเว็บเพจ

ในมุมมองของ JavaScript นั้น DOM (Document Object Model) คือออบเจ็กต์สำหรับการเข้าถึงและจัดการอิลิเมนต์ต่างๆ ภายในเพจ เช่น อ่านค่า เปลี่ยนแปลง เพิ่ม และลบอิลิเมนต์ในเอกสาร HTML เป็นต้น ส่วนใน React ก็มี DOM เช่นเดียวกัน โดยอยู่ในรูปแบบของคอมโพเนนต์ที่ชื่อ ReactDOM ซึ่งหน้าที่หลักๆ ที่เราจะได้ใช้งานในเบื้องต้นก็คือ การแสดงผลเว็บเพจ โดยมีแนวทางดังต่อไปนี้

- การใช้งาน ReactDOM อาจทำได้ใน 2 รูปแบบคือ
  - **แบบที่ 1** นำเข้าคอมโพเนนต์ **ReactDOM** จากโมดูล **'react-dom'** แล้วแสดงผลโดยเรียกเมธอด render() ผ่านคอมโพเนนต์นี้เช่น

```
import ReactDOM from 'react-dom'

ReactDOM.render(...)     //เมื่อต้องการแสดงผล
```

  - **แบบที่ 2** นำเข้าเมธอด render() แบบเจาะจงจากโมดูล **'react-dom'** แล้วแสดงผลด้วยการเรียกเมธอด render() โดยตรง เช่น (ถ้าไม่เข้าใจ ให้ย้อนกลับไปดูที่หัวข้อการนำเข้าโมดูล)

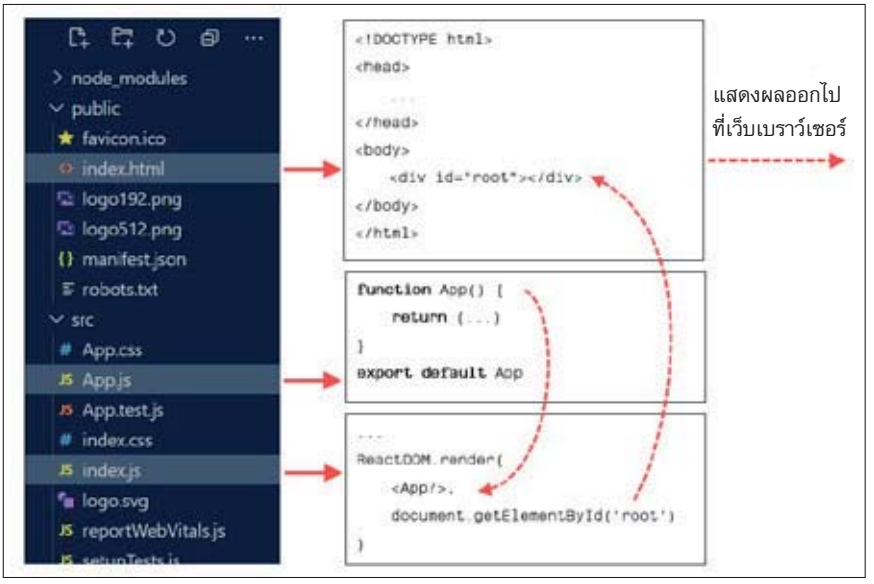
```
import {render} from 'react-dom'

render(...)                    //เมื่อต้องการแสดงผล
```

- เมธอด render() ซึ่งใช้สำหรับการแสดงผลเว็บเพจ มีรูปแบบดังนี้

```
render(element, container [, callback])
```

  - element คือสิ่งที่เราจะแสดงผล ซึ่งอาจกำหนดเป็น HTML ในแบบ JSX โดยตรง หรือจะนำค่ามาจากตัวแปร หรือฟังก์ชัน หรืออื่นๆ อีกหลายแบบ
  - container คือเป้าหมายในการแสดงผล (นำ HTML มาแสดงผลในอิลิเมนต์นี้)
  - callback อาจระบุเมื่อเราต้องการกระทำบางอย่างเมื่อแสดงผล โดยเรียกฟังก์ชันเป้าหมายขึ้นมาทำงาน

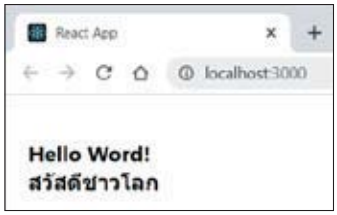
- สำหรับแนวทางการนำไปใช้ก็คือ การแสดงผลแอปที่เราสร้างขึ้น โดยในเบื้องต้นมี 3 ไฟล์หลักๆ ที่เกี่ยวข้องคือ
  - **public/index.html** เป็นตำแหน่ง container ที่จะแสดงผล
  - **src/App.js** กำหนดองค์ประกอบส่วนต่างๆ ของแอป
  - **src/index.js** แสดงองค์ประกอบที่กำหนดไว้ใน App.js ออกไปที่ index.html



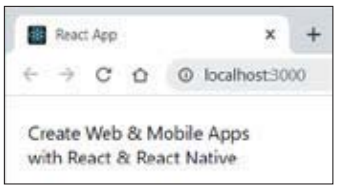
สำหรับในเบื้องต้น เราจะเขียนโค้ดที่ไฟล์ App.js เป็นหลัก โดยหากต้องการทดสอบ JSX ก็อาจกำหนดโค้ดไว้ในฟังก์ชัน App() โดยลบโค้ดเดิมที่ React สร้างมาให้ในฟังก์ชันทิ้งไป โดยหลังการแก้ไขโค้ด ให้บันทึกการเปลี่ยนแปลงก่อนรันทดสอบ (Ctrl + K + S) แล้วทำการ Refresh เบราว์เซอร์เพื่อดูผลลัพธ์ล่าสุด ซึ่งแนวทางการทดสอบบางส่วนเป็นดังนี้ (นำมาแสดงเฉพาะโค้ดที่เกี่ยวข้องในไฟล์ App.js)

react/app1-1/src/App.js

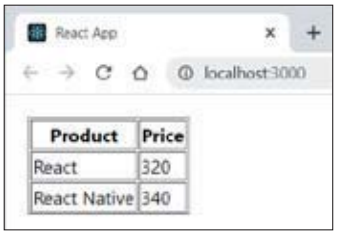
```
function App() {
      return <h3>Hello Word! <br/>สวัสดีชาวโลก</h3>
}
```




```
function App() {
      let r = 'React'
      let rn = 'React Native'
      const el = (
            <div>
                  Create Web & Mobile Apps<br/>
                  with {r} & {rn}
            </div>
      )
      return el
}
```




```
function App() {
      return (
            <table border="1">
                  <tr><th>Product</th><th>Price</th></tr>
                  <tr><td>React</td><td>320</td></tr>
                  <tr><td>React Native</td><td>340</td></tr>
            </table>
      )
}
```



การกำหนดสไตล์ CSS ร่วมกับ JSX

กรณีที่เราต้องการปรับแต่งเว็บเพจด้วย CSS หากเขียนโค้ดในแบบ JSX ก็มีหลักการบางส่วนที่แตกต่างไปจากกรณีที่เรากำหนด CSS ให้กับเอกสาร HTML โดยตรง ซึ่งการนำ CSS มาใช้ร่วมกับ JSX จะแบ่งเป็น 2 ลักษณะคือ แบบ Inline Style และ External Style ดังรายละเอียดต่อไปนี้

การกำหนดสไตล์แบบ Inline

การกำหนดสไตล์แบบ Inline ก็คือการเขียนโค้ด CSS ไว้ที่แท็ก HTML โดยตรง ซึ่งในกรณีที่เราเขียนโค้ดในแบบ JSX ก็มีหลักการที่แตกต่างจากแบบปกติดังนี้

- กำหนดสไตล์ในแบบออบเจ็กต์ของ JavaScript โดยชื่อพร็อปเพอร์ตี้ต้องเขียนในแบบ camelCase ห้ามมีขีดคั่นระหว่างคำตามหลักการของ JavaScript และค่าของพร็อปเพอร์ตี้ต้องกำหนดไว้ในเครื่องหมายคำพูด
- นำออบเจ็กต์ของสไตล์มากำหนดให้กับแท็กด้วยแอตทริบิวต์ style ตามปกติ แต่ต้องเขียนค่าของมันไว้ในวงเล็บ { } เช่น

```
const divStyle = {                          //สร้างสไตล์ในแบบออบเจ็กต์
     color: 'red',
     backgroundColor: 'powderblue',         //แบบ CSS คือ background-color
     fontSize: 'larger',                    //แบบ CSS คือ font-size
}

return <div style={divStyle}>Hello React</div>
```

- เราสามารถกำหนดไสตล์ย่อยๆ ให้แก่แท็กเดียวกันในแบบของอาร์เรย์ เช่น

```
const redText = { color: 'red' }
const darkBg = { backgroundColor: 'black' }
...
return <div style={[redText, darkBg]}>Hello React</div>
```

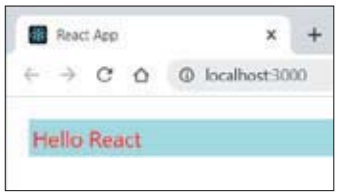
- หรืออีกวิธีคือ กำหนดไสตล์ในแบบออบเจ็กต์ให้กับแท็กโดยตรง ดังโค้ดถัดไป (วงเล็บ { } อันข้างในใช้กำหนดออบเจ็กต์ ส่วนอันข้างนอกใช้สำหรับกำหนดสไตล์ในแบบ JSX)

```
<div style={{color:'red', backgroundColor:'powderblue'}}>
     Hello React
</div>
```

แนวทางการเขียนโค้ดโดยรวมที่ไฟล์ App.js จะมีลักษณะดังนี้

react/app1-1/src/App.js

```
function App() {
      const divStyle = {
            color: 'red',
            backgroundColor: 'powderblue',
            fontSize: 'larger',
            padding: '3px',
      }
      return <div style={divStyle}>Hello React</div>
}
```



การกำหนดสไตล์แบบ External

การกำหนดสไตล์แบบ External เป็นการแยกส่วนของ CSS ไปสร้างเป็นไฟล์ไว้ต่างหาก และในกรณีของ React จะนำสไตล์เข้ามาใช้งานในเพจหรือไฟล์ที่ต้องการ โดยใช้คำสั่ง import ในแบบเดียวกับโมดูล ซึ่งหลักการที่สำคัญมีดังนี้

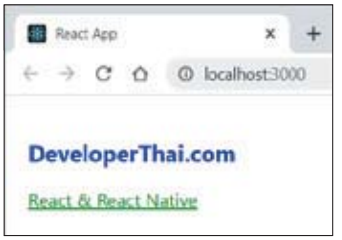
- สร้างไฟล์ใหม่เพิ่มเข้ามาในแอป โดยให้มีส่วนขยายเป็น .css แล้วภายในไฟล์ดังกล่าว จะต้องมีเฉพาะโค้ด CSS เท่านั้น ห้ามมีแท็ก <style> รวมอยู่ด้วย
- ไฟล์หรือเพจที่จะนำสไตล์ไปใช้งาน ให้นำเข้าด้วยคำสั่ง import './ชื่อไฟล์ CSS' เช่น สมมติว่าต้องการนำเข้าไฟล์ที่ชื่อ global.css ก็กำหนดโค้ดเป็น import './global.css'
- สไตล์ที่นำเข้ามาจะมีผลกับแท็ก HTML ที่ตรงกับ Selector ซึ่งระบุไว้ในไฟล์ CSS แม้จะเขียนในแบบ JSX ก็ตาม

react/app1-1/src/style.css

```
h2 {
      font-size: 20px;
      color: blue;
}

.title {
      font-size: 16px;
      color: green;
      text-decoration: underline;
}
```

react/app1-1/src/App.js

```
import React from 'react'
import './style.css'

function App() {
     return (
          <>
               <h2>DeveloperThai.com</h2>
               <div className="title">React & React Native</div>
          </>
     )
}
export default App
```



การใช้เมธอด Array.map() ร่วมกับ JSX

หากเราต้องการเข้าถึงเพื่อดำเนินการกับสมาชิกทั้งหมดในอาร์เรย์ ซึ่งตามปกติก็มีลูป for แบบต่าง ๆ รวมถึงเมธอด forEach() ของอาร์เรย์ให้ใช้งานอยู่แล้ว แต่เนื่องจากลูป for รวมถึงเมธอด forEach() ไม่คืนค่ากลับมา จึงไม่สะดวกต่อการใช้งานร่วมกับ React JSX อย่างไรก็ตาม อาร์เรย์ใน JavaScript ยังมีเมธอด map() ที่ทำงานคล้ายกับ forEach() ดังรูปแบบต่อไปนี้

```
map(function(item) {หรือ     map(function(item, index) { ...})
    ...
    return ...
})
```

หรืออาจเขียนในแบบ Arrow Function เป็น

```
map((item) => {  หรือ map(item => { ... })  หรือ map((item, index) => { ... })
    ...
    return ...
})
```

- item คือค่าของสมาชิกในการวนลูปแต่ละรอบ
- index อาจระบุเมื่อเราต้องการอ้างถึงลำดับของสมาชิก
- ภายในบล็อกของ map() ต้องมีคำสั่ง return เพื่อส่งค่าในแต่ละรอบกลับออกไป ยกเว้นกรณีที่ใช้ Arrow Function หากมีเพียงคำสั่งเดียว อาจไม่ระบุ return ก็ได้

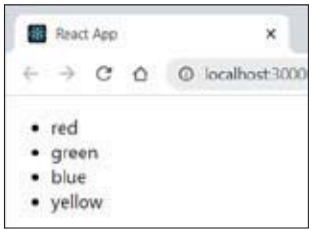
จะเห็นได้ว่า map() จะคล้ายกับ forEach() แต่ข้อแตกต่างที่สำคัญคือ **เมธอด map() จะคืนค่าผลลัพธ์ในแต่ละรอบของการวนลูปกลับคืนมา** โดยลักษณะปกติของ map() ใน JavaScript นั้น จะคืนค่ากลับมาเป็นอาร์เรย์ของผลลัพธ์ที่ได้ ดังโค้ดต่อไปนี้

```
let colors = ['red', 'green', 'blue', 'yellow']
let list = colors.map(c => `<li>${c}</li>`)
//list = ['<li>red</li>', '<li>green</li>', ...]
```

แต่เมื่อนำมาใช้กับ JSX อาร์เรย์จะถูกแปลงเป็นสตริงเดียวกันหรือนำสมาชิกแต่ละตัวมาเชื่อมต่อกันนั่นเอง ตามหลักการของ JSX ที่ได้กล่าวไว้แล้ว ซึ่งถือว่าเป็นประโยชน์อย่างมาก เพราะเราไม่ต้องเสียเวลาอ่านค่าสมาชิกมาเชื่อมต่อกันทีละตัว ดังแนวทางในโค้ดต่อไปนี้

react/app1-1/src/App.js

```
import React from 'react'
function App() {
     let colors = ['red', 'green', 'blue', 'yellow']
     let list = colors.map(c => <li>{c}</li>)      //แทรกค่าแบบ JSX
     //list = <li>red</li><li>green</li>...
     return <ul>{list}</ul>

     //หรือเรียกเมธอด map() ใน JSX โดยตรง
     /*
     return (
          <ul>
               {colors.map(c => <li>{c}</li>)}
          </ul>
     )
     */
}
```



ในบทต่อ ๆ ไปเรา เราจะได้ใช้งานเมธอด map() อยู่บ่อย ๆ ซึ่งในบทนี้ เพียงแค่แนะนำหลักการในเบื้องต้นให้ทราบล่วงหน้าเอาไว้ก่อนเท่านั้น

การแสดงรูปภาพใน React App

รูปภาพ คือองค์ประกอบที่สำคัญอย่างหนึ่งซึ่งเราจำเป็นต้องนำมาแสดงผลในเว็บเพจอยู่บ่อย ๆ และสำหรับใน React นั้น ก็มีวิธีการจัดเก็บและอ้างถึงตำแหน่งรูปภาพภายในแอป ดังนี้

- ตามหลักการแล้ว เราควรจัดเก็บรูปภาพในโฟลเดอร์ public หรืออาจสร้างโฟลเดอร์ย่อยๆ ซ้อนไว้ใน public เพื่อแยกเก็บรูปภาพให้เป็นสัดส่วนจากไฟล์อื่นๆ ก็ได้ เช่น public/images เป็นต้น
- กรณีที่เราจัดเก็บภาพไว้ในโฟลเดอร์ public หากต้องการอ้างถึงภาพในโค้ด JSX ก็อาจเลือกใช้วิธีใดวิธีหนึ่งดังนี้ (สมมติว่าต้องแสดงภาพที่ชื่อ logo192.png ซึ่งอยู่ใน public)

react/app1-1/src/App.js

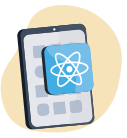
```
function App() {
      //เลือกใช้แบบใดแบบหนึ่ง
      //return <img src="./logo192.png" alt=""/>
      //return <img src={"./logo192.png"} alt=""/>
      //return <img src={(process.env.PUBLIC_URL + "/logo192.png")} alt=""/>

      //ถ้าจัดเก็บภาพในโฟลเดอร์ย่อยๆ ซ้อนอยู่ใน public ก็ให้ระบุโฟลเดอร์นั้นด้วย เช่น
      //ถ้าเก็บภาพไว้ใน public/images ก็อ้างอิงตำแหน่งเป็น
      //return <img src="./images/logo192.png" alt=""/>
}
```

- หรือถ้าเราจัดเก็บภาพในโฟลเดอร์ src การอ้างถึงตำแหน่งภาพในโค้ด JSX ก็ทำดังนี้ (สมมติว่าต้องการแสดงภาพที่ชื่อ logo.svg ซึ่งอยู่ในโฟลเดอร์ src)

react/app1-1/src/App.js

```
import logo from './logo.svg'        //ภาพอยู่ที่ src/logo.svg

function App() {
      return <img src={logo} width="10%" alt=""/>
}
```

สิ่งที่เราได้ศึกษาในบทนี้ ทั้งวิธีการเขียนโค้ด HTML และ CSS ในรูปแบบ JSX รวมถึงลักษณะปลีกย่อยอื่นๆ จะเป็นพื้นฐานสำคัญสำหรับใช้ในการพัฒนา Web Application ด้วย React ที่จะกล่าวถึงในบทต่อๆ ไป



พัฒนา Application ด้วย React และ React Native

ในปัจจุบัน กลุ่มนักพัฒนาต่างหันมาให้ความสนใจแนวทางการสร้างแอปแบบผสมผสาน (Hybrid) กันมากขึ้น ซึ่งเครื่องมือที่ได้รับความนิยมสูงสุดก็คือ React/React Native โดยการสร้างแอปขึ้นมาเพียงชุดเดียว แล้วสามารถแปลงเป็น Web App และ Mobile App สำหรับทั้งระบบ iOS และ Android ได้ทันที ซึ่งข้อได้เปรียบของ React/React Native ก็คือ การใช้ JavaScript เป็นหลักเพียงภาษาเดียวในการเขียนโค้ด ดังนั้นจึงสามารถเรียนรู้ได้ด้วยระยะเวลาอันรวดเร็ว และแอปที่พัฒนาขึ้นก็ทำงานตรงกันในทุกแพลตฟอร์ม นอกจากนี้ยังมีไลบรารีที่สร้างมาสนับสนุนการทำงานให้เลือกใช้มากมาย เราจึงมีแนวทางในการพัฒนาที่หลากหลายมากยิ่งขึ้น

เนื้อหาในหนังสือประกอบด้วย :

www.se-ed.com sbc.fans SE-ED Publisher

พร้อมจำหน่ายในรูปแบบ
☑ e-book (PDF) ☐ audiobooks
☐ e-book (EPUB) ☐ audio CD / MP3
☑ ปกอ่อน ☐ LARGE PRINT (ตัวอักษรขนาดใหญ่)

ISBN 978-616-08-4322-0
9 786160 843220
330 บาท

คอมพิวเตอร์-การเขียนโปรแกรม