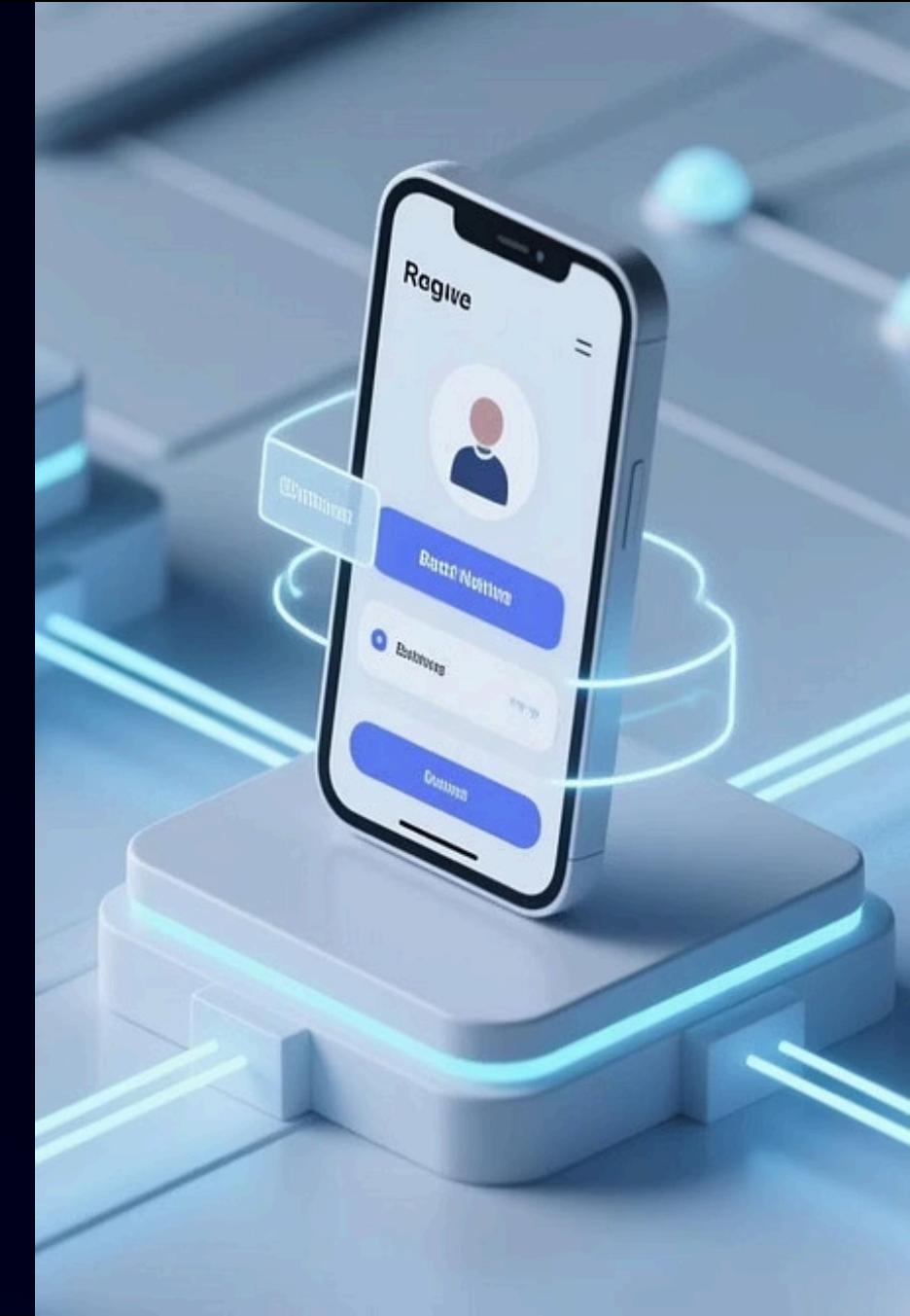


การพัฒนาโปรแกรมประยุกต์สำหรับ  
อุปกรณ์เคลื่อนที่ขั้นสูง

บทที่ 2: พื้นฐาน React Native

โดย อ.วรวุธ จันทริก



# การรวมและเป้าหมายของหลักสูตร

เป้าหมายหลักของหลักสูตรนี้คือการสร้างรากฐานที่แข็งแกร่งและครบวงจรในการพัฒนาแอปพลิเคชันมือถือด้วย React Native ให้กับผู้เรียนทุกคน แม้ว่าคุณจะไม่มีพื้นฐานในการเขียนโค้ดมาก่อนก็ตาม เราจะพาคุณเรียนรู้ตั้งแต่เบื้องต้นไปจนถึงความลึกซึ้งของ React Native ไปจนถึงการสร้างแอปที่มีฟังก์ชันการทำงานซับซ้อนได้จริง

ข้าใจในหลักการสำคัญของ React Native และมีทักษะที่จำเป็นในการต่อยอดความรู้เพื่อพัฒนาแอปพลิเคชันระดับมืออาชีพได้อย่างมั่นใจ ไม่ว่าจะเป็นแอปพลิเคชันส่วนตัว หรือแอปพลิเคชันสำหรับองค์กรใหญ่ ๆ หลักสูตรนี้จะช่วยให้คุณก้าวแรกสู่การเป็นนักพัฒนาแอปพลิเคชันที่ประสบความสำเร็จ





## แอปพลิเคชันมือถือ คือ

"แอป" คือโปรแกรมที่เราใช้บนอุปกรณ์พกพา เช่น โทรศัพท์มือถือ แท็บเล็ต หรือนาฬิกาอัจฉริยะ แอปช่วยให้เราทำงานต่างๆ ได้ง่ายและเร็วขึ้น ไม่ว่าจะคุยกัน ทำงาน หรือเล่นสนุก

แอปมือถือเริ่มเป็นที่นิยมมากในช่วงปลายปี 2000 เพราะมีสมาร์ทโฟนรุ่นแรกๆ อย่าง iPhone (ของ Apple) และ Android ออกมานั่นทำให้นักพัฒนาสามารถสร้างแอปที่ซับซ้อนและบ่าสูนใจได้มากขึ้น

# ประเภทของแอปมือถือ

แอปมือถือที่เราใช้ทุกวันนี้ แบ่งเป็น 3 แบบหลักๆ โดยแต่ละแบบก็มีข้อดีข้อเสียต่างกัน:

## Web Application

แอปที่เปิดใช้ในเว็บเบราว์เซอร์บนมือถือ ไม่ต้องติดตั้งในเครื่อง แต่ต้องต่อเน็ต และอาจใช้ฟังก์ชันบางอย่างของมือถือได้ไม่ครบ

## Native Application

แอปที่สร้างมาเพื่อระบบปฏิบัติการนั้นๆ โดยเฉพาะ เช่น iOS หรือ Android ทำให้ทำงานเร็ว และใช้ทุกฟังก์ชันของมือถือได้เต็มที่ แต่ต้องสร้างแยกสำหรับแต่ละระบบ

## Hybrid Application

แอปที่ใช้เทคโนโลยีเว็บมาช่วยให้สร้างครึ่งเดียว ใช้ได้กับ iOS และ Android แต่ประสิทธิภาพและการเข้าถึงฟังก์ชันต่างๆ อาจไม่ดีเท่า Native App

# Native



# ส่วนประกอบหลักของแอป

แอปพลิเคชันมือถือที่คุณใช้ มีส่วนประกอบสำคัญที่ทำงานร่วมกัน เพื่อให้แอปทำงานได้ดีและลื่นไหล



## Backend (หลังบ้าน)

ส่วนที่แอปทำงานอยู่เบื้องหลัง จัดการข้อมูล เก็บข้อมูลอย่างปลอดภัย และควบคุมการทำงานของระบบทั้งหมด



## Frontend (หน้าบ้าน)

ส่วนที่คุณเห็นและใช้งานบนหน้าจอ เช่น ปุ่ม, ข้อความ, รูปภาพ หรือซ่องให้กรอกข้อมูล



## API (ตัวเชื่อม)

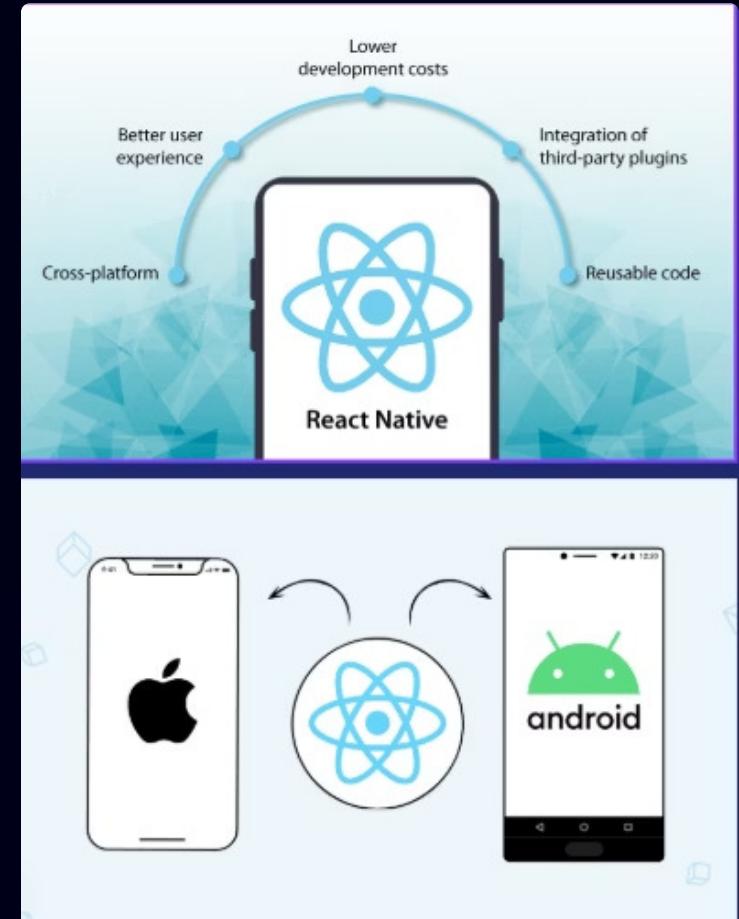
ตัวกลางที่ช่วยให้ส่วน "หน้าบ้าน" (Frontend) และ "หลังบ้าน" (Backend) คุยกันและแลกเปลี่ยนข้อมูลกันได้



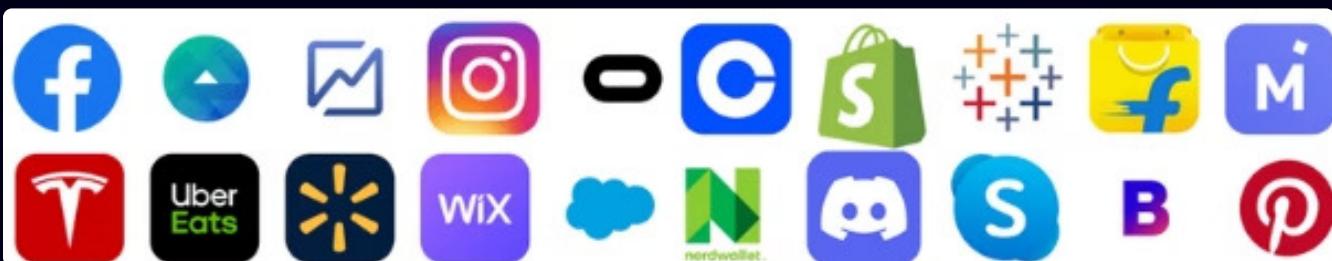
# React Native คืออะไร?

React Native คือเครื่องมือที่ช่วยให้คุณสามารถสร้างแอปมือถือที่ทำงานได้ดีเหมือนแอปที่เขียนขึ้นมาเพื่อเครื่องนั้นๆ โดยใช้ภาษา JavaScript

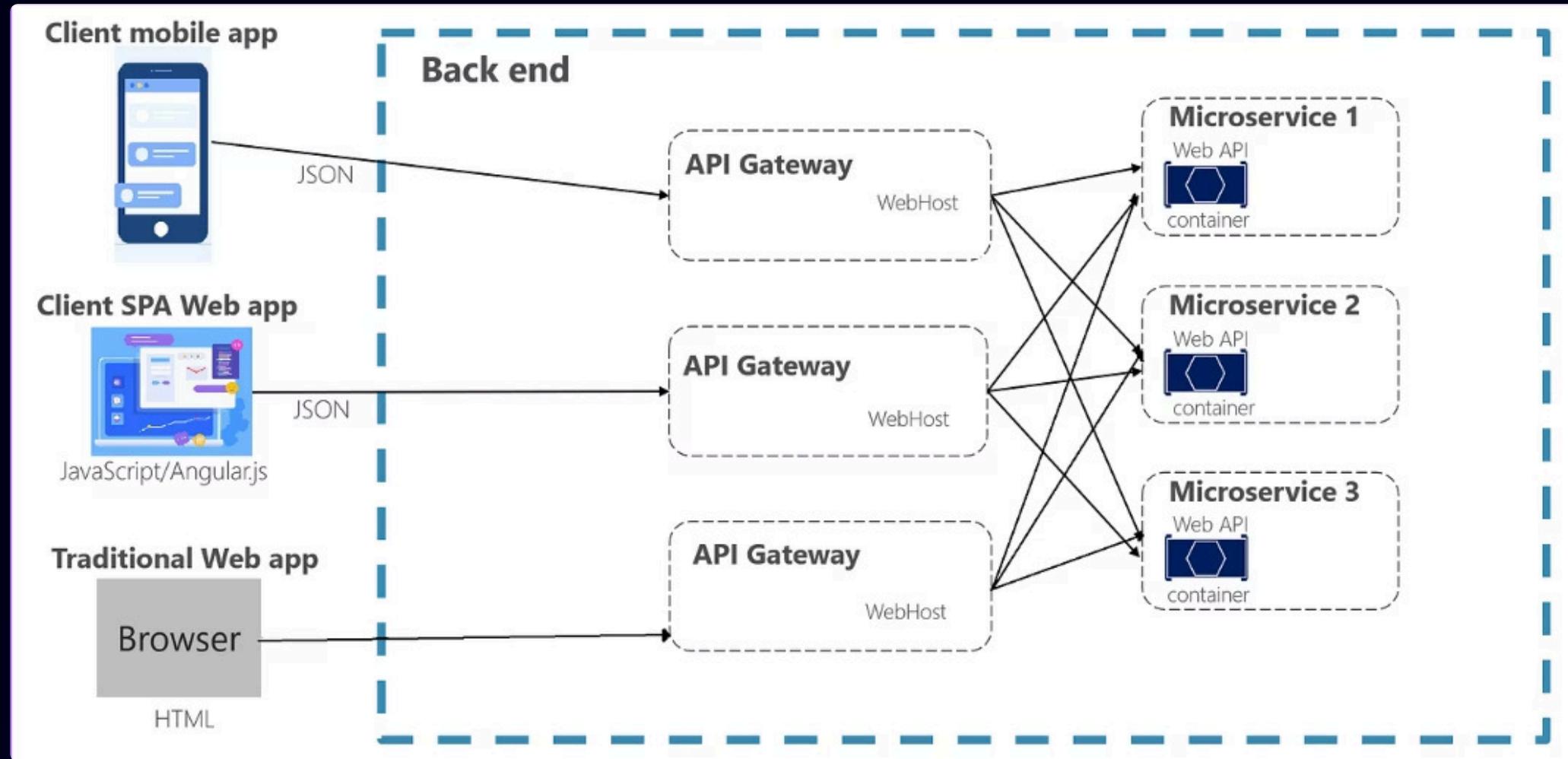
ข้อดี: คุณเขียนโค้ดแค่ครั้งเดียว แต่เอาไปใช้สร้างแอปได้ทั้งบน iPhone (iOS) และ Android ช่วยประหยัดเวลาและค่าใช้จ่ายในการทำแอป



ตัวอย่างแอปดังๆ: Facebook, Instagram, Discord ก็ใช้ React Native ในการสร้างแอปพลิเคชันของพวกรา



# โครงสร้างพื้นฐานของการพัฒนาแอปพลิเคชัน



# The Launchpad

เตรียมเครื่องมือและสิ่งจำเป็น

## เป้าหมาย

- ติดตั้งเครื่องมือ
- เข้าใจพื้นฐาน JavaScript
- รันโปรเจกต์แรกได้

## ผลลัพธ์

คุณจะรับแอป "Hello World" บนมือถือได้ และเข้าใจพื้นฐานที่ใช้ในการพัฒนาแอป React Native



# เครื่องมือที่ต้องติดตั้ง



## Node.js LTS

Node.js LTS เป็นโปรแกรมสำคัญที่ช่วยให้คอมพิวเตอร์ของเราเข้าใจและรันโค้ด JavaScript ได้ ซึ่งเป็นภาษาหลักที่ใช้ใน React Native นอกจากนี้ Node.js ยังมาพร้อมกับ npm ที่ช่วยให้เราติดตั้งโปรแกรมเสริมต่างๆ ที่จำเป็นสำหรับการสร้างแอปได้อย่างง่ายดาย



## Git

Git คือระบบที่ช่วยเราติดตามการเปลี่ยนแปลงของโค้ดโปรแกรม ทำให้เราสามารถย้อนกลับไปดูเวอร์ชันเก่าๆ ได้ หรือทำงานร่วมกับเพื่อนในโปรเจกต์เดียวกันได้อย่างราบรื่น มันช่วยให้การพัฒนาซอฟต์แวร์เป็นไปอย่างมีระเบียบและลดความผิดพลาด

01  
10

## Visual Studio Code

Visual Studio Code (VS Code) คือโปรแกรมสำหรับเขียนโค้ดที่เราใช้ในการสร้างแอป React Native มันเป็นที่นิยม เพราะใช้งานง่าย มีเครื่องมือช่วยเขียนโค้ด และมีส่วนเสริมมากมายที่ทำให้การทำงานสะดวกขึ้น ทั้งการเขียนโค้ด การแก้ข้อผิดพลาด และการทำงานร่วมกับคนอื่น



## Expo Go

Expo Go เป็นแอปบนมือถือ (ทั้ง iOS และ Android) ที่ทำให้เราสามารถทดสอบและ Run React Native ที่กำลังสร้างอยู่ได้แบบทันที ไม่ต้องรอนานเหมือนวิธีปกติ มันช่วยให้เราเห็นการเปลี่ยนแปลงของแอปบนมือถือได้เร็วขึ้น ทำให้การพัฒนาและแก้ไขแอปง่ายและสะดวกมากสำหรับนักพัฒนาที่เริ่มต้น

# ทำไมต้องใช้ Expo?

Expo เป็นชุดเครื่องมือที่ทำให้คุณเริ่มสร้างแอป React Native ได้ง่ายและเร็ว ไม่ต้องยุ่งยากกับการตั้งค่าที่ซับซ้อน เมื่อ用มีผู้ช่วยส่วนตัวมาจัดการให้



## ตั้งค่ารวดเร็ว

Expo จัดการการตั้งค่าเบื้องต้นให้ ทำให้คุณบุ่มเนิน การเขียนโค้ดได้กันที



## ประหยัดพื้นที่

ไม่ต้องติดตั้งโปรแกรมใหญ่อย่าง Xcode หรือ Android Studio ช่วยประหยัดพื้นที่และเวลาติดตั้ง



## ทดสอบบนมือถือได้กันที

เพียงดาวน์โหลดแอป Expo Go และสแกน QR Code คุณก็สามารถทดสอบแอปบนมือถือของคุณได้เลย



## แสดงผลแบบเรียลไทม์

ฟังก์ชัน Live Reload ทำให้คุณเห็นการเปลี่ยนแปลงของโค้ดในแอปได้กันที ไม่ต้องรันใหม่ ทุกครั้ง



## เครื่องมือและ API ครบครัน

Expo มีชุดคำสั่ง (API) และเครื่องมือสำหรับรูปภาพmany ( เช่น กล้อง, ตำแหน่ง, การแจ้งเตือน) ให้คุณนำไปใช้ในแอปได้อย่างง่ายดาย



## เผยแพร่แอปง่ายๆ

เมื่อพัฒนาแอปเสร็จ การนำแอปขึ้น App Store หรือ Play Store ก็ทำได้ง่ายขึ้นมาก ด้วยคำสั่งไม่กี่คำสั่ง



# Extention ที่จำเป็นใน VS Code

## ES7+ React/Redux/React-Native snippets

ช่วยพิมพ์โค้ด React ได้รวดเร็ว เช่น พิมพ์ `rnf` และกด Tab จะได้โครงสร้าง Component

## Prettier Code formatter

จัดระเบียบโค้ดให้อัตโนมัติ ทำให้โค้ดอ่านง่ายและสวยงาม

## ESLint

ช่วยตรวจจับข้อผิดพลาดในโค้ด ทำให้เขียนโค้ดได้มีคุณภาพมากขึ้น

## GitLens

เพิ่มความสามารถให้กับการใช้ Git ใน VS Code ช่วยให้ดูประวัติการเปลี่ยนแปลงได้ง่าย

# พื้นฐาน JavaScript และ ES6

ในการพัฒนาแอปพลิเคชันด้วย React Native การมีความเข้าใจพื้นฐานเกี่ยวกับ JavaScript โดยเฉพาะคุณสมบัติใหม่ๆ ที่มาจากการ ES6 (ECMAScript 2015) เป็นสิ่งสำคัญอย่างยิ่ง

## let (ตัวแปรที่เปลี่ยนแปลงได้)

ใช้เมื่อคุณต้องการตัวแปรที่สามารถเปลี่ยนค่าได้หลังจากที่ประกาศ ทำให้มีความยืดหยุ่นในการเขียนโค้ดมากขึ้น

```
let age = 25;  
age = 26; // เปลี่ยนค่าได้  
console.log(age); // Output: 26
```

## const (ค่าคงที่)

ใช้สำหรับประกาศตัวแปรที่ไม่สามารถเปลี่ยนค่าได้หลังจากที่ประกาศแล้ว หมายเหตุสำหรับค่าที่ไม่ควรเปลี่ยนแปลงตลอดการทำงานของโปรแกรม

```
const pi = 3.14159;  
// pi = 3.14; // Error: Assignment to constant variable
```

## Arrow Functions (ฟังก์ชันแบบลูกศร)

เป็นฟังก์ชันรูปแบบใหม่ใน ES6 ที่ช่วยให้โค้ดดูสะอาดและกระชับขึ้น โดยไม่ต้องใช้คำว่า function เมื่อ用 ฟังก์ชันแบบปกติ นอกจากนี้ยังมีการจัดการ this ที่แตกต่างกัน ซึ่งนิยมใช้ใน React

```
// รูปแบบเดิม  
function add(a, b) {  
  return a + b;  
}  
console.log(add(3, 5)); // Output: 8  
  
// รูปแบบ Arrow Function  
const add = (a, b) => a + b;  
console.log(add(3, 5)); // Output: 8
```

## Destructuring (การแยกโครงสร้าง)

เป็นฟีเจอร์ที่ช่วยให้เราสามารถดึงค่าจากออบเจกต์ (object) หรืออาร์เรย์ (array) ได้อย่างง่ายดาย โดยไม่ต้องเข้าถึงชื่อ屬性 หรือ [] หลายครั้ง ช่วยให้โค้ดอ่านง่ายขึ้น

```
const person = { name: 'John', age: 30, city: 'Bangkok' };  
  
// ใช้ Destructuring เพื่อดึงค่าจากออบเจกต์  
const { name, age } = person;  
console.log(name); // Output: John  
console.log(age); // Output: 30
```

# Workshop: สร้างโปรเจกต์แรก

## ติดตั้งเครื่องมือ

- ติดตั้ง Node.js LTS จาก [nodejs.org](https://nodejs.org)
- ติดตั้ง VS Code และ Extensions ที่แนะนำ
- ติดตั้ง Git จาก [git-scm.com](https://git-scm.com)
- ติดตั้งแอป Expo Go บนมือถือ

## ตรวจสอบการติดตั้ง

เปิด Terminal/Command Prompt และรันคำสั่ง:

- `node -v` (ควรแสดงเวอร์ชัน เช่น v18.18.0)
- `npm -v` (ควรแสดงเวอร์ชัน เช่น 9.8.1)
- ติดตั้ง Expo CLI: `npm install -g expo-cli` (หรือใช้ `npx expo` ได้โดยไม่ต้องติดตั้ง global)
- `npx expo --version` (ควรแสดงเวอร์ชันของ Expo CLI)

## สร้างและรันโปรเจกต์แรก

- สร้างโปรเจกต์ใหม่: `npx expo init rn-bootcamp` (เลือก template blank)
- เข้าไปในโฟลเดอร์โปรเจกต์: `cd rn-bootcamp`
- `npm install react-native-elements`
- เริ่มการทำงานของแอป: `npx expo start`
- เปิดแอป Expo Go บนมือถือ และสแกน QR Code



## Workshop: Hello World!

ເປີດໄຟລ໌ App.js ແລ້ວໂຄດເດີນກົ່ງກັ້ງໜົດ ແລ້ວພິມພາໄດ້ນີ້ເຂົ້າໄປ:

```
import React from 'react';
import { StyleSheet, Text, View } from 'react-native';

export default function App() {
  return (
    ສະບັບ React Native!
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#fff',
    alignItems: 'center',
    justifyContent: 'center',
  },
  text: {
    fontSize: 24,
    fontWeight: 'bold',
  },
});
```

ເນື່ອກຳດັບນົກໄຟລ໌ (Save) ແລ້ວບັນນຸ້ມຂອງຈະອັບເດີແທນຈະກັນທີ ນີ້ຄ້ອງຄຸນສົມບັດທີ່ເຮັດວຽກກວ່າ Hot Reload



# Your First Components

สร้างหน้าจอและสไตร์แกรก

## เป้าหมาย

- ทำความเข้าใจ JSX, Core Components
- เรียนรู้ Props, State และ Hooks
- สร้าง UI ที่โต้ตอบกับผู้ใช้ได้

## ผลลัพธ์

สามารถสร้าง UI ที่มีการโต้ตอบกับผู้ใช้ได้ เช่น แอป Counter และฟอร์มรับซื้อ

# เปรียบเทียบ React Native กับ ชีวิตจริง: เปิดร้านกาแฟ



**JSX:** ผังร้าน/คู่มือจัดวาง

เหมือน 'แบบแปลน' ที่บอกว่าสิ่งต่างๆ ในร้าน เช่น เคาน์เตอร์ โต๊ะ ป้ายเมนู ควรจะวางอยู่ตรงไหนบ้าง ในโค้ด JSX ก็จะบอกว่าส่วนต่างๆ ของแอปควรจะวางอย่างไร



**Core Components:** ของจริงในร้าน

คือ 'สิ่งของจริงๆ' ที่ใช้ในร้าน เช่น เคาน์เตอร์ โต๊ะ ป้ายข้อความ รูปภาพ หรือบุ้มกดสั่งเครื่องดื่ม



**Props:** ทางเลือก/คำสั่งเฉพาะชิ้น

เหมือน 'คำสั่งเฉพาะ' สำหรับแต่ละรายการ เช่น สั่ง "ลาเต้ แก้วใหญ่ หวานน้อย" คอมโพเนนต์จะรับคำสั่งนี้ไปใช้ แต่ตัวมันเองไม่สามารถเปลี่ยนคำสั่งนี้ได้



**State:** สถานะปัจจุบันในร้าน

คือ 'สถานะปัจจุบัน' ของร้าน เช่น คิวลูกค้า หรือจำนวนกาแฟที่กำลังเสร็จแล้ว เมื่อเมื่อไรเปลี่ยนแปลง (เช่น ลูกค้าสั่งเพิ่ม) สถานะนี้ก็จะเปลี่ยนตามและแสดงผลให้เห็น

# ເປີຍບເກີບກັບຊົວຕຈຣິງ: ເປີດຮ້ານກາແພ



## Hooks: ຜູ້ຂ່ວຍອັຕໂນມັຕີ/ຮະບບໍລັງຮ້ານ

ເໝັ້ນ 'ຜູ້ຂ່ວຍອັຕໂນມັຕີ' ຮັບອື່ນ 'ຮະບບໍລັງຮ້ານ' ທີ່ມີຄວາມສຳເນົາໃຫ້ຮ້ານກາແພຂອງຄຸນເດີນທີ່ໄດ້ຢ່າງຮາບຮັນແລະມີປະສິກທິກາພ



## AsyncStorage/MMKV: ຕູ້ເໜີ/ສຸມຸດບັນຫຼື

ຄວາມ 'ຕູ້ເໜີ' ຮັບອື່ນ 'ສຸມຸດບັນຫຼື' ຂອງຮ້ານທີ່  
ເກີບຂ້ອງມູລສຳຄັງໄວ້ ເນື່ອປັດຮ້ານແລ້ວ  
ຂ້ອມຍັງອູ່ ເປີດວັນໃໝ່ກີ່ຍັງສາມາດ  
ເຮັດວຽກໃຫ້ຂ້ອມມູລເດີນໄດ້ (ເຊັ່ນ ປະວັດຕອອດ  
ອົບ ຮັບອື່ນກັບຮ້ານກາແພ)



## Navigation: ກາງເດີນ/ປ້າຍບອກກາງ

ເໝັ້ນ 'ກາງເດີນ' ຮັບອື່ນ 'ປ້າຍບອກກາງ'  
ທີ່ມີຄວາມສຳເນົາໃຫ້ລູກຄ້າຮັດວຽກ  
ຢັງໂປບຕ່າງໆ ໃນຮ້ານໄດ້ອ່າຍ່າງຄຸກຕ້ອງ  
(ເຊັ່ນ ການປັບປຸງທີ່ຮ້ານກາແພ)



## API: ວິຖຸສື່ສາຮ້ານກາແພ

ເໝັ້ນ 'ວິຖຸສື່ສາຮ້ານ' ທີ່ຮ້ານໃຊ້ຕິດຕ່ອງກັບ  
ໂລກກາຍນອກ ເພື່ອຂອບຂ້ອງມູລຮັດວຽກ  
ສັ່ງໄປຢັງກີ່ອັນ (ເຊັ່ນ ໂກຮ້າຫັກພລາຍເອ  
ອຣ ເພື່ອສັ່ງວັດຄຸດົບ ຮັບອື່ນກັບຮ້ານກາແພ)

# เปรียบเทียบกับชีวิตจริง: เปิดร้านกาแฟ



**Camera/ImagePicker:** กล้องถ่ายรูปสินค้า/ใบเสร็จ

ใช้ถ่ายรูปสินค้า หรือบันทึกใบเสร็จ



**Location/Maps:** แผนที่ร้าน/ตำแหน่งลูกค้าเดลิเวอรี่

ใช้บอกร้านที่ตั้ง หรือหาที่ส่งของให้ลูกค้า

**Error>Loading/Empty state:** ป้ายแจ้งสถานะหน้าร้าน

เหมือนป้ายที่บอกสถานะ เช่น "กำลังทำ" (loading), "มีปัญหา" (error), หรือ "ไม่มีอยู่เดอ" (empty) ให้ลูกค้า

ดู

**Deployment/Updates:** เปิดร้านจริง/อัปเดตเมนู

การทำแอปเสร็จ คือการเปิดร้านจริง ส่วนการอัปเดตแบบ OTA คือเปลี่ยนเมนูหรือราคาได้ทันที ไม่ต้องปิดร้าน หาก

# JSX คืออะไร?

**JSX** ย่อมาจาก **JavaScript XML** มันไม่ใช่ภาษาใหม่ แต่เป็น วิธีพิเศษ ใน การเขียน JavaScript ที่ทำให้เราสามารถใส่โค้ดที่ดูเหมือน HTML ลงไปในไฟล์ JavaScript ได้เลย เพื่อบอกว่าหน้าตาของแอปเราจะเป็นอย่างไร

**เปรียบเทียบง่ายๆ:** ลองนึกว่า JSX คือ "แบบร่าง" ของหน้าจอแอป ก็คุณสร้างขึ้นมาด้วยโค้ดที่อ่านง่าย (คล้าย HTML) จากนั้น React ก็จะนำแบบร่างนี้ไปสร้างเป็นหน้าจอแอปจริงๆ ให้เห็น

**กฎสำคัญ:** Component จะต้องแสดงผล (return) JSX ที่มี องค์ประกอบหลักเพียง 1 ตัว เท่านั้น ถ้ามีหลายองค์ประกอบ จะต้องมีตัวครอบ เช่น `<View>...</View>` หรือ `<>...</>` (Fragment)

```
// ถูกต้อง  
return (
```

สวัสดี  
React Native

```
);
```

```
// ไม่ถูกต้อง  
return (  
สวัสดี  
React Native  
);
```

# React Component คืออะไร?

**React Component** คือ 'หน่วยย่อย' กี่เป็นอิสระและนำกลับมาใช้ใหม่ได้ ซึ่งทำหน้าที่ในการแสดงผลส่วนหนึ่งของ UI (User Interface) ของแอปพลิเคชัน พุดง่ายๆ คือ เป็นเหมือนบล็อกตัวต่อที่เราสามารถสร้างขึ้นมาเอง เพื่อประกอบรวมกันเป็นหน้าจอแอปฯ ที่ซับซ้อนได้



## ชิ้นส่วนสำคัญ

เปรียบเสมือน 'โมดูลสำคัญ' ในร้านกาแฟ เช่น 'สถานีรับออร์เดอร์อัตโนมัติ' ที่มีหน้าจอ บุ้น และเครื่องรับบัตร ซึ่งสามารถนำไปวางใช้งานที่ไหนก็ได้ในร้าน หรือแม้แต่ในสาขาอื่น



## รับข้อมูลและแสดงผล

คอมโพเนนต์จะรับข้อมูลที่เรียกว่า **Props** (คุณสมบัติภายนอก) และ **State** (สถานะภายในที่เปลี่ยนแปลงได้) เพื่อกำหนดว่าตัวเองจะแสดงผลอะไร เมื่อันกับที่สถานีรับออร์เดอร์สามารถแสดง 'เมนูวันนี้' (Props) หรือ 'คิวลูกค้าปัจจุบัน' (State) ได้



## ทำงานอิสระ

แต่ละคอมโพเนนต์ทำงานแยกจากกัน ทำให้ง่ายต่อการพัฒนา แก้ไข และบำรุงรักษาโดยไม่กระทบส่วนอื่น เมื่อันแต่ละ 'สถานีบริการ' ในร้านกาแฟทำงานเป็นของตัวเองได้ แต่ก็ประสานงานกันเพื่อสร้างประสบการณ์ที่ดีที่สุดให้ลูกค้า

# Core Components: สำหรับสร้างแอป

## <View>

คืออะไร: เป็น "กล่อง" องค์ประกอบที่พื้นฐานที่สุด ใช้สำหรับจัดกลุ่มและจัดวาง (Layout) Component อื่นๆ

เปรียบเทียบ: เหมือนแท็ก <div> ในเว็บ

## <Text>

คืออะไร: ใช้สำหรับแสดงข้อความทุกชนิด ข้อความทั้งหมดใน React Native ต้องอยู่ภายใน <Text> เท่านั้น

เปรียบเทียบ: เหมือนแท็ก <p>, <h1>, <span> ในเว็บ

## <TextInput>

คืออะไร: กล่องสำหรับให้ผู้ใช้กรอกข้อความ

เปรียบเทียบ: เหมือนแท็ก <input> ในเว็บ

## <TouchableOpacity>

คืออะไร: Component ที่ทำให้ Component ที่อยู่ข้างใน (เช่น <View>, <Text>) สามารถ "กด" ได้ เมื่อคลิกแล้วจะมีความโปร่งใสเล็กน้อย (Opacity)

ทำไมต้องใช้: เป็นที่นิยมมาก เพราะเราสามารถสร้างปุ่มที่ปรับแต่งสไตล์ได้อย่างอิสระ

# Core Components ที่สำคัญ: ส่วนที่ 2

## SafeAreaView

เป็น Component ที่สำคัญสำหรับ การแสดงผลบนหน้าจออุปกรณ์ที่หลักหลาย ช่วยให้ เนื้อหาของแอปพลิเคชันไม่ถูกบดบังด้วยส่วนเว้า (notch) หรือขอบโค้งของหน้าจอ

```
<SafeAreaView style={styles.container}>
  <Text>ข้อความในพื้นที่ปลอดภัย</Text>
</SafeAreaView>
```

## Image

ใช้สำหรับ แสดงรูปภาพ ในแอปของคุณ สามารถโหลดรูปภาพได้ก็จากไฟล์ในเครื่อง หรือจาก URL ภายนอก

```
<Image
  source={{uri: 'https://picsum.photos/200/200'}}
  style={{width: 100, height: 100}}
/>
```

## ScrollView

ช่วยให้ เมื่อหาที่ยวเกินหน้าจอสามารถเลื่อนได้ หมายความว่าหน้าที่มีข้อมูลจำนวนมาก เช่น บทความหรือฟอร์มยาวๆ

```
<ScrollView>
  <Text>ข้อความจำนวนมาก...</Text>
</ScrollView>
```

## Button

เป็น Component พื้นฐานสำหรับ สร้างปุ่ม ที่สามารถกดได้ ง่ายต่อการใช้งานและตั้งค่า

```
<Button
  title="กดอันนี้!"
  onPress={() => alert('กดแล้ว!')}
/>
```

# Core Components ที่สำคัญ: ส่วนที่ 3

## TextInput

ใช้สำหรับรับข้อมูลจากผู้ใช้ เช่น ช่องกรอกข้อความ สามารถกำหนดค่าต่าง ๆ เช่น `placeholder` หรือการจัดการข้อมูลเมื่อผู้ใช้กรอกข้อมูลได้

```
<TextInput  
  placeholder="กรอกชื่อของคุณ"  
  style={{ borderWidth: 1, padding: 10 }}  
/>
```

## FlatList

เป็น Component ที่มีประสิทธิภาพสำหรับการแสดงรายการข้อมูลจำนวนมากในรูปแบบเลื่อนสำลับ (scrollable list) เมนูสำหรับข้อมูลที่ต้องการการจัดเรียง เช่น รายการสินค้า หรือรายชื่อผู้ใช้

```
<FlatList  
  data={[{key: 'แอปเปิล'}, {key: 'กล้วย'}, {key: 'ส้ม'}]}  
  renderItem={({item}) => <Text>{item.key}</Text>}  
/>
```

## StatusBar

ช่วยในการควบคุมลักษณะของ **Status Bar** บนอุปกรณ์มือถือ (เช่น แอบด้านบนที่แสดงเวลา, แบตเตอรี่, สัญญาณ) คุณสามารถตั้งค่าสี, สไตล์, หรือซ่อน Status Bar ได้

```
<StatusBar  
  barStyle="dark-content"  
  backgroundColor="#ffffff"  
/>
```

## ActivityIndicator

ใช้สำหรับแสดงตัวบ่งชี้การโหลด (**loading spinner**) เมื่อแอปพลิเคชันกำลังดำเนินการบางอย่างที่ต้องใช้เวลา เช่น การเดินข้อมูลจากอินเทอร์เน็ต

```
<ActivityIndicator  
  size="large"  
  color="#0000ff"  
/>
```

# ຕົວຢ່າງການໃຊ້ຈຳນວນ Core Components

ໂຄດຕຳນີ້ແສດງຕົວຢ່າງການນຳ Core Components ທີ່ເຮົາໄດ້ເຮັດວຽກໄປແລ້ວ ມາປະກອບຮຸນກັນເປັນແອປພຶສີເຄັບຫ່າຍໆ ເພື່ອໃຫ້ເຫັນກາພຽບງານການກຳທຳນັບຈິງ ແລະວິທີການຈັດສຳເນົາດ້ວຍ StyleSheet.

```
import { StatusBar } from "expo-status-bar";
import {
  StyleSheet,
  Text,
  View,
  SafeAreaView,
  ScrollView,
  Image,
  TextInput,
  ActivityIndicator,
  FlatList,
  Button,
  TouchableOpacity,
} from "react-native";

export default function App() {
  return (
    <SafeAreaView style={styles.container}>
      <StatusBar style="auto" />

      {/* Header Section */}
      <View style={styles.header}>
        <Text style={styles.title}>ສະໜັບສິດ React Native!</Text>
      </View>

      {/* Main Content */}
      <View style={styles.content}>
        {/* Image Section */}
        <View style={styles.imageContainer}>
          <Image
            source={{ uri: "https://picsum.photos/200/200" }}
            style={styles.image}
          />
        </View>

        {/* Input Section */}
        <View style={styles.inputContainer}>
          <TextInput
            placeholder="ກຣອກຫຼັກຂໍ້ຄວາມ"
            style={styles.textInput}
            placeholderTextColor="#999"
          />
        </View>

        {/* Button Section */}
        <View style={styles.buttonContainer}>
          <TouchableOpacity style={styles.customButton} onPress={() => alert("ກດປູ່ນ")}>
            <Text style={styles.buttonText}>ກດປູ່ນ</Text>
          </TouchableOpacity>
        </View>

        {/* Loading Section */}
        <View style={styles.loadingContainer}>
          <ActivityIndicator size="large" color="#007AFF" />
          <Text style={styles.loadingText}>ກໍາລັງໂທລດ...</Text>
        </View>

        {/* Scrollable Content */}
        <View style={styles.scrollContainer}>
          <Text style={styles.sectionTitle}>ເນື້ອທາງກ່າວມາຮອດເລື່ອນໄດ້</Text>
          <ScrollView style={styles.scrollView} showsVerticalScrollIndicator={false}>
            <Text style={styles.scrollText}>ສະໜັບສິດ React Native!</Text>
            <Text style={styles.scrollText}>ສະໜັບສິດ React Native!</Text>
            <Text style={styles.scrollText}>ສະໜັບສິດ React Native!</Text>
            <Text style={styles.scrollText}>ສະໜັບສິດ React Native!</Text>
            {/* ... other scrollable text ... */}
          </ScrollView>
        </View>

        {/* List Section */}
        <View style={styles.listContainer}>
          <Text style={styles.sectionTitle}>ຮາຍການຜລໄມ</Text>
          <FlatList
            data={[{ key: "ແອປເປົ້າ", }, { key: "ກລັວຍ", }, { key: "ສັບ" }]}
            renderItem={({ item }) =>
              <View style={styles.listItem}>
                <Text style={styles.listText}>{item.key}</Text>
              </View>
            }
            keyExtractor={(item) => item.key}
            style={styles.flatList}
          />
        </View>
      </SafeAreaView>
    );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: "#f8f9fa",
  },
  header: {
    backgroundColor: "#007AFF",
    paddingVertical: 20,
    paddingHorizontal: 16,
    alignItems: "center",
  },
  title: {
    color: "#fff",
    fontSize: 24,
    fontWeight: "bold",
  },
  // ... other styles ...
});
```

ຈາກໂຄດຕຳນີ້ ເຮັດວຽກໄດ້ຢ່າງດັດເຈນ:

## Header Section

ໃຊ້ <View> ແລະ <Text> ເພື່ອສ້າງລ່ວມຫຼັກ (Header) ຂອງແອປພຶສີເຄັບພ້ອມຂ້ອງຄວາມຕືອນຮັບ

## Image Section

ແສດງຮູບພາບໄດ້ໃໝ່ <Image> Component ສ້າງສາມາດກຳທຳນັບສຳໄຕໄດ້

## Input Section

ສ່ອງສໍາຮັບບັນຫຼວງຈາກຜູ້ໃຊ້ດ້ວຍ <TextInput> ພັດທິນ ພັດທິນ ແລະສຳເນົາດ້ວຍ <placeholder> ແລະສຳເນົາດ້ວຍ <placeholderTextColor>

## List Section

ແສດງຮາຍການຂ້ອງບູລຈຳນວນນັກໄດ້ຢ່າງນີ້

ປະສິກີນພາວັນຍ <FlatList> ຊຶ່ງເໝາະສໍາຮັບ

ຂ້ອງມູນຄົກທີ່ຕ້ອງການການຈັດເຮືອງ

## Button Section

ສ້າງປູ່ນທີ່ສໍາາລັກໂດຍໃຊ້

<TouchableOpacity> ຊຶ່ງຊ່ວຍໃຫ້ສໍາາລັກປັບແລ້ວປູ່ນໄດ້ຢ່າງສະແດງ

## Loading Section

ແສດງຕັ້ງບັງຫຼັກໂທລດ (loading spinner) ດ້ວຍ

<ActivityIndicator> ຂະໜາກີ່ເພື່ອກໍາລັງກຳທຳນັບ

## Scrollable Content

ຈັດການເນື້ອທາງທີ່ກ່າວເກີນທັງນັດໃຫ້ສໍາາລັກເລື່ອນດີ

ໄດ້ດ້ວຍ <ScrollView>

# StyleSheet: หัวใจของการจัดสไตร์ใน React Native

ใน React Native, การจัดสไตร์จะทำคล้ายกับ CSS บนเว็บ แต่ใช้ JavaScript objects ผ่าน `StyleSheet.create()` เพื่อสร้างสไตร์ชุด สไตร์เหล่านี้สามารถนำไปใช้กับ Core Components ได้อย่างง่ายดาย



## กำหนดสไตร์ใน JavaScript

สไตร์จะถูกเขียนเป็น JavaScript object โดยมีคุณสมบัติที่คล้ายคลึงกับ CSS (เช่น `backgroundColor`, `fontSize`) และใช้รูปแบบ CamelCase



## ความเข้ากันได้ข้ามแพลตฟอร์ม

React Native จะจัดการการแสดงผลสไตร์ให้ถูกต้องกับ iOS และ Android รวมถึงการแปลงค่าบางอย่างโดยอัตโนมัติ



## เงาและมิติ

สามารถสร้างเงา (`shadow` สำหรับ iOS และ `elevation` สำหรับ Android) เพื่อเพิ่มมิติและความลึกให้กับองค์ประกอบต่างๆ



## การจัดวางด้วย Flexbox

ใช้ `flexbox` เป็นหลักในการจัดวางองค์ประกอบต่างๆ ทำให้การออกแบบ UI มีความยืดหยุ่นและตอบสนองได้ดี

การใช้ `StyleSheet.create()` ช่วยให้โค้ดเป็นระเบียบ อ่านง่าย และมีประสิทธิภาพในการจัดการหน่วยความจำมากขึ้น

# ຕັວອຢ່າງໂຄດ: ການຈັດສໂຕຣດ້ວຍ StyleSheet

ໄດ້ດໍານລ່າງນີ້ແສດງການໃຫ້ຈັນ `StyleSheet.create()` ເພື່ອກຳທັດສໂຕຣໃຫ້ກັບອົງຄປະກອບຕ່າງໆ ໃນແອປພລິເກັນ React Native ຄຸນຈະເກີບວ່າສໂຕຣແຕ່ລະຊຸດຄູກສ້າງເປັນ JavaScript object ແລະນຳໄປໃຫ້ກັບແຕ່ລະ `Component` ໄດ້ຢ່າງໄວ

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: "#f8f9fa",
  },
  header: {
    backgroundColor: "#007AFF",
    paddingVertical: 20,
    paddingHorizontal: 16,
    alignItems: "center",
    shadowColor: "#000",
    shadowOffset: {
      width: 0,
      height: 2,
    },
    shadowOpacity: 0.25,
    shadowRadius: 3.84,
    elevation: 5,
  },
  title: {
    color: "#fff",
    fontSize: 24,
    fontWeight: "bold",
  },
  content: {
    flex: 1,
    padding: 16,
  },
  imageContainer: {
    alignItems: "center",
    marginBottom: 24,
  },
  image: {
    width: 120,
    height: 120,
    borderRadius: 60,
    shadowColor: "#000",
    shadowOffset: {
      width: 0,
      height: 4,
    },
    shadowOpacity: 0.3,
    shadowRadius: 4.65,
    elevation: 8,
  },
  inputContainer: {
    marginBottom: 24,
  },
  buttonContainer: {
    marginBottom: 24,
    width: '100%',
  },
  customButton: {
    backgroundColor: '#007AFF',
    paddingVertical: 16,
    paddingHorizontal: 24,
    borderRadius: 8,
    alignItems: 'center',
    shadowColor: "#000",
    shadowOffset: {
      width: 0,
      height: 2,
    },
    shadowOpacity: 0.25,
    shadowRadius: 3.84,
    elevation: 5,
  },
  buttonText: {
    color: "#fff",
    fontSize: 18,
    fontWeight: 'bold',
  },
  textInput: {
    backgroundColor: "#fff",
    borderWidth: 1,
    borderColor: "#ddd",
    borderRadius: 8,
    paddingHorizontal: 16,
    paddingVertical: 12,
    fontSize: 16,
    shadowColor: "#000",
    shadowOffset: {
      width: 0,
      height: 1,
    },
    shadowOpacity: 0.1,
    shadowRadius: 2,
    elevation: 2,
  },
  loadingContainer: {
    alignItems: "center",
    marginBottom: 24,
  },
  loadingText: {
    marginTop: 8,
    color: "#666",
    fontSize: 14,
  },
  scrollContainer: {
    marginBottom: 24,
  },
  sectionTitle: {
    fontSize: 18,
    fontWeight: "bold",
    color: "#333",
    marginBottom: 12,
  },
  scrollView: {
    backgroundColor: "#fff",
    borderRadius: 8,
    padding: 16,
    maxHeight: 200,
    shadowColor: "#000",
    shadowOffset: {
      width: 0,
      height: 2,
    },
    shadowOpacity: 0.1,
    shadowRadius: 3.84,
    elevation: 5,
  },
  scrollText: {
    fontSize: 16,
    color: "#333",
    marginBottom: 8,
    paddingVertical: 4,
  },
  listContainer: {
    flex: 1,
  },
  flatList: {
    backgroundColor: "#fff",
    borderRadius: 8,
    shadowColor: "#000",
    shadowOffset: {
      width: 0,
      height: 2,
    },
    shadowOpacity: 0.1,
    shadowRadius: 3.84,
    elevation: 5,
  },
  listItem: {
    paddingVertical: 12,
    paddingHorizontal: 16,
    borderBottomWidth: 1,
    borderBottomColor: "#f0f0f0",
  },
  listText: {
    fontSize: 16,
    color: "#333",
  },
});
```

ຈາກໂຄດຕັວອຢ່າງນີ້ ເຮັດວຽກເກີບວ່າສຳເນົາມາດເກີບການໃຫ້ຈັນຄຸນສັບຕັດເລັກຖ.

## ຄຸນສັບຕັດພື້ນສໂານ

ກຳທັດສັ່ນຫຼັງ (`backgroundColor`), ສັ້ນຄວາມ (`color`), ພາດຕັວອັກຂະໜາດ (`fontSize`) ແລະການວັບຮະຍະ (`padding, margin`) ເພື່ອຈັດວາງອົງຄປະກອບ

## Flexbox ເພື່ອການຈັດວາງ

ໃຊ້ `flex: 1` ສໍາຮັບການຂາຍເຕີບພື້ນຖື ແລະ `alignItems: "center"` ໃນ `header` ຮັບ `imageContainer` ເພື່ອຈັດອົງຄປະກອບໃຫ້ຢູ່ກົ່ງລາງ

## ການເພີ່ມມືດັດວຍເງາ

ສ້າງເງົາ (`shadow` ສໍາເຫັນ iOS ແລະ `elevation` ສໍາເຫັນ Android) ໃນລ່ວມ `header, image, customButton` ແລະເຊັ່ນ ເພື່ອໃກ້ດູມເປີຕິ

## ສໂຕຣດ້ວຍ Component

ເກີບການໃຫ້ຈັນ `borderRadius` ເພື່ອສ້າງງວດກອນໃນ `image, borderWidth` ໃນ `textInput` ແລະການກໍາທັດ `width: '100%'` ສໍາຮັບປຸປັກໃຫ້ເຕີບພື້ນຖື

# Props คืออะไร?

Props คือข้อมูลที่ถูกส่งผ่านจากคอมโพเนนต์หนึ่งไปยังอีกคอมโพเนนต์หนึ่ง โดย Props บักจะถูกส่งจากคอมโพเนนต์ที่เป็น Parent (แม่) ไปยังคอมโพเนนต์ที่เป็น Child (ลูก) เพื่อให้คอมโพเนนต์ลูกสามารถแสดงผลข้อมูลที่ได้รับจากคอมโพเนนต์แม่

คุณสมบัติของ Props:

- Props เปรียบเสมือน พารามิเตอร์ ที่ถูกส่งไปยังฟังก์ชัน
- Props ไม่สามารถเปลี่ยนแปลงได้จากคอมโพเนนต์ลูก และคอมโพเนนต์ลูกมีหน้าที่ในการแสดงข้อมูลที่ได้รับจาก Props เท่านั้น **เปรียบเทียบ: Props เมื่อตอน "คุณสมบัติที่ติดตัวมาตั้งแต่เกิด" ของ Component นั้นๆ**

การทำงาน:

- ข้อมูลไหลจาก บนลงล่าง (แม่ -> ลูก) เท่านั้น (One-way data flow)
- Component ลูก ไม่สามารถแก้ไข props ที่ได้รับมาได้ (Immutable)

ตัวอย่าง:

```
// Component ลูก: Greeting.js
function Greeting(props) {
  return สวัสดี, {props.name}!;
}
```

```
// Component แม่: App.js
export default function App() {
  return (
    );
}
```

ในตัวอย่างนี้ App ส่ง name (ซึ่งเป็น props) ไปให้ Greeting

# ຕົວຢ່າງການໃຊ້ຈາບ Props

ໃນການພັນນາ React Native, ເຮົາໃຊ້ Props ເພື່ອສ່ວນຂ້ອມູລຈາກຄອນໂພເບນຕີແມ່ (Parent Component) ໄປຢັງຄອນໂພເບນຕີລຸກ (Child Component) ເພື່ອໃຫ້ຄອນໂພເບນຕີລຸກສາມາດເປົ້າຂ້ອມູລນັ້ນໄປແສດງຜລກຮອປະມວລຜລຕ່ວໄດ້

## ແບວຄົດຫັກໃນການສ່ວນຂ້ອມູລດ້ວຍ Props

- ການສ່ວນຕົວຂ້ອມູລ: Props ຕົວວິທີກາຮັກໃນການສ່ວນຂ້ອມູລຈາກຄອນໂພເບນຕີແມ່ (Parent) ໄປຢັງຄອນໂພເບນຕີລຸກ (Child)
- ກາຣໄຫລວຂອງຂ້ອມູລກີສກາກເດືອຍ: ຂ້ອມູລຈະໄກລຈາກບໍລິສັດລ່າງເກົ່ານັ້ນ (One-way Data Flow) ທີ່ໜ້າຍຄວາມວ່າຂ້ອມູລຈະຖຸກສ່ວນຈາກຄອນໂພເບນຕີແມ່ໄປຢັງຄອນໂພເບນຕີລຸກເສນວ
- ໄປສາມາດແກ້ໄຂໄດ້: ຄອນໂພເບນຕີລຸກໄມ່ສາມາດແກ້ໄຂຕ່າງອອງ Props ຖໍ່ໄດ້ຮັບມາໄດ້ (Immutable) ມີບັນກີ່ເພີຍງແກ່ນຳຂ້ອມູລນັ້ນໄປໃຫ້ຮຽວແສດງຜລກເກົ່ານັ້ນ

ກາພດ້ານລ່າງແສດງໃຫ້ເກີບຄືການສ່ວນຜ່ານຂ້ອມູລຜ່ານ Props ຈາກຄອນໂພເບນຕີຫັກໄປຢັງຄອນໂພເບນຕີຍ່ອຍ

## ໂຄດຕົວຢ່າງການໃຊ້ຈາບ Props

ຕົວຢ່າງນີ້ຈະແສດງຄອນໂພເບນຕີ UserProfile (ຄອນໂພເບນຕີລຸກ) ກີ່ຮັບຄໍາ name ແລະ email ເປັນ Props ຈາກຄອນໂພເບນຕີ App (ຄອນໂພເບນຕີແມ່)

```
// UserProfile.js (ຄອນໂພເບນຕີລຸກ)
import React from 'react';
import { View, Text, StyleSheet } from 'react-native';

const UserProfile = (props) => {
  return (
    <View style={styles.card}>
      <Text style={styles.name}>ຊື່: {props.name}</Text>
      <Text style={styles.email}>ອີເມວ: {props.email}</Text>
    </View>
  );
};

const styles = StyleSheet.create({
  card: {
    backgroundColor: '#fff',
    padding: 20,
    borderRadius: 10,
    shadowColor: '#000',
    shadowOffset: { width: 0, height: 2 },
    shadowOpacity: 0.1,
    shadowRadius: 5,
    elevation: 3,
    marginBottom: 20,
  },
  name: {
    fontSize: 20,
    fontWeight: 'bold',
    marginBottom: 5,
  },
  email: {
    fontSize: 16,
    color: '#666',
  },
});

export default UserProfile;

// App.js (ຄອນໂພເບນຕີແມ່)
import React from 'react';
import { View, StyleSheet } from 'react-native';
import UserProfile from './UserProfile';

export default function App() {
  return (
    <View style={styles.container}>
      <UserProfile name="ສົມຈະ ໄຈດີ" email="somchai@example.com" />
      <UserProfile name="ມາລີ ພຶສຸພົ" email="malee@example.com" />
    </View>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
    padding: 20,
    backgroundColor: '#f0f0f0',
  },
});
```

# State คืออะไร?

State คือข้อมูลที่อยู่ภายในคอมโพเนนต์และสามารถเปลี่ยนแปลงได้เมื่อผู้ใช้โต้ตอบกับแอป หรือเมื่อมีการเปลี่ยนแปลงสถานะบางอย่างในคอมโพเนนต์ ตัวอย่างเช่น การเปลี่ยนแปลงข้อมูลฟอร์ม การกดปุ่ม หรือการอัปเดตข้อมูลที่ดึงมาจาก API

การใช้งาน State ใน React Native

- ใน React Native คอมโพเนนต์ที่ใช้ State จะเป็นคอมโพเนนต์ที่มีสถานะเป็นคอมโพเนนต์แบบ Functional Component โดยใช้ Hook ที่ชื่อว่า useState ซึ่งเป็น Hook พื้นฐานสำหรับจัดการ State

เรียบเทียบ: State เมื่อเปรียบ "ข้อมูลบนไวท์บอร์ด" ของ Component นั้นๆ เมื่อผู้ใช้ทำอะไรบางอย่าง (เช่น กดปุ่ม) Component ก็จะลบข้อมูลเก่าบนไวท์บอร์ด และเขียนข้อมูลใหม่ลงไป

การทำงาน:

- เป็นข้อมูลส่วนตัว (private) ของ Component
- เมื่อ State เปลี่ยนแปลง React จะ "วัด Component นั้นใหม่" (re-render) โดยอัตโนมัติ
- เราใช้ **useState Hook** ในการสร้างและอัปเดต State

```
function Counter() {  
  const [count, setCount] = useState(0);
```

```
  return (
```

จำนวน: {count}

```
    setCount(count + 1)} /> ); }
```

# การจัดการข้อมูลด้วย State และ Props

ในการสร้างแอป React Native การเข้าใจวิธีจัดการข้อมูลสำคัญมาก **State** และ **Props** ทำงานร่วมกันเพื่อให้ข้อมูลไหลเวียนได้ดีใน Component ของแอป

เมื่อข้อมูลใน **State** ของ Parent Component เปลี่ยนไป React จะทำการ "re-render" Component นั้นใหม่ และถ้าข้อมูลนั้นถูกส่งต่อเป็น **Props** ไปยัง Child Component, Child Component ก็จะ re-render ตามไปด้วย เพื่อแสดงข้อมูลล่าสุด

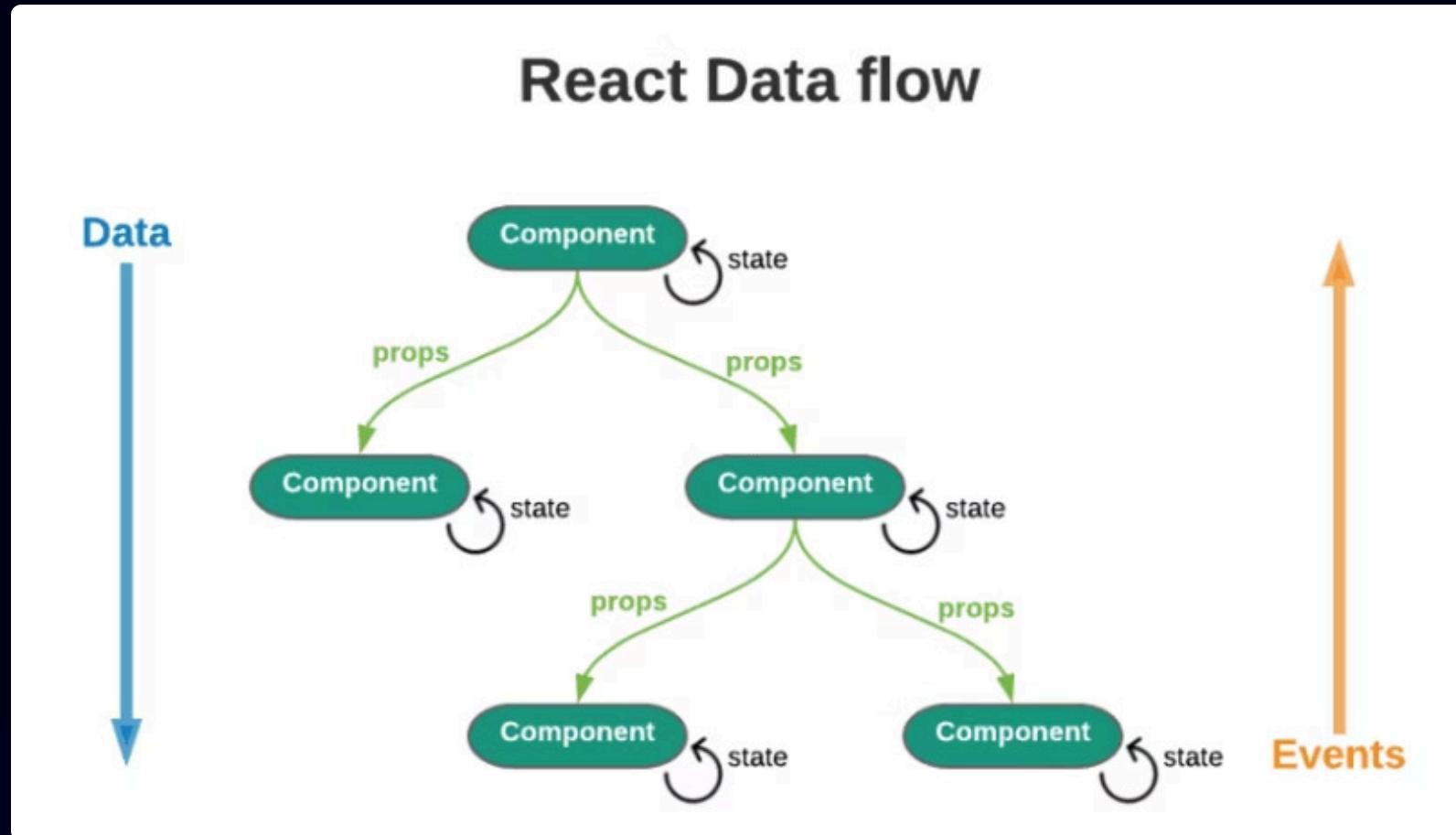
นี่คือหลักการของ "**One-way Data Flow**" ใน React Native: ข้อมูลจะไหลจาก Component แม่ (Parent) ไปยัง Component ลูก (Child) เท่านั้น Component ลูกไม่สามารถแก้ไข Props ที่ได้รับมาได้โดยตรง แต่สามารถสื่อสารกลับไปยัง Component แม่ได้ผ่านฟังก์ชันที่ถูกส่งมาเป็น Props

## How React Updates State

```
const [val, setVal] = useState(0);  
  
setVal(1);  
val; // => ???
```

การเข้าใจหลักการนี้จะช่วยให้คุณออกแบบแอปพลิเคชันที่มีการโต้ตอบกับโลกหลายและมีประสิทธิภาพได้อย่างถูกต้อง

# การจัดการข้อมูลด้วย State และ Props



# เปรียบเทียบ Props vs. State

คุณสมบัติ	Props	State
ที่มาของข้อมูล	ส่งมาจาก Component แม่	จัดการภายใน Component เอง
การเปลี่ยนแปลง	เปลี่ยนแปลงไม่ได้ (Immutable)	เปลี่ยนแปลงได้ (Mutable)
การไหลของข้อมูล	ทางเดียว (บบ -> ล่าง)	อยู่ภายใน Component
วัตถุประสงค์	ใช้สำหรับกำหนดค่าจากภายนอก	ใช้สำหรับจัดจำส่วนที่เปลี่ยนไป
เปรียบเทียบ	คุณสมบัติติดตัว	ข้อมูลบนໄວก์บอร์ด

# พื้นฐาน Array Functions

Array Functions หรือที่เรียกว่า Higher-Order Functions ใน JavaScript เป็นเครื่องมือสำคัญที่ช่วยให้เราจัดการกับข้อมูลในอาร์เรย์ได้ง่ายขึ้น ทำให้โค้ดของเราสั้นลง อ่านง่าย และทำงานได้ดีขึ้น



## .map()

ใช้สำหรับสร้างอาร์เรย์ใหม่ โดยนำแต่ละอย่างในอาร์เรย์เดิมไป "เปลี่ยนรูป" ตามที่เราสั่ง แล้วได้ผลลัพธ์เป็นอาร์เรย์ใหม่ ส่วนอาร์เรย์เดิมจะไม่ถูกเปลี่ยน

```
const numbers = [1, 2, 3];
const doubled = numbers.map(num => num * 2);
// doubled: [2, 4, 6]
```



## .filter()

ใช้สำหรับสร้างอาร์เรย์ใหม่ โดยเลือกเฉพาะข้อมูลที่ "ตรงตามเงื่อนไข" ที่เรากำหนดไว้ เช่นเดียวกับการกรองข้อมูลให้อาร์เรย์ใหม่มีเฉพาะสิ่งที่เราต้องการ

```
const ages = [10, 20, 30, 15];
const adults = ages.filter(age => age >= 18);
// adults: [20, 30]
```



## .reduce()

ใช้สำหรับรวมข้อมูลทั้งหมดในอาร์เรย์ให้เหลือ "ค่าเดียว" โดยจะนำแต่ละอย่างไปบวก ลบ คูณ หาร หรือทำอะไรบางอย่างกับค่ารวมที่สะสมมาเรื่อยๆ จนได้ผลลัพธ์สุดท้าย

```
const numbers = [1, 2, 3, 4];
const sum = numbers.reduce((acc, num) => acc + num, 0);
// sum: 10
```

การใช้ Array Functions ช่วยให้โค้ดของคุณทำงานได้อย่างมีประสิทธิภาพและเข้าใจง่ายขึ้น โดยเฉพาะเมื่อคุณต้องจัดการกับข้อมูลจำนวนมาก

# พื้นฐาน Array Functions

```
['a', 'b'].concat(['c']) //['a','b','c']
['a', 'b'].join(~) //a~b
['a', 'b', 'c'].slice(1) //['b', 'c' ]
['a', 'b', 'b'].indexOf('b') // 1
['a', 'b', 'b'].lastIndexOf('b') //2
```

```
['a', 'b', 'c'].forEach(x => console.log(x))
[1,2,3].every(x => x < 10 )//true
[1,2,3].some(x => x < 2 )//true
[1,2,3].filter(x => x < 2 )//[1]
```

## ARRAY CHEATSHEET

```
[1, 2, 3].map(x => x * 2 )//[2, 4, 6]
[1, 2, 3].reduce((x,y) => x * y)//6
[2, 15,3].sort()//[ 15, 2, 3 ] 😊
[1, 2, 3].reverse()//[ 3, 2, 1 ]
[1, 2, 3].length//3
```

```
const arr = [1, 2, 3]
const x=arr.shift()//arr=[ 2, 3 ],x=1
const x=arr.unshift(9)//arr=[ 9,1,2,3 ],x=4
const x=arr.pop()//arr=[ 1, 2 ],x=3
const x=arr.push(5)//arr=[1,2,3,5],x=4
```

```
const arr=['a','b','c','d'];const mod = arr.splice(1,2,'z');//arr=['a','z','d'],mod=['b','c']
```



# React Hooks คืออะไร?

**Hooks** คือ พัฟ์ชันพิเศษ ที่ช่วยให้เราสามารถใช้ความสามารถต่างๆ ของ React ได้โดยตรงใน Functional Component

ทำไมต้องมี Hooks? เนื่องจาก ถ้าต้องการใช้ State หรือ Lifecycle ต้องเขียน Component แบบ Class ซึ่งซับซ้อนกว่า และ Hooks ทำให้ Functional Component (ที่เป็นแค่พัฟ์ชันธรรมดา) มีความสามารถเทียบเท่า Class Component แต่เขียนง่ายและกระชับกว่ามาก

เฟรียบเทียบ: Hooks เมื่อตอน "กล่องเครื่องมือ" ที่เราสามารถหยิบเครื่องมือต่างๆ มาใช้ในพัฟ์ชันของเราได้

กฎการใช้ Hooks:

- ต้องเรียกใช้ Hooks ที่ระดับบนสุดของ Component เท่านั้น (ห้ามเรียกใน loop, condition, หรือ nested function)
- ต้องเรียกใช้ Hooks จาก React Functional Component เท่านั้น

# React Hooks: ฉบับสรุป

## Hooks คืออะไร?

คือฟังก์ชันพิเศษที่ช่วยให้ Functional Component สามารถใช้ความสามารถต่างๆ ของ React ได้ เช่น state, lifecycle, context, refs และการจูนประสิทธิภาพ

## เปรียบเทียบ: "กล่องเครื่องมือ"

Hooks เมื่อตอน "กล่องเครื่องมือ" ที่เราสามารถหยิบเครื่องมือต่างๆ มาใช้ในฟังก์ชันของเราได้

- หยิบใช้เฉพาะที่จำเป็น

## ทำไมต้องมี Hooks?

เดิมต้องใช้ Class Component ที่ซับซ้อนกว่าเพื่อจัดการ state/lifecycle และ Hooks ทำให้การเขียนโค้ดแบบฟังก์ชันง่าย กระชับ และสามารถแยก Logic กลับมาใช้ซ้ำได้ (**Custom Hooks**)

## กฎการใช้ Hooks

- ต้องเรียกใช้ Hooks ที่ระดับบนสุดของคอมโพเนนต์/คัสตอมอւคเก็บนั้น (ห้ามใน loop, condition, หรือ nested function)
- เรียกใช้ Hooks ได้เฉพาะ **"React functions"** เท่านั้น (Functional Component หรือ Custom Hook)
- ชื่อ Custom Hook ต้องขึ้นต้นด้วย `use` เสมอ
- ใส่ dependencies ของ `useEffect/useMemo/useCallback` ให้ครบ (เปิด `eslint-plugin-react-hooks` ช่วยเตือน)

# Hook กี่ต้องรู้จัก: useState

หน้ากี: เพิ่ม "หน่วยความจำ" หรือ State เข้าไปใน Component

การเรียกใช้งาน: `const [stateValue, setStateValue] = useState(initialValue);`

สิ่งที่ได้กลับมา:

- `stateValue`: ตัวแปรที่เก็บค่า state ปัจจุบัน
- `setStateValue`: พังก์ชันสำหรับอัปเดตค่า state นั้น
- `initialValue`: ค่าเริ่มต้นของ state

ตัวอย่าง (Counter):

```
import React, { useState } from 'react';
import { View, Text, Button } from 'react-native';

export default function Counter() {
  const [count, setCount] = useState(0); // ค่าเริ่มต้นคือ 0
```

return (

คุณกดไปแล้ว: {count} ครั้ง

`setCount(count + 1)} // เรียก setCount เพื่ออัปเดตค่า /> ); }`

# Hook กี่ต้องรู้จัก: useEffect

หน้าที่: ใช้สำหรับจัดการ "Side Effects" ซึ่งก็คือโค้ดใดๆ กี่ทำงานนอกเหนือจากการคำนวณและคืนค่า JSX

เปรียบเทียบ: useEffect เมื่อൺการบอก React ว่า "หลังจากที่วิ่งหน้าจอเสร็จแล้ว ช่วยทำงาน X ให้หน่อย"

การเรียกใช้งาน: `useEffect(() => { /* โค้ดที่ต้องการให้ทำงาน */ }, [dependencies]);`

ส่วนประกอบ:

- **Effect Function:** พังก์ชันแรก คือโค้ดที่เราต้องการให้ทำงาน
- **Dependency Array:** `[dependencies]` คือ "รายการเฝ้าดู" ที่บอก `useEffect` ว่าควรจะทำงานซ้ำเมื่อไหร่
  - `[] (array ว่าง):` ทำงานแค่ ครั้งเดียว หลังจาก Component render ครั้งแรก
  - `[name, age] (มีตัวแปร):` ทำงานครั้งแรก และจะทำงานซ้ำ ทุกครั้งที่ค่า `name` หรือ `age` เปลี่ยนแปลง
  - `(ไม่ใส่ array):` ทำงานซ้ำ ทุกครั้งที่มีการ **re-render** (ควรหลีกเลี่ยง)

# ตัวอย่าง: การใช้ useState + useEffect

ตัวอย่างโค้ดนี้สาธิตการใช้งาน useState และ useEffect ร่วมกันเพื่อจัดการสถานะและโหลดข้อมูลจาก API เมื่อ Component ถูก Render ครั้งแรก

```
import React, { useEffect, useState } from 'react'

export default function QuoteCard() {
  const [quote, setQuote] = useState('')
  const [loading, setLoading] = useState(false)

  useEffect(() => {
    let cancelled = false
    const load = async () => {
      setLoading(true)
      try {
        const res = await fetch('https://api.quotable.io/random')
        const data = await res.json()
        if (!cancelled) setQuote(data.content)
      } finally {
        if (!cancelled) setLoading(false)
      }
    }
    load()
    return () => { cancelled = true } // cleanup ป้องกัน setState หลัง unmount
  }, []) // รันครั้งเดียวเมื่อ mount

  return <div>{loading ? 'กำลังโหลด...' : quote}</div>
}
```

ในโค้ดนี้ useState ถูกใช้เพื่อเก็บคำคม (quote) และสถานะการโหลด (loading) ส่วน useEffect จะทำงานเพียงครั้งเดียวเมื่อ Component โหลดเสร็จ (สังเกตจาก  ก้ายฟังก์ชัน) เพื่อดึงข้อมูลจาก API

ฟังก์ชัน cleanup (return () => { cancelled = true }) ช่วยป้องกันข้อผิดพลาดจากการพยายามอัปเดตสถานะบน Component ที่ถูก unmount ไปแล้ว

# กฎของ Hooks: เรียกใช้ที่ระดับบนสุด

Hooks ต้องถูกเรียกใช้ที่ ระดับบนสุดของ **Functional Component** เท่านั้น หรือ Custom Hook คุณไม่สามารถเรียก Hooks ได้ภายในเงื่อนไข (if), ลูป (for), หรือฟังก์ชันที่ซ้อนกัน



**✗ ผิด: เรียก useEffect ในเงื่อนไข**

```
// ผิด: เรียก useEffect ใน if
function Bad() {
  const online = true
  if (online) {
    useEffect(() => { console.log('online') }, [])
  }
  return null
}
```

การเรียก `useEffect` ภายในบล็อก `if` ทำให้ React ไม่สามารถรับประคันลำดับการเรียก Hooks กับสอดคล้องกับในการ Render แต่ละครั้ง ซึ่งอาจนำไปสู่ข้อผิดพลาดที่คาดไม่ถึงได้



**✓ ถูก: จัดการเงื่อนไขภายใน Effect**

```
// ถูก: แยกเงื่อนไขออกมาในตัว effect
function Good() {
  const online = true
  useEffect(() => {
    if (online) console.log('online')
  }, [online])
  return null
}
```

ในตัวอย่างที่ถูกต้อง `useEffect` ถูกเรียกที่ระดับบนสุดเสมอ โดยเงื่อนไข `(online)` ถูกย้ายไปอยู่ภายในฟังก์ชัน Effect ทำให้ React รักษาลำดับของ Hooks ได้ถูกต้อง และ Effect จะทำงานเมื่อค่า `online` เปลี่ยนแปลง

# Hook ที่ต้องรู้จัก: Custom Hook

หน้ากี่: คือฟังก์ชัน JavaScript ที่ใช้ Hooks ตัวอื่น ๆ (เช่น useState, useEffect) และมีวัตถุประสงค์เพื่อ encapsulate (ห่อหุ้ม) และนำ logic ที่มี stateful กลับมาใช้ซ้ำได้ในหลาย ๆ Component

ประโยชน์:

- นำกลับมาใช้ซ้ำได้: ช่วยให้โค้ดสะอาดขึ้นและไม่ต้องเขียนโค้ดซ้ำ ๆ
- แยก Logic ออกจาก UI: ทำให้ Component มีหน้าที่แค่การ Render UI เท่านั้น
- อ่านง่ายและเข้าใจง่าย: ช่วยให้โค้ดเป็นระเบียบมากขึ้น

กฎการตั้งชื่อ: Custom Hook จะต้องขึ้นต้นด้วยคำว่า use เช่น useOnline (เช่น useState, useAuth)

ตัวอย่าง: useOnline Custom Hook

```
import { useEffect, useState } from 'react'

function useOnline() {
  const [online, setOnline] = useState(false)
  useEffect(() => {
    // ตย. subscribe สถานะเครือข่าย (ยกตัวอย่าง)
    const id = setInterval(() => setOnline(o => !o), 3000)
    return () => clearInterval(id)
  }, [])
  return online
}

function Status() {
  const online = useOnline()
  return <span>{online ? 'Online' : 'Offline'}</span>
}
```

ในตัวอย่างนี้ useOnline จะจัดการสถานะออนไลน์/ออฟไลน์ โดยใช้ useState และ useEffect และ Status Component สามารถเรียกใช้ useOnline เพื่อเข้าถึงสถานะดังกล่าวได้โดยตรง

# สรุปการรวม React Hooks

ตอนนี้เรารู้จัก Hooks แต่ละตัวแล้ว มาดูกาพร้อมของ Hooks สำคัญๆ กี่ใช้บ่อยๆ กันครับ

## A) จัดการสถานะ (State Management)

### useState

สำหรับ: เก็บข้อมูลที่เปลี่ยนใน Component

ตัวอย่าง: ช่องกรอกข้อมูล, ปุ่มเปิด/ปิด, ตัวนับ

ข้อแนะนำ: เหมาะกับข้อมูลที่ไม่ซับซ้อน อัปเดตโดยการแก้ไขค่าเดิม

### useReducer

สำหรับ: จัดการข้อมูลที่ซับซ้อน หรือมีการเปลี่ยนตาม "กฎ" หลายอย่าง (ส่งคำสั่ง)

ตัวอย่าง: จัดการรายการ Todo (เพิ่ม/แก้ไข/ลบ), ตະกร้าสิบค้า

ข้อแนะนำ: เหมาะกับข้อมูลซับซ้อน ช่วยให้จัดการได้่ายกว่า useState

## B) เอฟเฟกต์/วงศ์ชีวิต (Side Effects)

### useEffect

สำหรับ: ใช้สำหรับงานที่อยู่นอกการแสดงผล (Side Effects) เช่น ดึงข้อมูลจาก API, ตั้งเวลา, สมัคร/ยกเลิก Events

ตัวอย่าง: โหลดข้อมูลเมื่อ Component แสดงครั้งแรก, บันทึกข้อมูล, ตั้งค่า Subscription

ข้อแนะนำ: ต้องกำหนด dependencies array ให้ถูก; อย่าลืมคืนฟังก์ชัน cleanup ถ้ามีการสมัคร Events หรือตั้งเวลา

### useLayoutEffect

สำหรับ: ทำงานพร้อมกับการเปลี่ยนหน้าจอ (DOM) ใช้ปรับแก้ Layout หรืออ่านค่าจาก DOM

ตัวอย่าง: คำบวนคำแหง Scroll, ปรับขนาด Element ก่อนผู้ใช้จะเห็น

ข้อแนะนำ: ใช้เก่าก็จำเป็น เพราะอาจทำให้หน้าจอโหลดช้าลงถ้าทำงานหนักไป

### useInsertionEffect

สำหรับ: ใช้แทรก Style ก่อน React จะแก้ DOM จริงๆ (ส่วนใหญ่ใช้ใน Library CSS-in-JS)

ตัวอย่าง: Library ที่จัดการ Style แบบรันไทม์

ข้อแนะนำ: เป็น Hook ขั้นสูง ไม่ค่อยได้ใช้โดยตรงในการพัฒนาแอป ก็จะไป

# สรุปภาษา React Hooks (ต่อ)

## C) อ้างอิง/สั่งงานเชิงคำสั่ง (Refs)

1

### useRef

ใช้ก็องอะไร: ใช้เก็บค่าต่างๆ ที่ไม่ต้องการให้ทำให้ Component โหลดใหม่ หรือใช้อ้างอิงส่วนประกอบบนหน้าจอ (เช่น ช่องกรอกข้อมูล)

สถานการณ์จริง: เช่น ให้ช่องกรอกข้อมูลแสดงเคอร์เซอร์อัตโนมัติ หรือเก็บค่าตัวเลขที่นับไปเรื่อยๆ โดยไม่แสดงผลบนหน้าจอ

ข้อควรระวัง/ทิปส์: เมื่อค่าใน .current ของ Ref เปลี่ยน หน้าจอจะไม่แสดงผลใหม่

2

### useImperativeHandle

ใช้ก็องอะไร: ช่วยให้ Component แม่สั่งให้ Component ลูกทำงานบางอย่างได้โดยตรง (ต้องใช้คู่กับ forwardRef)

สถานการณ์จริง: เช่น สั่งให้ Component ลูกที่เป็นช่องกรอกข้อมูลแสดงเคอร์เซอร์ หรือล้างข้อมูล

ข้อควรระวัง/ทิปส์: ควรใช้เมื่อจำเป็นเท่านั้น เพื่อไม่ให้โค้ดซับซ้อนเกินไป

## D) แชร์ข้อมูลข้ามคอมโพเนนต์ (Context)

### useContext

ใช้ก็องอะไร: ใช้สำหรับอ่านข้อมูลส่วนกลางที่แชร์กันได้กันทั่วแอป โดยไม่ต้องส่งข้อมูลผ่าน Props ลงไปทีละชั้น (แก้ปัญหา Prop Drilling)

สถานการณ์จริง: เช่น ข้อมูลร่มของแอป, ข้อมูลผู้ใช้งาน, ตะกร้าสินค้า หรือรายการสิ่งที่ต้องกำ

ข้อควรระวัง/ทิปส์: ระวังเรื่องการโหลดหน้าจอใหม่ก็ังหนด หากข้อมูลใน Context เปลี่ยนบ่อยๆ ถ้าเป็นไปได้ควรแบ่ง Context ย่อยๆ

# สรุปการรวม React Hooks (ต่อ)

## E) เพิ่มประสิทธิภาพ (Memoization)

### useMemo

ใช้ก็າວໂໄສ: ຊ່ວຍ "ຈໍາ" ພຸລັບພົນຂອງການຄໍານວນທີ່ສັບສ້ອນ ເພື່ອໄປເຕັມຄໍານວນໃໝ່ທຸກຄົງທີ່ Component Render

ສານກາຮົງ: ເນື່ອຕ້ອງຮອງຮັບເຮັດວຽກຂ້ອງມູລເຍວະໆ ຫ້າງ ມາຍຄົ້ນ

ຂ້ອຍຮະວັງ/ກີບສີ: ໃຫ້ເນື່ອຮັບສິກວ່າແອປ້າລາງຈົງທີ່ ໂມ່ຕ້ອງໃສ່ທຸກທີ່

### useCallback

ໃຊ້ກໍາວໂໄສ: ຊ່ວຍ "ຈໍາ" ພັງກັບສັນ ເພື່ອໄປໃຫ້ React ສ້າງພັງກັບສັນໃໝ່ຫ້າງ ຕອນ Component Render ກໍາໃຫ້ Component ລູກຖີ່ຮັບພັງກັບສັນນີ້ໄປເຕັມຕ້ອງ Render ໄກມໄດຍ່ໄປຈ້າເປັນ

ສານກາຮົງ: ສ່າງພັງກັບສັນ (ເສັນ ປຸ່ມຄດ, ເປົ້າຍັນຄ່າ) ໄປໃຫ້ Component ລູກຖີ່ຄຸດ "ຈໍາ" ໄວແລ້ວ (ດ້ວຍ React.memo)

ຂ້ອຍຮະວັງ/ກີບສີ: ຕ້ອງໃສ່ຕັວແປຣກໍທີ່ເກີຍຂ້ອງກັ້ງໜັດໃນ dependencies array ໄກສະບັບ

## F) ຄວາມສາມາດ React 18 (ກໍາໃຫ້ UI ລື່ນໜຶ່ນ)

### useTransition

ໃຊ້ກໍາວໂໄສ: ໃຫ້ຈັດການການອັບເດດ UI ກໍ່ໃຫ້ເວລານາບ ໄກສັນ ແລ້ວ ອັດຕະກຳໃຫ້ຈັນໄດ້ ໄມ່ຄ້າງ

ສານກາຮົງ: ເວລາຄົດເປົ້າຍັນແກີບ ຮັບໃຫ້ຕັວກຮອງຂ້ອງມູລຈຳນວນນາກ ແອປະດູໄໂລດລື່ນໜຶ່ນ

ຂ້ອຍຮະວັງ/ກີບສີ: isPending ຈະຊ່ວຍບອກວ່າກໍາລັງໂລດອ່ຍ່ ກໍາໃຫ້ເຮົາແສດງສານະ "ກໍາລັງໂລດ" ໄດ້

### useDeferredValue

ໃຊ້ກໍາວໂໄສ: ນັ່ງເວລາການອັບເດດຄ່າບາງອ່າຍ່າງທີ່ເປົ້າຍັນມ່ວຍໆ ເພື່ອໃຫ້ React ຈັດການ UI ສ່ວນອັນກ່ອນ

ສານກາຮົງ: ຕອນຄັນຫາຂ້ອງມູລແບບພິບພົມໄປຄັນຫາໄປໃນຂ້ອງມູລຈຳນວນນາກ

ຂ້ອຍຮະວັງ/ກີບສີ: ໃຫ້ກັບຄ່າທີ່ສ່ົງຜົດຕ່ອງການຄໍານວນເຍວະໆ ຮັບໃຫ້ Component ທີ່ແສດງຜົດຊ້າ

### useId

ໃຊ້ກໍາວໂໄສ: ສ້າງ ID ຖໍ່ໄປຫ້າກັນ ເພື່ອໃຫ້ເຂົ້າມໂຍງ Label ກັບ Input ຮັບໃຫ້ມີຄວາມເຂົ້າດີເກີດໜຶ່ນ

ສານກາຮົງ: ສ້າງພົມກົມທີ່ມີ Input ລາຍລັບ ຮັບໃຫ້ກຳນົດ ID ສໍາເຫຼັດກົດສອບໃນ React Native

ຂ້ອຍຮະວັງ/ກີບສີ: ລັບໃຫ້ເປັນ key ສໍາເຫຼັດຮາຍການໃນ List ທີ່ມີການເປົ້າຍັນແປລົງບ່ອຍ

### useSyncExternalStore

ໃຊ້ກໍາວໂໄສ: ເຊື່ອນຕ່ອແລະໃຫ້ React Component ວັດທະນາຂ້ອງມູລຈາກແຫລ່ງກາຍນອກ React

ສານກາຮົງ: ເຊື່ອນກັບໄລຍະຮັດການ State ນອກ React ເຊິ່ງ Zustand, Redux ຮັບໃຫ້ Custom Store ທີ່ເຊື່ອນເອງ

ຂ້ອຍຮະວັງ/ກີບສີ: ຄວາມມີຄວາມຮູ້ເຮັດວຽກຈັດການຂ້ອງມູລແບບ Publish-Subscribe (Pub/Sub) ກ່ອນໃຫ້

# สรุปการรวม React Hooks (ต่อ)

## G) ดีบัก/ช่วยพัฒนา Custom Hooks

### useDebugValue

ใช้ก็อธิค: ช่วยให้เห็นค่าข้างใน Custom Hook ของเราได้ง่ายขึ้นในเครื่องมือสำหรับพัฒนา (DevTools)

สถานการณ์จริง: เช่น ถ้ามี Custom Hook ที่เช็คสถานะแบตเตอรี่ (`useBatteryStatus`) ก็สามารถใช้ตัวนี้แสดงสถานะแบตเตอรี่ใน DevTools ได้เลย

ข้อควรระวัง/กิปส์: ใช้แค่ตอนเดบักเท่านั้น ไม่ควรส่งผลกับการทำงานของแอป และส่วนใหญ่จะไม่ส่งผลกับเวอร์ชันที่ใช้งานจริง (Production Build)

## H) React Native/Navigation (เสริมที่ใช้บ่อย)

### useWindowDimensions (React Native)

ใช้ก็อธิค: บอกขนาดหน้าจอ (กว้าง, สูง) และการวางแผนหน้าจอ (แนวตั้ง/แนวนอน) แบบเรียลไทม์ จึงอัปเดตเองเมื่อขนาดหน้าจอเปลี่ยนไป

สถานการณ์จริง: ใช้ปรับการจัดวาง, ขนาดตัวอักษร, หรือระยะห่างต่างๆ ใน UI ให้เข้ากับขนาดหน้าจออุปกรณ์ที่ต่างกัน หรือเมื่อมีการหมุนหน้าจอ

ข้อควรระวัง/กิปส์: สะดวกกว่าการใช้ `Dimensions.addEventListener` เพราะบันจุ Re-render คอมโพenenต์ให้เองเมื่อขนาดเปลี่ยน ทำให้โค้ดกระซับลง

### useColorScheme (React Native)

ใช้ก็อธิค: ตรวจสอบว่าผู้ใช้ตั้งค่าโหมดสี (สว่าง/มืด) ไว้แบบไหนในระบบปฏิบัติการของอุปกรณ์

สถานการณ์จริง: ใช้สลับธีมของแอปให้เป็น Light Mode หรือ Dark Mode อัตโนมัติ เพื่อประสบการณ์ที่ดีขึ้น

ข้อควรระวัง/กิปส์: สามารถใช้ร่วมกับ Context API เพื่อสร้างระบบ Theme ที่ยืดหยุ่นและใช้ได้กึ่งแอป

### useFocusEffect (React Navigation)

ใช้ก็อธิค: ใช้รับโค้ดเมื่อหน้าจอ (Screen) ใน Navigation Stack ถูก "เลือก" หรือแสดงขึ้นมาอยู่บนสุด

สถานการณ์จริง: โหลดข้อมูลล่าสุดเมื่อผู้ใช้กลับมาที่หน้าจอหนึ่น (เช่น โหลดรายการใหม่), หรือเริ่มจับเวลาเฉพาะเมื่อหน้าจอนี้เปิดอยู่

ข้อควรระวัง/กิปส์: ต้องส่งฟังก์ชัน `cleanUp` คืนมาเสมอ ถ้ามีการสมัครสมาชิกหรือเริ่มจับเวลา เพื่อยกเลิกเมื่อหน้าจอกลับไปได้ฟรีสแล้ว

# สรุปการรวม React Hooks (จบ)

## กฎสำคัญของ Hooks ที่ต้องจำ

01

### เรียก Hook ก็ “บนสุด” เก่าบ้าน

อย่าเรียก Hook ในช่วง if, อุป for, หรือในฟังก์ชันซ้อนกัน เพราะจะทำให้ Hooks ทำงานผิดพลาดได้

02

### เรียกได้แค่ใน Functional Component หรือ Custom Hook เก่าบ้าน

Hooks ถูกสร้างมาให้ใช้ใน React Functional Component หรือใน Custom Hook ของคุณเอง ห้ามเรียก Hooks ในฟังก์ชัน JavaScript ก็ว่าไป

03

### ใส่ Dependency Array ให้ถูกต้อง

เวลาใช้ `useEffect`, `useMemo`, `useCallback` ต้องใส่ค่าทุกอย่างที่ Hook ใช้อ้างอิงครบถ้วนใน Array เพื่อให้ React รู้ว่าเมื่อไหร่ควรจะทำงานใหม่

## จักรวาลความสำคัญของ React Hooks

### พื้นฐานที่แข็งแกร่ง

เริ่มต้นด้วย `useState` สำหรับจัดการข้อมูลใน Component และ `useEffect` สำหรับจัดการสิ่งที่ต้องทำหลังจาก Render (Side Effects)

### การส่งข้อมูลที่ดีขึ้น

เมื่อต้องส่งข้อมูลข้ามหลาย Component ให้ใช้ `useContext` หรือ `useReducer` ร่วมกับ Context API เพื่อแก้ปัญหา "Prop Drilling"

### เพิ่มประสิทธิภาพเมื่อจำเป็น

ใช้ `useMemo` และ `useCallback` เพื่อช่วยให้แอปทำงานเร็วขึ้นในส่วนที่มีการคำนวณซับซ้อน หรือใช้กรัฟพยากรณ์มาก โดยใช้เมื่อเรื่องเห็นปัญหาด้านประสิทธิภาพ

### ใช้ได้ดีกับ React Native

Hooks เอกพากงอย่าง `useWindowDimensions`, `useColorScheme`, และ `useFocusEffect` ช่วยให้การพัฒนาแอป Native ยืดหยุ่นและตอบโจทย์มากขึ้น

### Custom Hooks: จัดระเบียบโค้ด

แยกส่วนการทำงานที่ซับซ้อนและนำกลับมาใช้ใหม่ได้ ให้เป็น Custom Hooks เพื่อให้โค้ดสะอาด อ่านง่าย และจัดการได้ง่ายขึ้นในแอปฯขนาดใหญ่



# ເລືອກໃຊ້ໄທຄູກງານ (Quick Map)

ສຽງສັບໆ ວ່າ Hook ແຕ່ລະຕັ້ງເໝາະກັບສຄານກາຮນໄດ້

## useState

ເໝາະກັບ: ຈັດກາຮ້ອມວຸດທີປ່ອຍໃນ Component

## useReducer (+ Context API)

ເໝາະກັບ: ຈັດກາຮ້ອມວຸດທີປ້ອນ ຮັບເປັນແປ່ງແປ່ງ

## useEffect

ເໝາະກັບ: ກໍາຈຳກັດກາຮ້ອມວຸດ ເພີ້ນໂລດຫຼັມວຸດ ຕັ້ງເວລາ, ຮັບເປັນແປ່ງແປ່ງ

## useRef

ເໝາະກັບ: ເຂົ້າດຶງອົງກົດຂອງຫຼັມວຸດ ຮັບເປັນແປ່ງແປ່ງ

## useMemo

ເໝາະກັບ: ຂ່າຍໃຫ້ກາຮ້ອມວຸດທີປ້ອນກໍາຈຳກັດກົດ

## useCallback

ເໝາະກັບ: ຂ່າຍໃຫ້ຝັ້ງກົດກົດສ້າງໃຫ້ປ່ອຍໃນ Component

## useLayoutEffect

ເໝາະກັບ: ປັບແຕ່ງຫຼັມວຸດທີຕ້ອງກໍາຈຳກົດກົດໃນ Browser ແລ້ວແປ່ງແປ່ງ

## useTransition/useDeferredValue

ເໝາະກັບ: ກໍາໃຫ້ຫຼັມວຸດກໍາຈຳກົດກົດໄດ້ເລີ່ມໄລ້ ແນ້ນມີກາຮ້ອມວຸດ

## useSyncExternalStore

ເໝາະກັບ: ເຊື່ອມຕ່ອງກົດກົດຫຼັມວຸດກາຍນອກ React ແລ້ວຕິດຕາມກາຮ້ອມວຸດ

## useWindowDimensions/useColorScheme (React Native)

ເໝາະກັບ: ສ້າງຫຼັມວຸດທີປ້ອນຂາດວຸດໂຄຣນີ ແລ້ວໂໂມດສີ (ເຫັນ ໂບດມືດ/ສວ່າງ) ໃນ React Native

## useFocusEffect (React Navigation)

ເໝາະກັບ: ກໍາຈຳກົດກົດຫຼັມວຸດໃຫ້ກົດເປີດໂຮກລັບນາໃຫ້ງານ (ໃນ React Navigation)

# ข้อควรระวังในการใช้ React Hooks

เพื่อให้ React Hooks ทำงานได้ดีและไม่มีปัญหา ควรทำความเข้าใจและปฏิบัติตามคำแนะนำเหล่านี้

## หลีกเลี่ยงการใช้ useEffect เพื่อ คำนวณค่าที่ได้จาก props/state

ถ้าค่าที่ต้องคำนวณมาจาก `props` หรือ `state`  
ที่เปลี่ยนไป ให้ใช้ useMemo แทน useEffect  
 เพราะจะคำนวณแค่ตอนที่ข้อมูลที่เกี่ยวข้อง  
(dependencies) เปลี่ยนเท่านั้น ทำให้โค้ดดูเด่น  
และทำงานได้มีประสิทธิภาพกว่า

## ตรวจสอบ dependencies array ให้ ครบถ้วน

เมื่อใช้ useEffect, useMemo, หรือ  
useCallback คุณต้องมั่นใจว่าใส่ค่าทุกอย่างที่  
Hook ใช้อย่างครบถ้วนใน dependencies  
array การใส่ไม่ครบอาจทำให้เกิดบีกที่หายาก  
หรือ Hook ทำงานไม่ตรงตามที่ต้องการ eslint-  
plugin-react-hooks เป็นเครื่องมือที่ดีที่จะช่วย  
เตือนเรื่องนี้ได้

## อย่าใช้ useMemo/useCallback เยอะเกินไป

Hook เหล่านี้ช่วยเพิ่มประสิทธิภาพ แต่การใช้ทุก  
ที่โดยไม่มีเหตุผลจะทำให้โค้ดซับซ้อนขึ้น และอาจ  
ทำให้แอปช้าลงได้ เพราะมีภาระในการจดจำค่า  
ต่างๆ ควรลองตรวจสอบประสิทธิภาพ (profile)  
และป้องกันก่อน เพื่อหาจุดที่ช้าลงๆ แล้วค่อย  
ใช้ Hook เหล่านี้เพื่อปรับปรุงเฉพาะจุดนั้นๆ

# Workshop: สร้าง Counter & Form

## แอป Counter

```
import React, { useState } from 'react';
import { View, Text, Button, StyleSheet } from 'react-native';

export default function App() {
  const [count, setCount] = useState(0);

  return (
    

      คุณกดไปแล้ว: {count} ครั้ง

    
  );
}

const styles = StyleSheet.create({ container: { flex: 1, justify... })
```



ลองเปลี่ยนโค้ดให้มีบุํม "ลด" ด้วย และกดลอง กดดูว่า state เปลี่ยนอย่างไร

# การจัดสไตร์ด้วย Flexbox

Flexbox คือระบบการจัดวาง Layout ที่ทรงพลัง แนวคิดหลักคือการจัดเรียงของใน "กล่อง" (View) ไม่ว่าจะในแนวตั้ง (column) หรือแนวนอน (row)

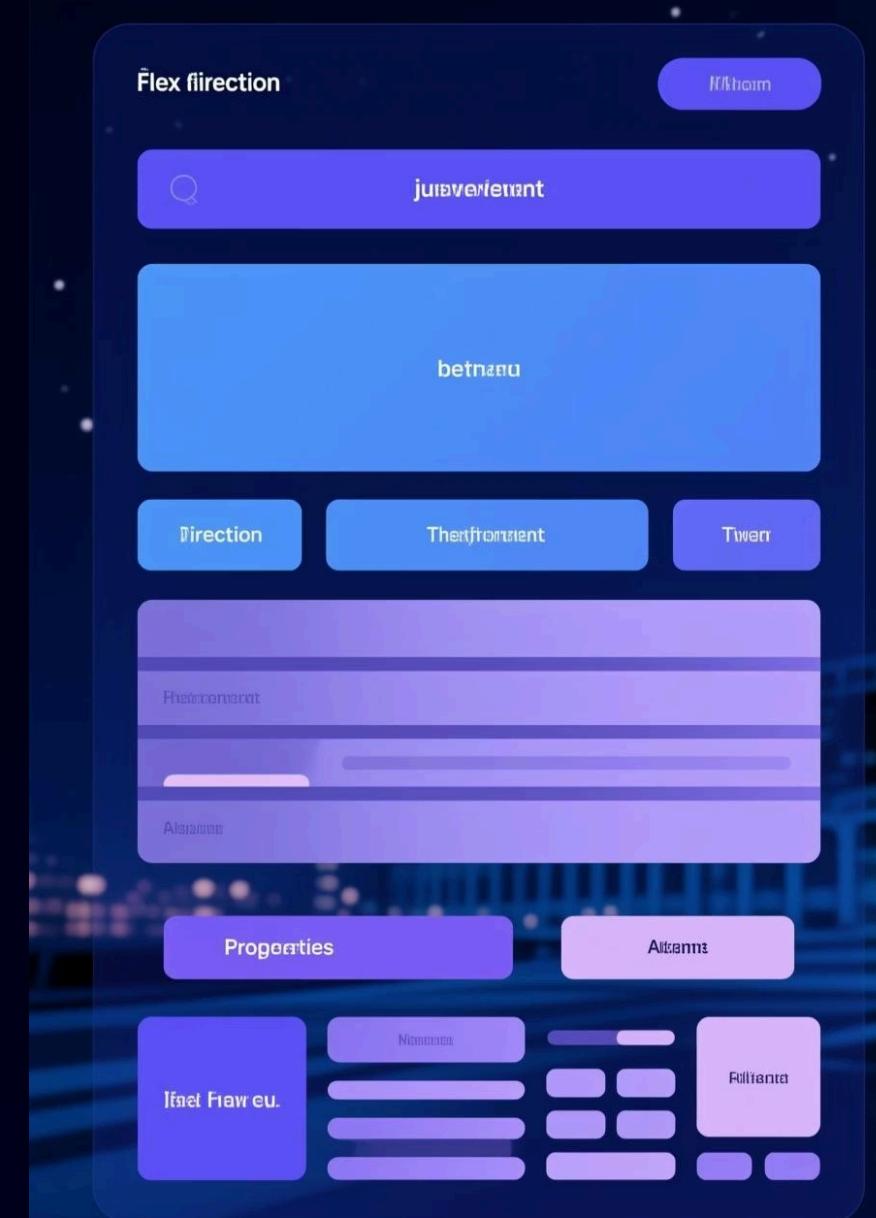
คุณสมบัติหลัก:

flex: 1 - ให้ขยายเต็มพื้นที่ที่เหลือ

flexDirection - กำหนดแนวการจัดวาง (row, column)

justifyContent - จัดตำแหน่งตามแนวแกนหลัก

alignItems - จัดตำแหน่งตามแนวแกนรอง



# UI Libraries สำหรับ React Native

คลัง UI Components ยอดนิยมที่จะช่วยให้คุณสร้างแอปได้สวยงามและรวดเร็ว

## React Native Elements ★★★★★

ติดตั้ง: npm install react-native-elements

จุดเด่น:

- ใช้งานง่าย เมื่อ用 Material UI
- มี components ครบครัน (Button, Card, Input, etc.)
- ปรับแต่งได้ง่าย
- Community ใหญ่

ตัวอย่าง:

```
import { Button, Card, Input } from 'react-native-elements';

<Button title="กดปุ่ม" />
<Input placeholder="กรอกข้อความ" />
```

## NativeBase ★★★★★

ติดตั้ง: npm install native-base

จุดเด่น:

- Design system ที่สวยงามและครบวงจร
- มี theme system ในตัว
- รองรับ Responsive design

ตัวอย่าง:

```
import { Button, Box, Text } from 'native-base';

<Button>กดปุ่ม</Button>
<Box bg="blue.500" p={4}>
  <Text color="white">Hello</Text>
</Box>
```

## React Native Paper ★★★★★

ติดตั้ง: npm install react-native-paper

จุดเด่น:

- รองรับ Material Design ของ Google
- ใช้งานง่ายและมีประสิทธิภาพ
- Components สวယงานและเข้ากันได้ดี

ตัวอย่าง:

```
import { Button, Card, TextInput } from 'react-native-paper';

<Button mode="contained">กดปุ่ม</Button>
<TextInput label="กรอกข้อความ" />
```

## UI Kitten ★★★★★

ติดตั้ง: npm install @ui-kitten/components @eva-design/eva

จุดเด่น:

- ใช้ Eva Design System
- สวยงามและกันสมัย
- รองรับ Theming และ Customization

ตัวอย่าง:

```
import { Button, Card, Input } from '@ui-kitten/components';

<Button>กดปุ่ม</Button>
<Input placeholder="กรอกข้อความ" />
```

# สรุป

## JSX

ส่วนขยายไวยากรณ์ JavaScript ที่ช่วยให้เขียน UI ได้ง่าย ต้องมี 1 element หลักเสมอ

## Core Components

View, Text, TextInput, Button, TouchableOpacity คือ "ตัวต่อเลโก้" พื้นฐานสำหรับสร้างแอป

## Props vs State

Props คือข้อมูลจากภายนอก (แก้ไขไม่ได้), State คือข้อมูลภายใน (เปลี่ยนแปลงได้)

## Hooks

useState ใช้จัดการข้อมูลที่เปลี่ยนแปลง, useEffect ใช้จัดการ side effects



# Structuring the App

วางแผนสร้างและจัดการ State ส่วนกลาง

## เป้าหมาย

- จัดโครงสร้างโฟลเดอร์ให้เป็นระเบียบ
- เรียนรู้วิธีจัดการ State ที่ต้องใช้ร่วมกันหลาย ๆ ที่
- สร้างแอป To-Do List ที่มีการจัดการข้อมูลอย่างเป็นระบบ

## ผลลัพธ์

แอป To-Do List ที่มีโครงสร้างเป็นระเบียบ และมีการจัดการข้อมูลแบบรวมศูนย์ด้วย Context API

# ทำไมต้องวางแผนโครงสร้าง?

การวางแผนโครงสร้างโค้ดที่ดีช่วยให้:

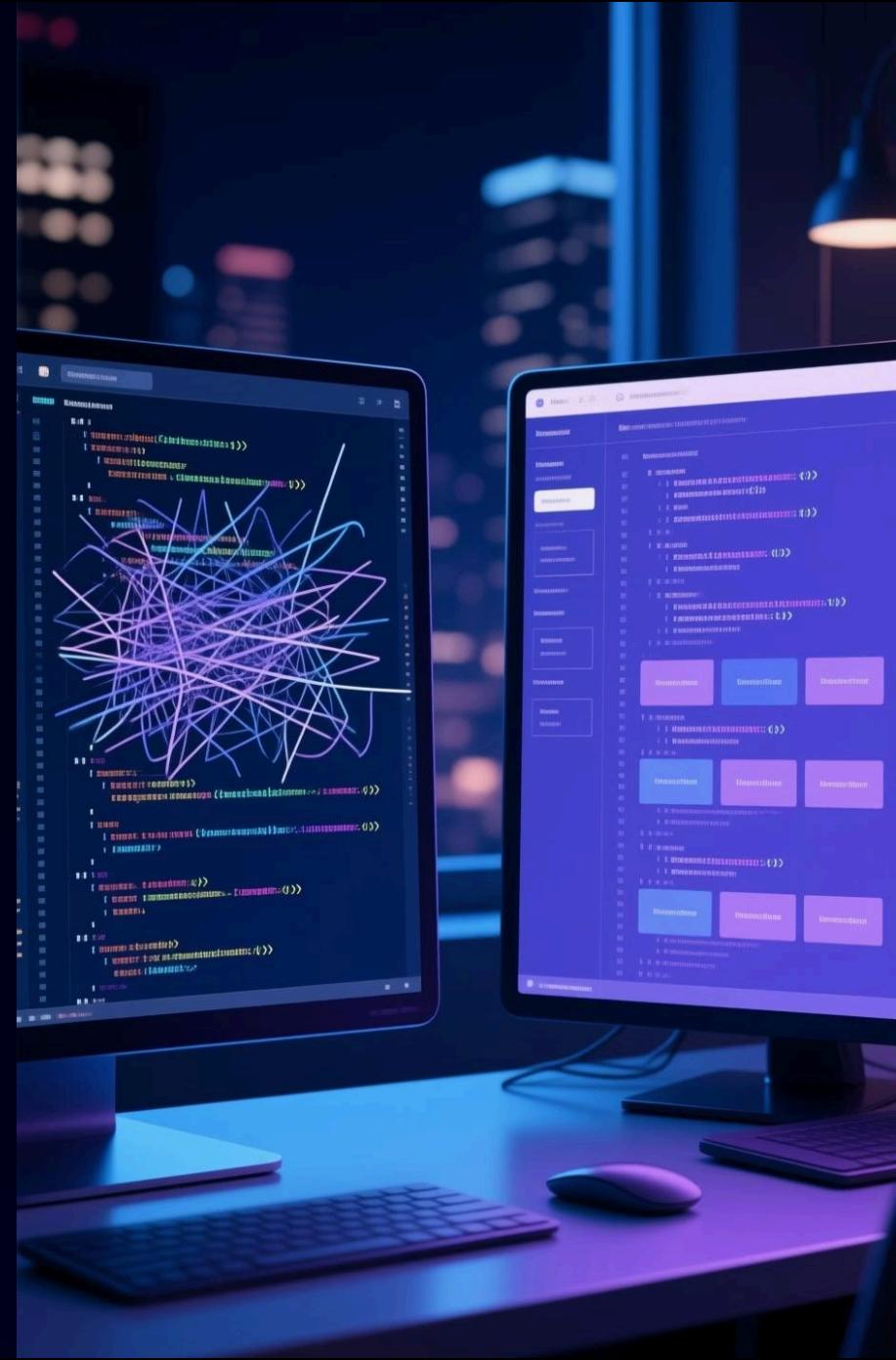
โค้ดหาเจอง่าย

ดูแลรักษาง่าย

ทำงานร่วมกับคนอื่นได้สะดวก

ขยายแอนด์ปรับปรุงได้ง่าย

เปรียบเทียบ: เมื่อ้อนการจัดระเบียบห้องสมุด ถ้าหนังสือวางระยะจะ ใจหนังสือยาก แต่ถ้าจัดเป็นหมวดหมู่ จะหาได้ง่ายและรวดเร็ว



# โครงสร้างไฟล์ที่แนะนำ

การจัดระเบียบโค้ดในโปรเจกต์ React Native เป็นสิ่งสำคัญอย่างยิ่ง เพื่อให้ง่ายต่อการพัฒนา ดูแลรักษา และทำงานร่วมกับผู้อื่น นี่คือโครงสร้างที่แนะนำสำหรับโปรเจกต์ของคุณ:

```
src/
  ├── components/      # Reusable UI components เช่น Button, Card, Header
  |   ├── UserProfile.js # ตัวอย่าง: Component แสดงข้อมูลผู้ใช้
  |   └── ProductCard.js # ตัวอย่าง: Component แสดงข้อมูลสินค้า
  ├── screens/         # UI สำหรับแต่ละหน้าจอของแอป เช่น HomeScreen, SettingsScreen
  |   └── HomeScreen.js # ตัวอย่าง: หน้าหลักของแอป
  ├── providers/       # Context API providers สำหรับ Global State เช่น UserContext, ThemeContext
  |   └── AppContext.js # ตัวอย่าง: Global State Management
  ├── models/          # Data models หรือ Type definitions เช่น UserType, ProductType
  |   ├── User.js        # ตัวอย่าง: User Model
  |   └── Product.js    # ตัวอย่าง: Product Model
  ├── constants/       # ค่าคงที่ต่างๆ เช่น colors, API_KEYS, enums
  |   └── colors.js     # ตัวอย่าง: Color Constants
  └── utils/           # Utility functions, helper functions เช่น formatDate, validateEmail
    └── helpers.js     # ตัวอย่าง: Helper Functions
```

การแบ่งโค้ดออกเป็นส่วนย่อยๆ ตามหน้าที่ ช่วยให้โค้ดสะอาดตา คืนหาส่วนที่ต้องการแก้ไขได้ง่าย และส่งเสริมการนำกลับมาใช้ใหม่ (reusability)

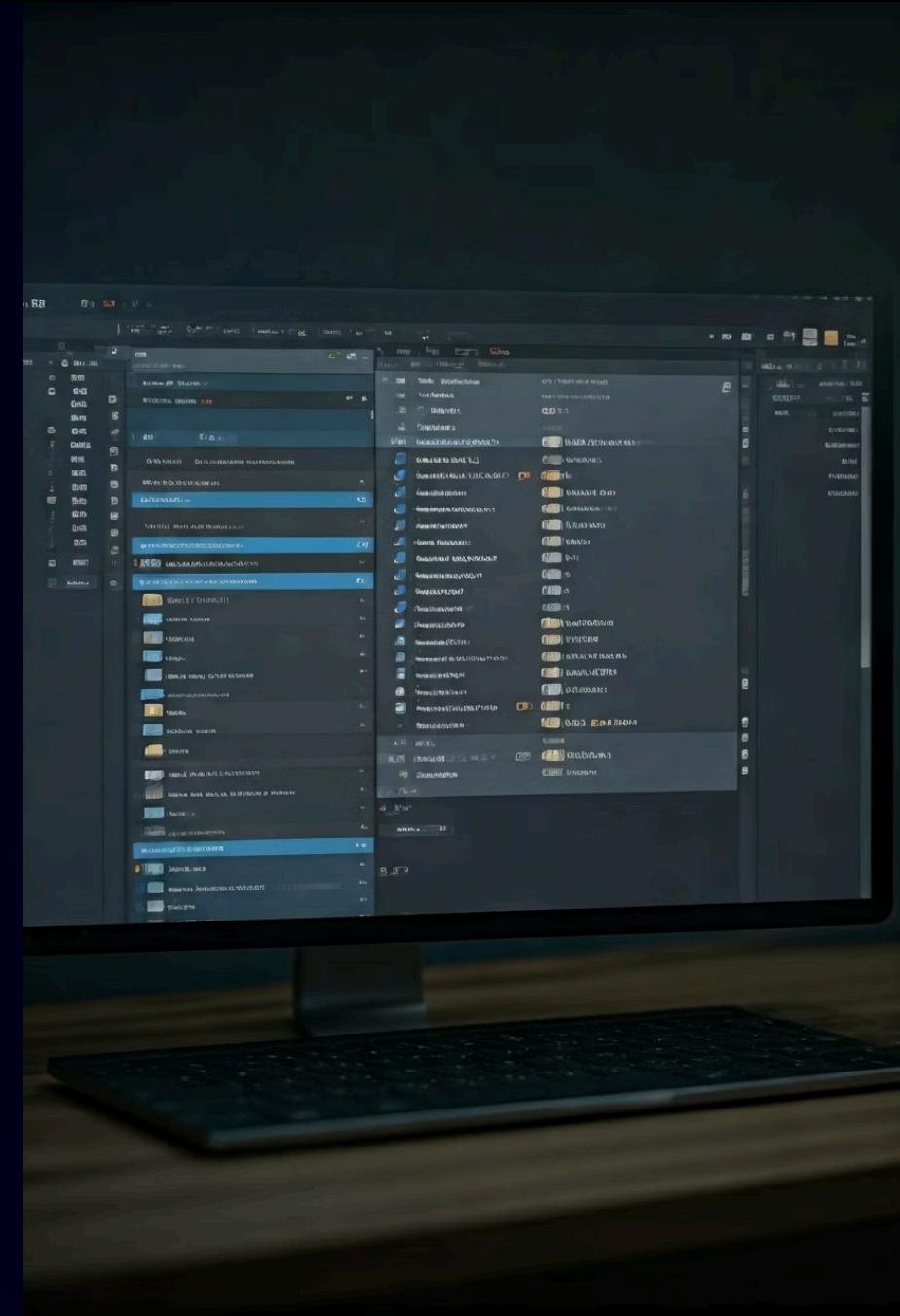
# โปรเจกต์ตัวอย่าง

เพื่อให้เห็นภาพการนำโครงสร้างฟลเดอร์และการจัดการ State ไปใช้จริง เราได้เตรียมโปรเจกต์ตัวอย่างที่สมบูรณ์ไว้ให้คุณได้ศึกษาเพิ่มเติมบน GitHub

โปรเจกต์นี้จะสาธิตวิธีการจัดระเบียบโค้ดตามแนวทางที่เราแนะนำ พร้อมกับแสดงการใช้งาน Context API เพื่อจัดการ State ส่วนกลางของแอปพลิเคชัน

ดูโปรเจกต์บน GitHub

URL: <https://github.com/ignoric/rn-bootcamp.git>



# ปัญหา "Prop Drilling"

**Prop Drilling** คือปัญหาที่เกิดขึ้นเมื่อเราต้องส่ง props ผ่านหลายๆ Component เพื่อให้ลึกลง Component ที่ต้องการใช้จริงๆ

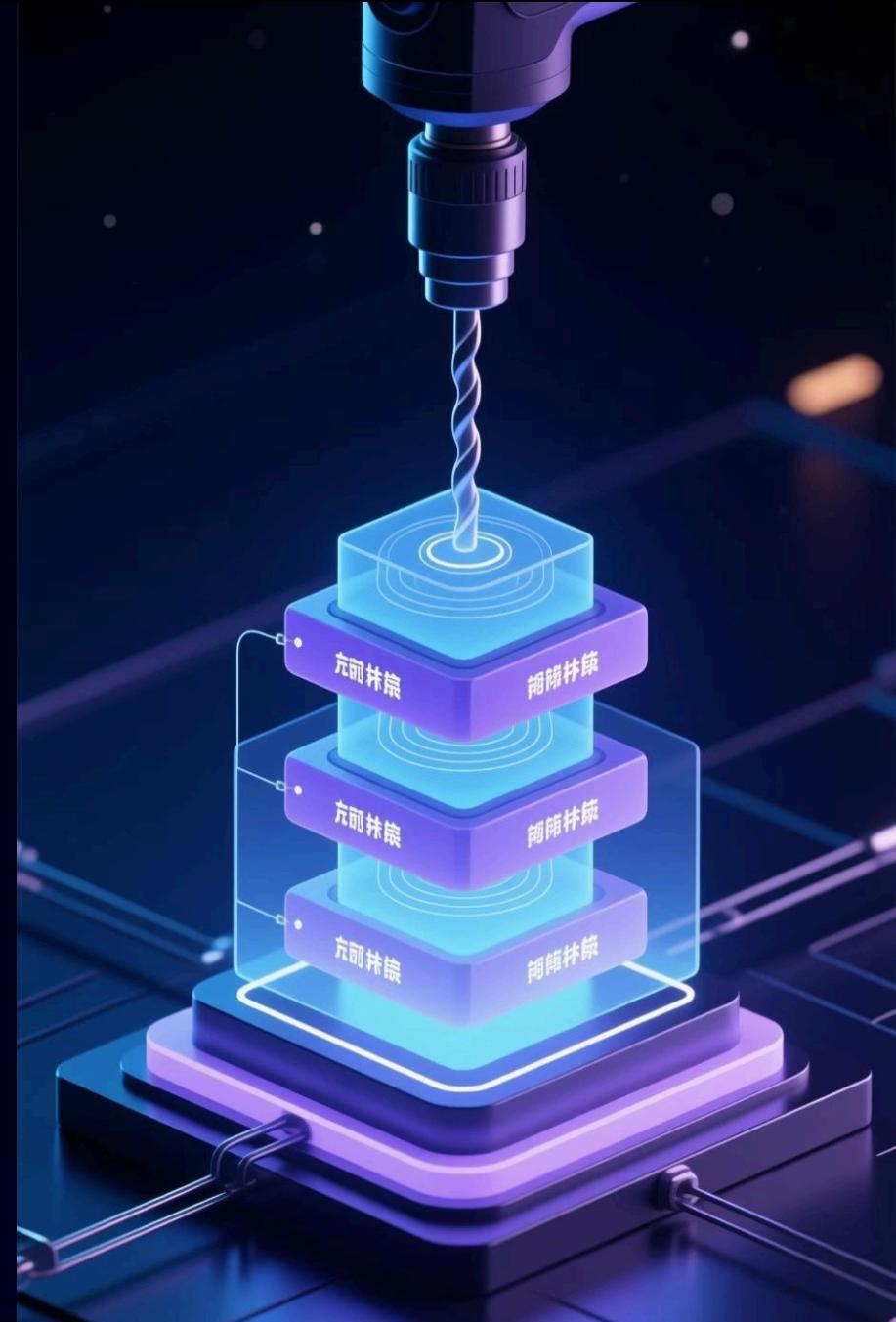
เปรียบเทียบ: ลองนึกภาพการล่า "ขดบ้า" ต่อๆ กันไปให้คนที่อยู่ไกลสุด ก็ง่ายๆ ที่คนตรงกลางไม่ได้ต้องการดื่มน้ำเลย

ปัญหา:

โค้ดซับซ้อน ยากต่อการบำรุงรักษา

Component กลางๆ ต้องรับ props ที่ไม่ได้ใช้

ถ้าเปลี่ยนโครงสร้าง ต้องแก้หลายที่



# Context API

Context API คือวิธีการจัดการ State ส่วนกลางของ React ที่ช่วยแก้ปัญหา Prop Drilling

เปรียบเทียบ: Context สร้าง "ตู้กดน้ำสาธารณะ" (Provider) ที่ครุฑ (Component) ก็สามารถเดินมากดน้ำ (ใช้ข้อมูล) ได้โดยตรง ไม่ต้องส่งต่อกันมา

ประโยชน์:

ลดความซับซ้อนของการส่ง props

จัดการ state ส่วนกลางได้ง่าย

Component ใดๆ สามารถเข้าถึงข้อมูล  
ได้โดยตรง



# Workshop: สร้าง TasksProvider

สร้างไฟล์ `src/providers/TasksProvider.js` (หัวใจหลักในการจัดการข้อมูล)

```
import React, { createContext, useContext, useState } from 'react';

// 1. สร้าง Context object
const TasksContext = createContext();

// 2. สร้าง Component ที่เป็น "Provider"
export function TasksProvider({ children }) {
  const [tasks, setTasks] = useState([]);

  const addTask = (title) => {
    if (title.trim().length === 0) return; // ไม่เพิ่ม task ว่างๆ
    const newTask = { id: Date.now().toString(), title, done: false };
    setTasks(prevTasks => [newTask, ...prevTasks]);
  };

  const toggleTask = (id) => {
    setTasks(tasks.map(task =>
      task.id === id ? { ...task, done: !task.done } : task
    ));
  };

  const removeTask = (id) => {
    setTasks(tasks.filter(task => task.id !== id));
  };

  // 3. ส่ง State และฟังก์ชันต่างๆ ผ่าน value prop
  const value = { tasks, addTask, toggleTask, removeTask };
  return (
    {children}
  );
}

// 4. สร้าง Custom Hook เพื่อให้เรียกใช้งานได้ง่าย
export const useTasks = () => useContext(TasksContext);
```

# Workshop: สร้าง HomeScreen

สร้างไฟล์ src/screens/HomeScreen.js (หน้าจอหลัก)

```
import React, { useState } from 'react';
import { View, Text, TextInput, Button, FlatList, StyleSheet } from 'react-native';
import { useTasks } from '../providers/TasksProvider';

export default function HomeScreen() {
  const { tasks, addTask, toggleTask, removeTask } = useTasks();
  const [text, setText] = useState('');

  const handleAddTask = () => {
    addTask(text);
    setText('');
  };

  return (
    <View>
      <Text>Hello, React Native!</Text>
      <Text>Number of tasks: </Text>
      <Text>{tasks.length}</Text>
      <Text></Text>
      <FlatList
        data={tasks}
        keyExtractor={(item) => item.id}
        renderItem={({ item }) =>
          <Text>{item.text}</Text>
          <Text>[<Button title="Edit" /> <Button title="Delete" />]</Text>
        }
      />
    </View>
  );
}
```

item.id}

renderItem={({ ite...

removeTask(item.id)} /> )} /> ); } const styles = StyleSheet.create({ container: { flex: 1, padding: 10 }, inputContainer: { fl...

# Workshop: ประคอบร่างใน App.js

ปรับปรุงไฟล์ App.js (ไฟล์เริ่มต้นของแอป)

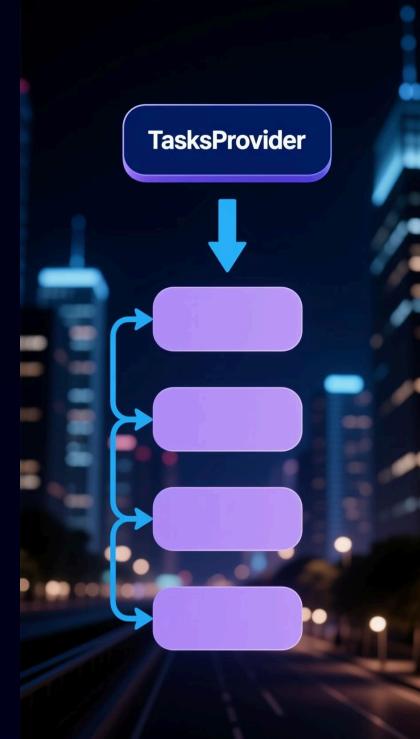
```
import React from 'react';
import { SafeAreaView, StyleSheet } from 'react-native';
import { TasksProvider } from './src/providers/TasksProvider';
import HomeScreen from './src/screens/HomeScreen';

export default function App() {
  return (
    // TasksProvider จะครอบ HomeScreen เพื่อให้ HomeScreen และ component ลูกๆ
    // สามารถเรียกใช้ useTasks() ได้
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#f5f5f5',
  },
});
```

สังเกต:

- เราครอบ <HomeScreen /> ด้วย <TasksProvider>
- ทำให้ HomeScreen และ Component ลูกๆ กั้งหมดสามารถเรียกใช้ useTasks() เพื่อเข้าถึงข้อมูลและฟังก์ชันได้
- ไม่ต้องส่ง props ผ่านหลายชั้น (แก้ปัญหา Prop Drilling)



## To-do-List



## Periwlam

Suimihens  
BeevadæPormipeyiting  
Todasñong.Eath Petropre  
BoobasSulim Roret  
BeevadæEarim နမတ္တစ်။  
BeetvadæPeam Cateve  
PosadeSerim travas  
Beogadæ

## ผลลัพธ์ที่ได้

ตอนนี้เรามีแอป To-Do List เวอร์ชันแรกที่สามารถ:

เพิ่มงานใหม่

ตັກເພື່ອກຳເຄົ່າງໝາຍວ່າເສື້ອຈແລ້ວ

ลบงานที่ไม่ต้องการ

และที่สำคัญ โค้ดของเรามีโครงสร้างที่เป็นระเบียบ และใช้ Context API ในการจัดการ State ส่วนกลาง

# สรุป

## โครงสร้างโฟลเดอร์

จัดโครงสร้างให้เป็นระเบียบ ด้วยการแบ่งโฟลเดอร์เป็น src/components, src/screens, src/providers

## Provider Pattern

ใช้ Provider คลุมส่วนต่างๆ ของแอป เพื่อให้ทุกส่วนเข้าถึงข้อมูลได้

## Context API

ช่วยให้การส่งข้อมูลข้าม Component ง่ายขึ้น โดยใช้ createContext และ useContext

## Custom Hook

สร้าง useTasks() เพื่อให้เรียกใช้ข้อมูลจาก Context ได้ง่ายขึ้น

ในช่วงมองหน้า เราจะมาเรียนรู้วิธีบันทึกข้อมูลในเครื่อง เพื่อให้ข้อมูลไม่หายไปเวลาปิดแอป



# Data Persistence

ข้อมูลอยู่ครบ

## เป้าหมาย

- บันทึกข้อมูลในเครื่อง
- ข้อมูลไม่หายไปเมื่อปิดแอป
- เข้าใจการทำงานแบบ Asynchronous (ไม่พร้อมกัน)

## ผลลัพธ์

แอป To-Do List ที่ข้อมูลไม่หายไป แม้จะปิดแล้วเปิดใหม่

# ปัญหา: ข้อมูลหาย!

ตอนนี้ค่าเราเพิ่งงานในแอป To-Do List และปิดแอป เมื่อเปิดแอปใหม่ รายการงานทั้งหมดจะหายไป

ทำไม? เพราะ State ที่เราใช้ (`useState`) อาศัยอยู่ใน Memory ของแอป เมื่อแอปปิด Memory ก็จะถูกล้าง

ทางแก้: เราต้องบันทึกข้อมูลลงในพื้นที่เก็บข้อมูลภายนอกของเครื่อง

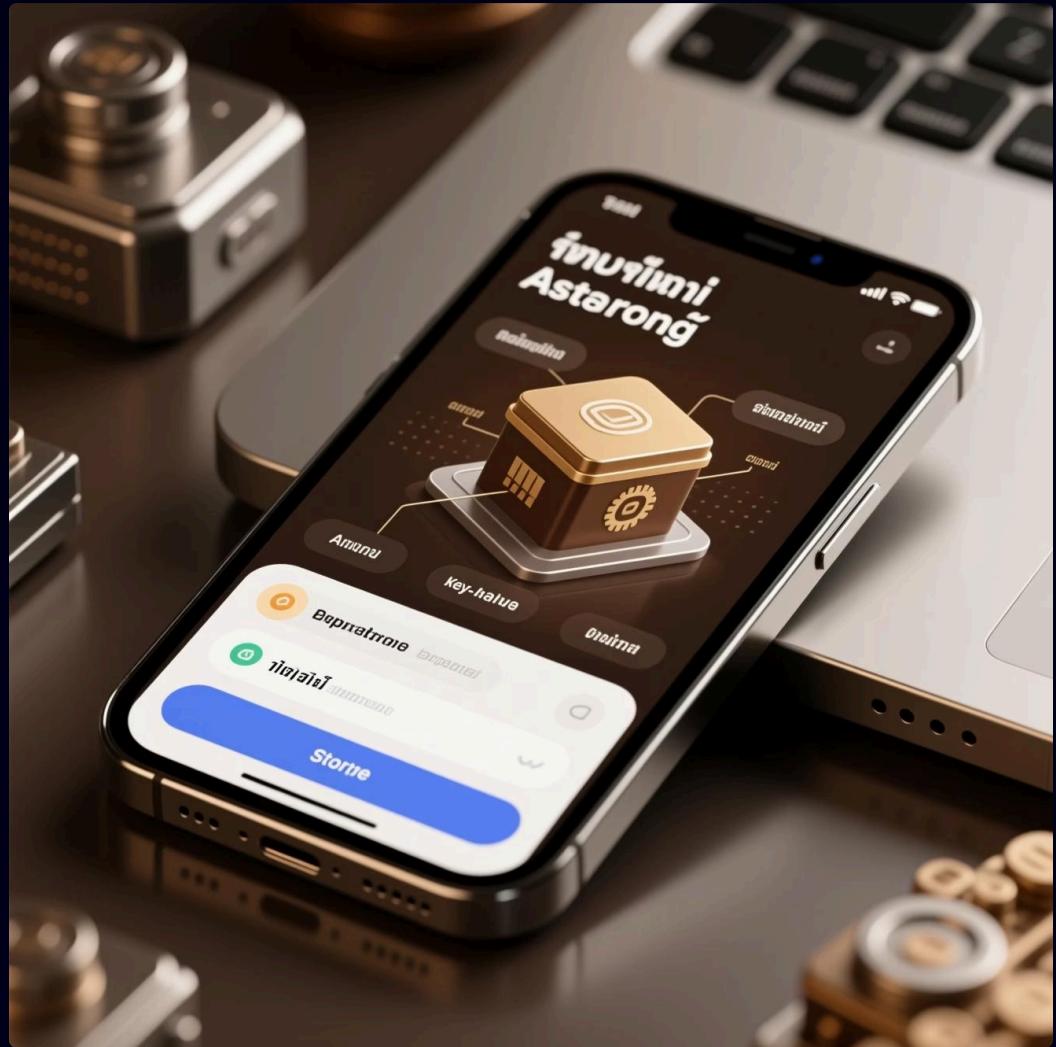


# AsyncStorage คืออะไร?

AsyncStorage เป็นเหมือน "ตู้เก็บของ" ง่ายๆ ในมือถือ สำหรับเก็บข้อมูลแบบ key-value (เป็นข้อความ)

สักขนะส่าคัญ:

- เก็บข้อมูลแบบ Key-Value: เหมือนกับ Object ใน JavaScript
- เก็บได้เฉพาะข้อความ: ต้องแปลง Object/Array เป็นข้อความก่อนด้วย JSON.stringify()
- Async (Asynchronous): การทำงานเป็นแบบ "ไม่รอ" หมายความว่าเมื่อเราสั่งบันทึกข้อมูล แล้วฯ จะไม่หยุดค้างเพื่อรอให้บันทึกเสร็จ



# การใช้งาน AsyncStorage

ก่อนใช้งาน ต้องติดตั้งก่อน: npx expo install @react-native-async-storage/async-storage

## บันทึกข้อมูล

```
// บันทึกข้อความ  
await AsyncStorage.setItem('username',  
'สมชาย');  
  
// บันทึก Object/Array (ต้องแปลงเป็นข้อความ  
ก่อน)  
const tasks = [{ id: '1', title: 'ซื้อของ', done:  
false }];  
await AsyncStorage.setItem('tasks',  
JSON.stringify(tasks));
```

## อ่านข้อมูล

```
// อ่านข้อความ  
const username = await  
AsyncStorage.getItem('username');  
console.log(username); // 'สมชาย'  
  
// อ่าน Object/Array (ต้องแปลงกลับ)  
const tasksJson = await  
AsyncStorage.getItem('tasks');  
const tasks = JSON.parse(tasksJson);  
console.log(tasks); // [{ id: '1', title: 'ซื้อ  
ของ', done: false }]
```

## ลบข้อมูล

```
// ลบข้อมูลบางรายการ  
await  
AsyncStorage.removeItem('username');  
  
// ลบข้อมูลทั้งหมด  
await AsyncStorage.clear();
```

# Workshop: ប្រើប្រាស់ TasksProvider (ទេសចរណ៍មុខ)

ប្រើប្រាស់កូដខាងក្រោមនេះដើម្បីបង្កើតកម្មវិធី

```
import React, { createContext, useContext, useState, useEffect } from 'react';
import AsyncStorage from '@react-native-async-storage/async-storage';

const TASKS_STORAGE_KEY = '@tasks_data';
const TasksContext = createContext();

export function TasksProvider({ children }) {
  const [tasks, setTasks] = useState([]);
  const [isLoading, setIsLoading] = useState(true);

  // useEffect ដំឡើល "ទេសចរណ៍" មុខទៅបានបើកប្រើប្រាស់
  useEffect(() => {
    async function loadTasksFromStorage() {
      try {
        const storedTasks = await AsyncStorage.getItem(TASKS_STORAGE_KEY);
        if (storedTasks !== null) {
          setTasks(JSON.parse(storedTasks));
        }
      } catch (e) {
        console.error("Failed to load tasks.", e);
      } finally {
        setIsLoading(false);
      }
    }
    loadTasksFromStorage();
  }, []); // Dependency array គាំង ហើយតើងໃរកការងារនេះឡើង

  // ... ផែងក្រោម addTask, toggleTask, removeTask អំពីការបង្កើត ...

  const value = { tasks, addTask, toggleTask, removeTask, isLoading };
  return (
    
      {children}
    
  );
}

export const useTasks = () => useContext(TasksContext);
```

# Workshop: ปรับปรุง TasksProvider (บันทึกข้อมูล)

เพิ่ม useEffect อีกตัวเพื่อบันทึกข้อมูลทุกครั้งที่ tasks เปลี่ยนแปลง

```
// เพิ่มโค้ดนี้ต่อจาก useEffect ตัวแรก
// useEffect สำหรับ "บันทึก" ข้อมูลทุกครั้งที่ tasks เปลี่ยนแปลง
useEffect(() => {
  async function saveTasksToStorage() {
    // ไม่ต้องบันทึกตอนที่ยังโหลดข้อมูลครั้งแรกไม่เสร็จ
    if (!isLoading) {
      try {
        const jsonValue = JSON.stringify(tasks);
        await AsyncStorage.setItem(TASKS_STORAGE_KEY, jsonValue);
      } catch (e) {
        console.error("Failed to save tasks.", e);
      }
    }
  }
  saveTasksToStorage();
}, [tasks, isLoading]); // Dependency array มี tasks หมายถึงให้ทำงานทุกครั้งที่ tasks เปลี่ยน
```

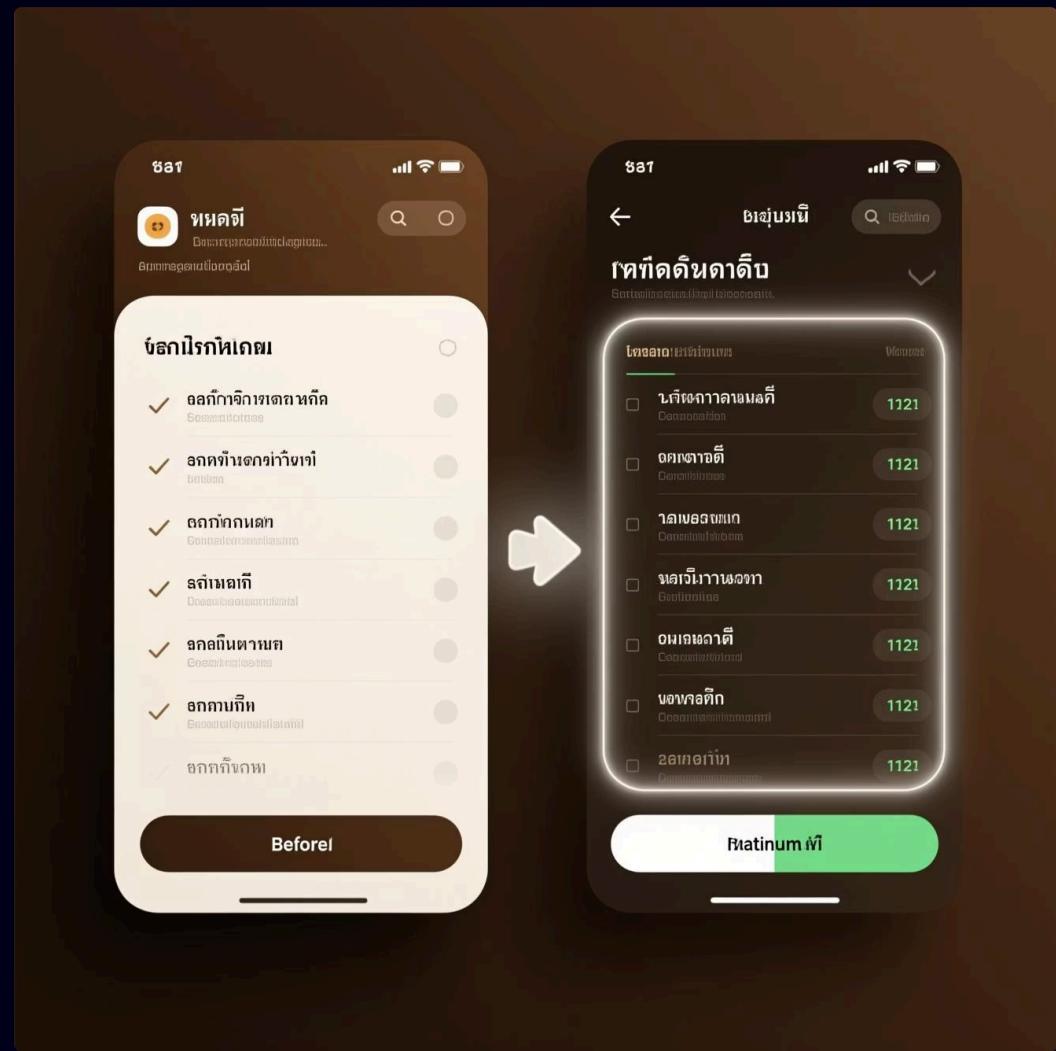
สังเกต: เราใช้ useEffect 2 ตัว ตัวแรกทำงานครั้งเดียวตอนเปิดแอป (โหลดข้อมูล) ตัวที่สองทำงานทุกครั้งที่ tasks เปลี่ยน (บันทึกข้อมูล)

# ผลลัพธ์และทดสอบ

ตอนนี้แอป To-Do List ของเราสามารถเก็บข้อมูลไว้ได้เมื่อจะปิดแอปและเปิดใหม่

ลองทดสอบ:

- เพิ่มงานใหม่ 2-3 รายการ
- ตีกเครื่องหมายว่าเสร็จบางรายการ
- ปิดแอป (ลากกัน Home และปิด)
- เปิดแอปใหม่
- สังเกตว่าข้อมูลยังอยู่ครบถ้วน



# สรุปช่วง蒙กี่ 4

## AsyncStorage

ระบบเก็บข้อมูลแบบ Key-Value ในมือถือ ที่ช่วยให้ข้อมูลคงอยู่แม้ปิดแอป

## JSON.stringify / JSON.parse

ใช้แปลง Object/Array เป็นข้อความและกลับ เพื่อเก็บใน AsyncStorage

## useEffect สำหรับโหลดข้อมูล

ใช้ useEffect ที่มี dependency array เป็น [] เพื่อโหลดข้อมูลครั้งเดียวตอนเปิดแอป

## useEffect สำหรับบันทึกข้อมูล

ใช้ useEffect ที่มี dependency array เป็น [tasks] เพื่อบันทึกข้อมูลทุกครั้งที่ tasks เปลี่ยน

ในช่วง蒙กี่ 4 เราจะเรียนรู้การดำเนินการหลายหน้าและการเชื่อมต่อ กับ API ภายนอก

# ចំណាំទី 5: Navigation & APIs

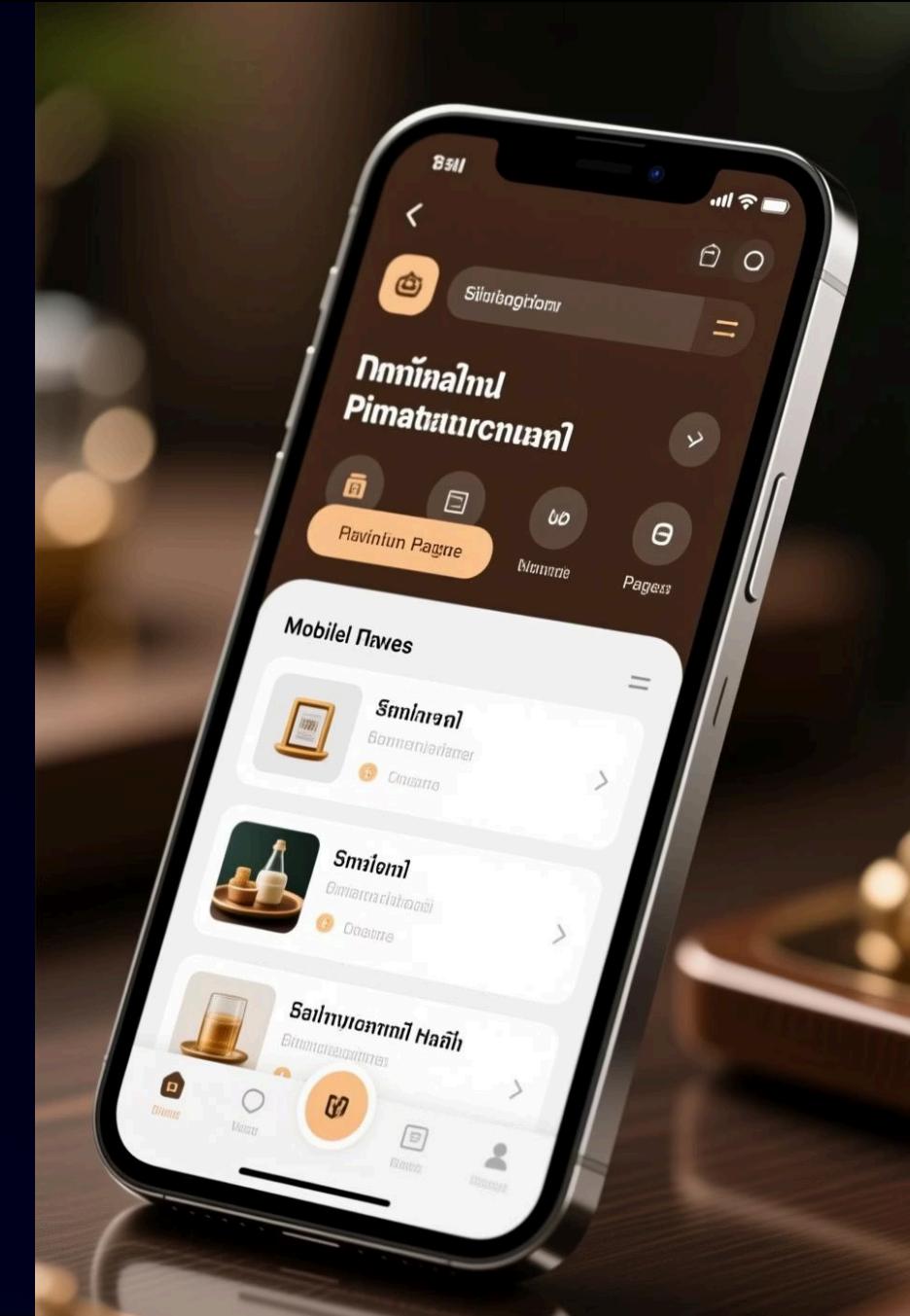
ការកំឡាយហ្មាត់និងផ្ទេរព័ត៌មាន

## ផែអុយ

- សរាប់អេបក់ដែលមានការកំឡាយហ្មាត់
- រួមចំណាំការកំឡាយនៃការប្រើប្រាស់ផែអុយ
- បង្កើតការកំឡាយដោយប្រើប្រាស់បច្ចេកទេស (API) និងផែអុយ

## ផលិតផល

ផែអុយ To-Do List ដែលមានការកំឡាយហ្មាត់ និងបានបញ្ជាក់ថាបានបញ្ចប់ដោយប្រើប្រាស់ការកំឡាយហ្មាត់



# React Navigation

React Navigation គឺ Library មានច្បាស់អនុវត្តការណ៍បង្កើតនៃការផ្ទាល់ខ្លួននៅក្នុង React Native

**Stack Navigator:** រូបແບບការបង្ការការងារដែលមានការបង្ហាញជាការបង្ហាញបំពុំ (តាមការបង្ហាញមិនមែនបានបង្ហាញឡើង) ការផ្តល់ការបង្ហាញតាមការបង្ហាញបំពុំ និងការបង្ហាញតាមការបង្ហាញបំពុំ។

ស៊ីវិភាគកំណត់:

- NavigationContainer: តួគោរបក្រុមដែលបង្ហាញការបង្ហាញបំពុំ
- createNativeStackNavigator: ផែនការដែលបង្ហាញការបង្ហាញបំពុំ



# Workshop: ติดตั้งและสร้าง Navigator

ติดตั้ง React Navigation:

```
npx expo install @react-navigation/native react-native-screens react-native-safe-area-context @react-navigation/native-stack
```

สร้างไฟล์ src/navigation/AppNavigator.js:

```
import React from 'react';
import { NavigationContainer } from '@react-navigation/native';
import { createStackNavigator } from '@react-navigation/native-stack';
import HomeScreen from '../screens/HomeScreen';
import SettingsScreen from '../screens/SettingsScreen';

const Stack = createStackNavigator();

export default function AppNavigator() {
  return (
    );
}
```

# Workshop: สร้างหน้า Settings

สร้างไฟล์ `src/screens/SettingsScreen.js`:

```
import React from 'react';
import { View, Text, Button, StyleSheet, Alert } from 'react-native';
import { useTasks } from '../providers/TasksProvider';

export default function SettingsScreen() {
  const { tasks, removeTask } = useTasks();

  const clearCompleted = () => {
    Alert.alert("ยืนยัน", "คุณต้องการลบงานที่เสร็จแล้วทั้งหมดหรือไม่?", [
      { text: "ยกเลิก", style: "cancel" },
      { text: "ตกลง", onPress: () => {
        tasks.forEach(task => {
          if (task.done) {
            removeTask(task.id);
          }
        });
      }}
    ]);
  };

  return (
    <View>
      <Text>ตั้งค่า</Text>
      <Text>ลบงาน</Text>
      <Text>ลบทั้งหมด</Text>
    </View>
  );
}

const styles = StyleSheet.create({ container: { flex: 1, padding: 20 }, title: { fontSize: 24, marginBottom: 20, fontWeight: 'bold' } });
```

ปรับปรุงไฟล์ `App.js` ให้ใช้ Navigator:

```
import React from 'react';
import { TasksProvider } from '../src/providers/TasksProvider';
import AppNavigator from '../src/navigation/AppNavigator';

export default function App() {
  return (
    <TasksProvider>
      <AppNavigator>
    </AppNavigator>
  );
}
```

# API คืออะไร?

API (Application Programming Interface) คือช่องทางที่ให้แอปของเราระบบสามารถสื่อสารกับเซิร์ฟเวอร์อื่นๆ เพื่อขอข้อมูลหรือบริการ

เช่น: API เหนือ "เมนูอาหาร" ของร้านอาหาร (เซิร์ฟเวอร์) ที่บอกว่าเราสามารถสั่งอะไรได้บ้าง และจะได้อะไรกลับมา

ประโยชน์:

- ดึงข้อมูลจากแหล่งภายนอก
- ส่งข้อมูลไปเก็บบนเซิร์ฟเวอร์
- ใช้บริการจากผู้ให้บริการอื่นๆ



# การดึงข้อมูลด้วย fetch

**fetch** คือฟังก์ชันมาตรฐานของ JavaScript ที่ใช้ในการดึงข้อมูลจาก API

**async/await** คือไวยากรณ์ที่ช่วยให้เขียนโค้ดที่อ่อน "รอ" ( เช่น รอข้อมูลจาก API ) ให้ดูเหมือนโค้ดปกติ อ่านง่ายขึ้นมาก

**การจัดการสถานะ:** เป็นเรื่องสำคัญมากที่ต้องบอกผู้ใช้ว่าตอนนี้กำลังโหลด... (Loading), โหลดสำเร็จ (Success), หรือเกิดข้อผิดพลาด (Error)

```
// พังก์ชันดึงข้อมูลจาก API
async function fetchQuote() {
  try {
    // แสดงสถานะกำลังโหลด
    setLoading(true);

    // ดึงข้อมูลจาก API
    const response = await fetch('https://api.quotable.io/random');

    // ตรวจสอบว่าการดึงข้อมูลสำเร็จหรือไม่
    if (!response.ok) {
      throw new Error('Network response was not ok');
    }

    // แปลงข้อมูล JSON เป็น JavaScript Object
    const data = await response.json();

    // อัปเดต state ด้วยข้อมูลที่ได้
    setQuote(data.content);
  } catch (error) {
    // จัดการข้อผิดพลาด
    setError('ไม่สามารถโหลดข้อมูลได้');
  } finally {
    // ปิดสถานะกำลังโหลด (ไม่ว่าจะสำเร็จหรือล้มเหลว)
    setLoading(false);
  }
}
```

# Workshop: ดึง Quote API

ปรับปรุงไฟล์ `src/screens/HomeScreen.js` เพื่อเพิ่มการดึงข้อมูลคำคม:

```
// เพิ่ม imports
import { useEffect } from 'react';
import { useNavigation } from '@react-navigation/native';

export default function HomeScreen() {
  // ... โค้ดเดิม ...
  const navigation = useNavigation(); // Hook สำหรับเรียกใช้ navigation

  // State สำหรับ API
  const [quote, setQuote] = useState('');
  const [loadingQuote, setLoadingQuote] = useState(false);
  const [quoteError, setQuoteError] = useState('');

  const fetchQuote = async () => {
    setLoadingQuote(true);
    try {
      const response = await fetch('https://api.quotable.io/random');
      const data = await response.json();
      setQuote({ content: data.content, author: data.author });
    } catch (e) {
      console.error(e);
      setQuoteError('โหลดคำคมไม่สำเร็จ');
    } finally {
      setLoadingQuote(false);
    }
  };

  useEffect(() => {
    fetchQuote();
  }, []);
}

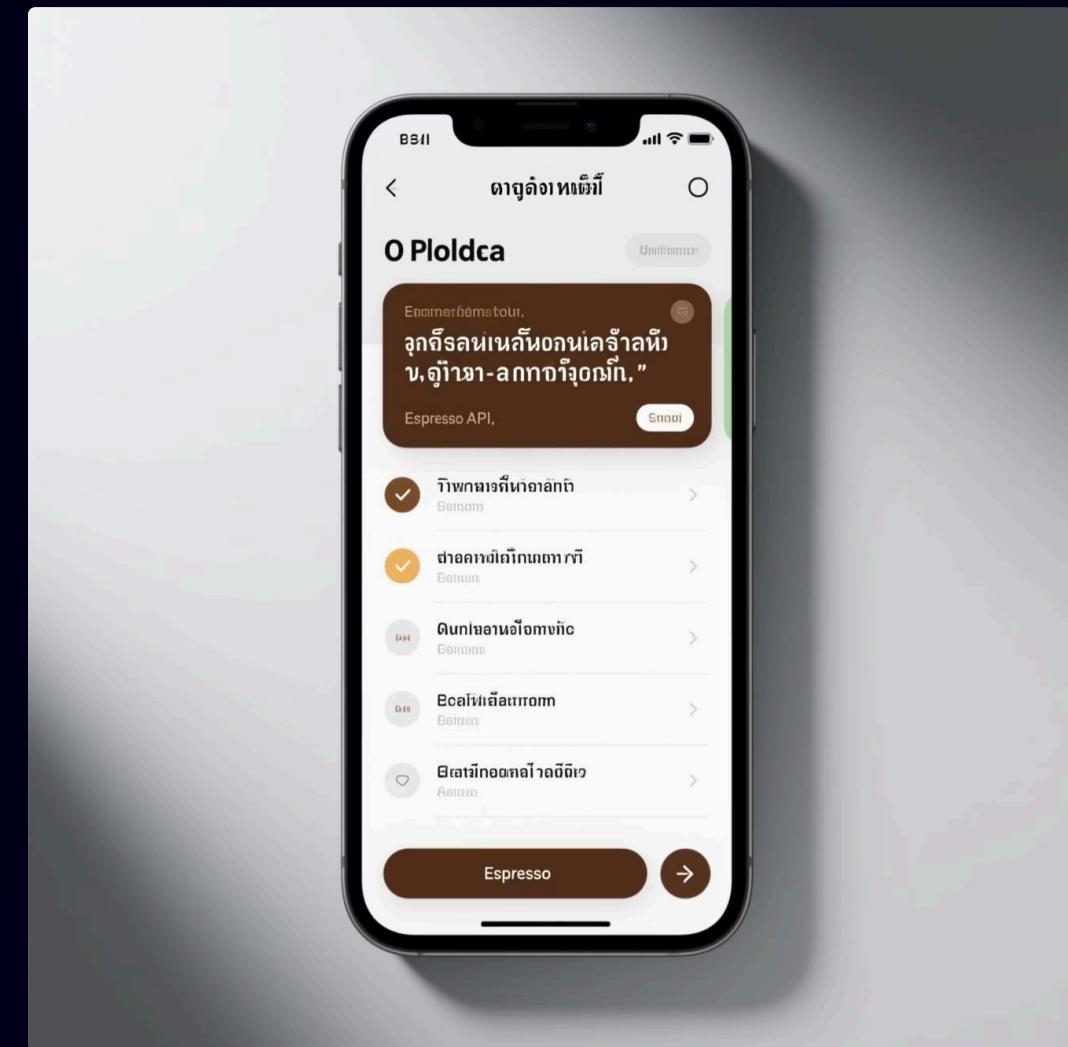
// เพิ่มปุ่ม Settings ใน Header
useEffect(() => {
  navigation.setOptions({
    headerRight: () => (
      ตั้งค่า
    ), [navigation];
  });
  // ... แก้ไข return เพื่อแสดง Quote ... return (
    /* ส่วนแสดง Quote */
    /* ... ส่วน Input และ FlatList ให้มองเห็น ... */ );
}); // ... เพิ่ม styl...
```

# ผลลัพธ์ที่ได้

ตอนนี้แอป To-Do List ของเรามี:

- หน้าหลัก (Home และ Settings)
- การนำทางระหว่างหน้า
- การดึงข้อมูลคำคำนจาก API ภายนอก
- การจัดการสถานะการโหลดและข้อผิดพลาด

และที่สำคัญ ข้อมูลงานยังคงถูกบันทึกไว้ในเครื่องด้วย AsyncStorage



# สรุปชั่วโมงที่ 5

## React Navigation

จัดการหลายหน้าจอและการนำทางระหว่างหน้า

## Stack Navigator

รูปแบบการนำทางแบบซ้อนกัน มีปุ่ม Back อัตโนมัติ

## API

ช่องทางสื่อสารกับเซิร์ฟเวอร์ภายนอก

## fetch & async/await

เครื่องมือสำหรับดึงข้อมูลจาก API และจัดการโค้ดแบบ Asynchronous

ในชั่วโมงสุดท้าย เราจะเรียนรู้การปรับปรุงประสิทธิภาพของแอปและสรุปภาพรวมทั้งหมด

# ชั้วโมงที่ 6: Performance & Polishing

ปรับปรุงประสิทธิภาพและเก็บตก

## เป้าหมาย

- เรียนรู้เทคนิคพื้นฐานในการทำให้อแอปทำงานเร็วขึ้น
- ทำความเข้าใจการ Debugging เบื้องต้น
- สรุปภาพรวมทั้งหมดและแนวการทำงานเรียนรู้ต่อ

## ผลลัพธ์

แอป To-Do List ที่มีประสิทธิภาพดี และความเข้าใจในการพัฒนาแอปด้วย React Native อย่างลึกซึ้ง

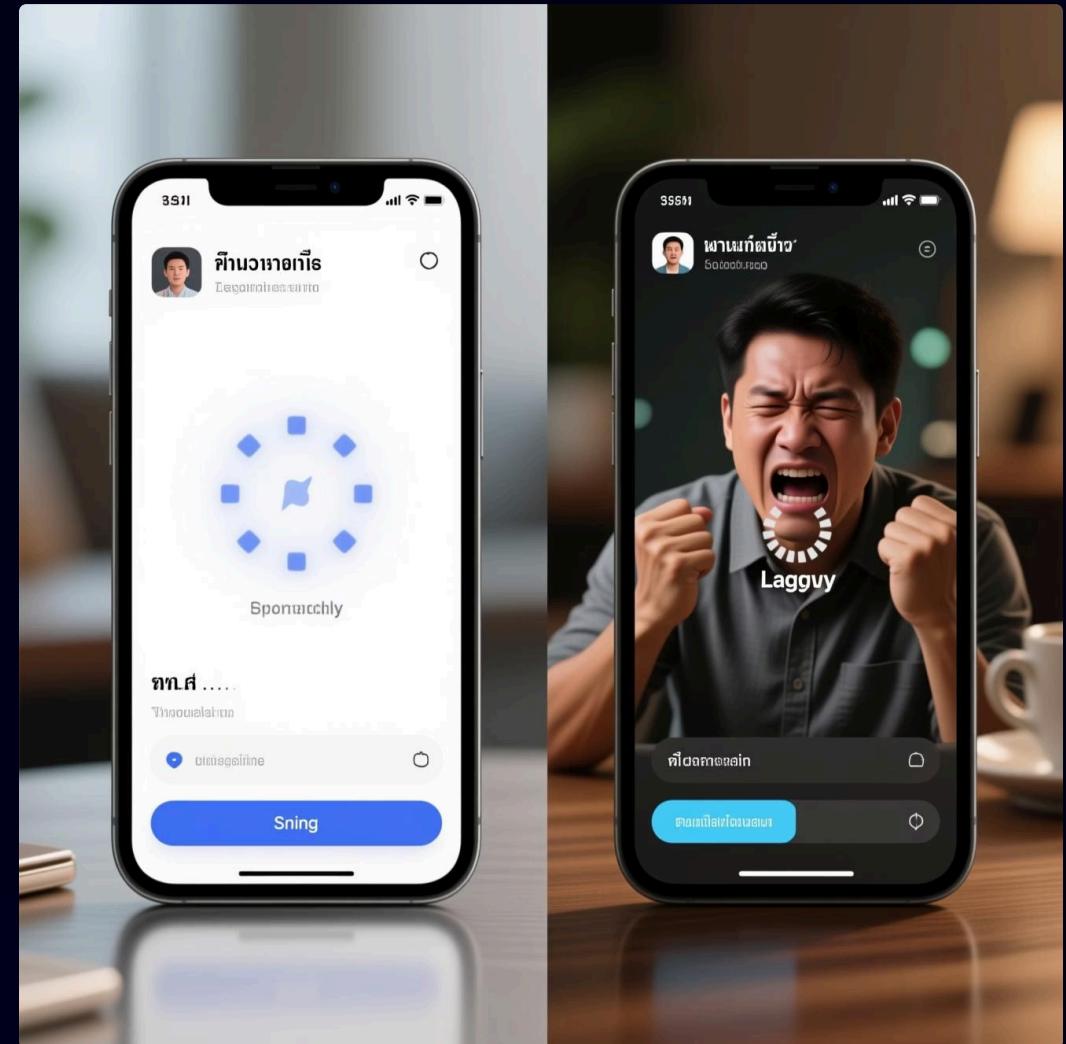


# ทำไม Performance ถึงสำคัญ?

แอปที่ลื่นไหลสร้างประสบการณ์ดีให้ผู้ใช้ ในขณะที่แอปที่กระตุกอาจทำให้ผู้ใช้ลับแอปทิ้งได้

ผลกระทบของ Performance ที่ไม่ดี:

- ผู้ใช้รู้สึกหงุดหงิด
- อัตราการลับแอปสูง
- รีวิวแย่บน App Store/Play Store
- ใช้แบตเตอรี่มากเกินไป



# เครื่องมือ #1: FlatList

**FlatList** คือ Component สำหรับแสดงรายการยาวๆ ที่ต้องลากมาก มันจะ Render เดพารายการที่เห็นบนจอเท่านั้น ทำให้แอปไม่ซ้ำແบ້ຈະນีข้อมูลเป็นพันรายการ

ข้อดี:

- Render เดพารายการที่เห็นบนจอ
- รองรับรายการจำนวนมาก
- มี Pull-to-refresh ในตัว
- มี Infinite Scrolling ในตัว

```
item.id}
renderItem={({ item }) => (
)}
// แสดงข้อความเมื่อไม่มีรายการ
ListEmptyComponent={
  // ยังไม่มีงานในรายการ
}
// แสดงตัวโหลดเมื่อถึงข้อมูลเพิ่ม
onEndReached={loadMoreTasks}
onEndReachedThreshold={0.5}
/>>
```

# เครื่องมือ #2: React.memo

**React.memo** เป็นการ "ห่อ" Component เพื่อบอกว่า "ถ้า Props ที่ส่งเข้ามาไม่เปลี่ยนแปลง ก็ไม่ต้อง Render ใหม่บะ ใช้ของเก่าไปเลย"

เหมาะกับ Component ในรายการอย่าง TaskItem ที่อาจมีจำนวนมาก และส่วนใหญ่ไม่ได้เปลี่ยนแปลงเมื่อมีการ Render ใหม่

ข้อดี:

- ลดการ Render ที่ไม่จำเป็น
- เพิ่มประสิทธิภาพโดยเฉพาะกับรายการที่มีจำนวนมาก

```
// TaskItem.js
import React from 'react';
import { View, Text, Button } from 'react-native';

function TaskItem({ task, onToggle, onRemove }) {
  console.log(`Rendering Task: ${task.title}`);
  return (
    
      {task.title}
      
      
    
  );
}

export default TaskItem;
```

onRemove(task.id)} /> ); } // ห่อด้วย React.memo เพื่อป้องกันการ re...

# เครื่องมือ #3: useCallback

**useCallback** ช่วย "จำ" ฟังก์ชันที่เราสร้างขึ้น เพื่อไม่ให้มันถูกสร้างใหม่ทุกครั้งที่ Component แม่ Render

ซึ่งสำคัญมากเมื่อเราส่งฟังก์ชันนั้นไปให้ Component ลูกที่ถูกห่อด้วย React.memo

ทำไมต้องใช้: ถ้าไม่ใช้ useCallback ฟังก์ชันจะถูกสร้างใหม่ทุกครั้งที่ Component แม่ Render ทำให้ React.memo ไม่มีประโยชน์ เพราะ props (ฟังก์ชัน) จะเปลี่ยนทุกครั้ง

```
// HomeScreen.js
import React, { useCallback } from 'react';

export default function HomeScreen() {
  const { tasks, toggleTask, removeTask } = useTasks();

  // ใช้ useCallback เพื่อ "จำ" ฟังก์ชันไว้
  // ไม่ให้ถูกสร้างใหม่ทุกครั้งที่ re-render
  const handleToggleTask = useCallback((id) => {
    toggleTask(id);
  }, [toggleTask]);

  const handleRemoveTask = useCallback((id) => {
    removeTask(id);
  }, [removeTask]);

  return (
    (
      )
    />

    );
}
```

# Workshop: ปรับปรุง Performance

สร้างไฟล์ src/components/TaskItem.js เพื่อแยก Logic และใช้ React.memo

```
import React from 'react';
import { View, Text, Button, StyleSheet } from 'react-native';

function TaskItem({ task, onToggle, onRemove }) {
  console.log(`Rendering Task: ${task.title}`); // เพื่อดูว่า re-render เมื่อไหร่
  return (
    
      <Text>{task.title}</Text>
      <Text>{task.description}</Text>
      <Text>{task.dueDate}</Text>
      <Text>{task.priority}</Text>
      <Text>{task.isDone ? 'Done' : 'Not Done'}</Text>
    
  );
}

const styles = StyleSheet.create({
  taskContainer: {
    flex: 1,
    padding: 10,
    margin: 10,
    border: 1px solid #ccc,
    borderRadius: 10,
    position: 'relative',
  },
  taskText: {
    color: '#333',
    font: 'bold',
    margin: 0,
  },
  taskDescription: {
    color: '#666',
    margin: 0,
  },
  taskDueDate: {
    color: '#666',
    margin: 0,
  },
  taskPriority: {
    color: '#666',
    margin: 0,
  },
  taskIsDone: {
    color: '#666',
    margin: 0,
  },
});

export default TaskItem;
```



# การ Debugging เปื้องตัว

## console.log()

วิธีง่ายที่สุดในการดูค่าของตัวแปร ณ เวลาต่างๆ

```
console.log('tasks:', tasks);
console.log('Current state:', { text,
  isLoading, error });
```

## Developer Menu

เขย่าเครื่องหรือกด Cmd+D (iOS) / Ctrl+M (Android) เพื่อเปิดเมนูบักพัฒนา

- Toggle Inspector: ตรวจสอบ Component
- Performance Monitor: ดูประสิทธิภาพ
- Debug JS Remotely: Debug ด้วย Chrome

## Error Boundaries

Component พิเศษที่จับข้อผิดพลาดและแสดงข้อความແກบที่จะให้แอปพลิเคชันดำเนินต่อไป

> เกิดข้อผิดพลาด{>

# สรุปภาพรวมสถาปัตยกรรม

แอป To-Do List ของเราใช้สถาปัตยกรรมที่เป็นระบบและขยายได้ง่าย



## Components

UI ชิ้นเล็กๆ ที่ใช้ช้าได้ เช่น TaskItem



## Screens

หน้าจอหลักของแอป เช่น HomeScreen, SettingsScreen



## Providers

จัดการ State ส่วนกลาง เช่น TasksProvider



## Navigation

จัดการการนำทางระหว่างหน้า



## Services

จัดการการเชื่อมต่อ กับ API และการเก็บข้อมูล

# แนวทางการเรียนรู้ต่อ



## State Management

สำหรับการจัดการ State ที่ซับซ้อนมากๆ ในแอปขนาดใหญ่

- Redux Toolkit
- Zustand
- MobX



## Data Fetching

สำหรับจัดการการดึงข้อมูลจาก API ที่ซับซ้อน

- TanStack Query (React Query)
- SWR
- Apollo Client (GraphQL)



## UI Libraries

ชุด Component สำเร็จรูป sway ที่ช่วยให้ลร้างแอปได้เร็วขึ้น

- React Native Paper
- Native Base
- UI Kitten



## Testing

การทดสอบแอปเพื่อให้มั่นใจว่าทำงานถูกต้อง

- Jest
- React Native Testing Library
- Detox (E2E Testing)

# ตัวอย่างการบ้าน

ลองท้าทายตัวเองด้วยการเพิ่มฟีเจอร์ใหม่ให้กับแอป To-Do List

## เพิ่มน้ำᾳ้ไข

สร้างหน้าจอ "Edit Task" ที่สามารถแก้ไขข้อความของ To-Do ได้

- เพิ่ม Screen ใหม่ใน Navigator
- สร้างฟอร์มแก้ไขข้อความ
- เพิ่มฟังก์ชัน editTask ใน TasksProvider

## ระบบ Theme

สร้าง Provider อีกตัวสำหรับจัดการ Theme (Light/Dark Mode) และปุ่มสลับ Theme ในหน้า Settings

- สร้าง ThemeProvider
- กำหนดชุดสีสำหรับแต่ละ Theme
- บันทึกการตั้งค่าด้วย AsyncStorage

## ตัวกรอง

เพิ่มปุ่มสำหรับกรองรายการ (ทั้งหมด / ที่กำลังใช้งาน / ที่ยังไม่เสร็จ)

- สร้าง Component FilterBar
- เพิ่ม State สำหรับเก็บค่าตัวกรอง
- กรองรายการก่อนส่งให้ FlatList

← To-Do →

ໃຫ້ຄວາມກວບຮັກ



To-Do List



ການຄົ້ນພື້ນ

ໄຟເຫດສະບູລົງອຸທອອນ



ຕະຫຼາດ teso



ມາເກົດຈີ

ມາເກົດຈີ



ອາກາຈັງ

Odbod kipane, laemhyat



Edit list

# สรุปและขอบคุณ

ในหลักสูตร 6 ชั่วโมงนี้ เราได้เรียนรู้:

- พื้นฐาน React Native และ JavaScript
- การสร้าง UI ด้วย Components
- การจัดการ State และ Props
- การใช้ Hooks (useState, useEffect, useCallback)
- การจัดโครงสร้างแอปและจัดการ State ส่วนกลาง
- การบันทึกข้อมูลด้วย AsyncStorage
- การทำแอปหลายหน้าด้วย Navigation
- การเชื่อมต่อภายนอก API
- การปรับปรุงประสิทธิภาพของแอป

และที่สำคัญที่สุด เราได้สร้างแอป To-Do List ก็ใช้งานได้จริง มีหลายหน้า บันทึกข้อมูลได้ และเชื่อมต่อกับอินเทอร์เน็ตได้!

นี่เป็นเพียงจุดเริ่มต้นของการเดินทางในโลกของการพัฒนาแอปมือถือ ยังมีอีกมากmany ให้เรียนรู้และค้นพบ ขอให้สบุกค้นการพัฒนาแอป และขอบคุณที่เข้าร่วมหลักสูตรนี้!

