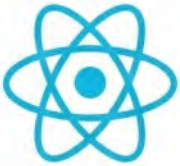
React.js Professional

Web Programming

(Integrative-Generative AI Edition)



Hooks

Deployment

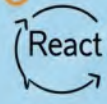
Components

Components

Hooks

Introduction to
JSX and Component
State and Event
Conditional Rendering

Forms in React
Forms in React
Lifecycle and UseEffect
Component Communication
Context API
API and Data Handling
Custom Hooks
Optimization
TypeScript
Testing
Real-World Projects
Professional Development
Practices



React Rooks
Advanced Form
Redux / Stesumd
Animation
Testing
Deployment

STUDENT PRICE
BOOK CENTER

คำนำ

ในยุคปัจจุบันที่แอปพลิเคชันบนเว็บต้องการประสิทธิภาพสูง ตอบสนองเร็ว และมีประสบการณ์ผู้ใช้ที่ลื่นไหล การพัฒนา Frontend ด้วย JavaScript จึงกลายเป็นหัวใจสำคัญของการพัฒนาระบบสมัยใหม่ โดยเฉพาะอย่างยิ่ง **React.js** ซึ่งเป็นไลบรารีที่ได้รับความนิยมอย่างแพร่หลายจากนักพัฒนาและองค์กรทั่วโลก ด้วยแนวคิด “Component-Based Architecture” และความสามารถในการจัดการ UI อย่างมีประสิทธิภาพ React ได้กลายเป็นเครื่องมือหลักสำหรับการสร้าง Single Page Applications (SPA) ที่ยืดหยุ่นและดูแลรักษาง่าย

หนังสือเล่มนี้ถูกออกแบบให้เป็น “คู่มือฉบับสมบูรณ์” สำหรับผู้ที่ต้องการเริ่มต้นจากศูนย์ จนสามารถพัฒนาแอปพลิเคชันระดับ Production ได้ด้วยตนเอง โดยครอบคลุมทั้งแนวคิดพื้นฐาน เช่น JSX, Component, State, Props และ Event Handling ไปจนถึงแนวทางระดับมืออาชีพ เช่นการใช้ Redux, Context API, การสร้าง Custom Hook, การจัดการฟอร์มขั้นสูง, การเชื่อมต่อ API, การทำ Animation, การใช้ TypeScript, การทดสอบระบบ และการ Deploy แอปขึ้น Server จริง

จุดเด่นของหนังสือเล่มนี้คือ การเรียบเรียงเนื้อหาอย่างเป็นลำดับขั้น มีตัวอย่างโค้ดและโครงสร้างโปรเจกต์แบบเต็ม พร้อมอธิบายเชิงลึกประกอบทุกบท รวมถึงการบูรณาการกับเครื่องมือยอดนิยมใน ecosystem ของ React เช่น React Router, React Hook Form, Framer Motion, Jest, และ Cypress เพื่อให้ผู้อ่านเข้าใจทั้งภาคทฤษฎีและภาคปฏิบัติอย่างแท้จริง

นอกจากนี้ ในบทท้ายของหนังสือยังได้นำเสนอโปรเจกต์จริงหลายรูปแบบ เช่น Todo List, Blog, Dashboard และ E-Commerce เพื่อให้ผู้อ่านได้ฝึกฝนการพัฒนาแบบครบวงจร พร้อมแนวทางการจัดโครงสร้างโปรเจกต์ระดับ Production และแนวคิด DevOps เบื้องต้นที่จำเป็นสำหรับการทำงานในองค์กรจริง

หวังเป็นอย่างยิ่งว่าหนังสือเล่มนี้จะเป็นคู่มือที่ทรงคุณค่าสำหรับทั้งนักเรียน นักศึกษา นักพัฒนา และผู้สนใจทั่วไป ที่ต้องการพัฒนาเว็บแอปพลิเคชันด้วย React อย่างมืออาชีพ และสามารถนำความรู้ไปประยุกต์ใช้จริงในโลกของการทำงานได้อย่างมั่นใจ

ด้วยความปรารถนาดี

ศูนย์หนังสือราคานักเรียน

สารบัญ

หน้า

บทที่ 1
ทำความรู้จักกับ React.js
(Introduction to React.js)

เนื้อหา

- ทำความรู้จักกับ React.js
- ประวัติของ React.js อย่างละเอียด
- การติดตั้ง React.js อย่างละเอียด
- วิธีใช้ NetBeans กับ React.js (ทางอ้อม)
- ขั้นตอนการติดตั้ง React.js บน Visual Studio Code (VS Code)

บทนำ: ทำความรู้จักกับ React.js

ในโลกของการพัฒนาเว็บแอปพลิเคชันยุคใหม่ ความสามารถในการสร้างอินเทอร์เฟซที่สวยงาม ตอบสนองเร็ว และดูแลรักษาได้ง่าย ได้กลายเป็นทักษะที่จำเป็นอย่างยิ่งสำหรับนักพัฒนาทุกคน และหนึ่งในเครื่องมือที่ได้รับความนิยมสูงสุดในการพัฒนา Frontend คือ **React.js** ซึ่งเป็น JavaScript Library ที่ถูกพัฒนาโดยทีมงานของ Facebook และเปิดให้ใช้งานแบบโอเพ่นซอร์สตั้งแต่ปี 2013 นับแต่นั้นมา React ก็เติบโตขึ้นอย่างรวดเร็วและกลายเป็นรากฐานสำคัญของเว็บแอปหลายล้านเว็บไซต์ทั่วโลก

React ถูกออกแบบมาเพื่อจัดการ **UI (User Interface)** โดยเฉพาะ ด้วยแนวคิดที่เรียกว่า **Component-Based Architecture** ซึ่งช่วยให้นักพัฒนาสามารถแบ่งหน้าจอออกเป็นส่วนย่อยๆ (Component) ที่นำกลับมาใช้ซ้ำได้ ทำให้โค้ดเป็นระบบมากขึ้น และลดความซับซ้อนในการดูแลระบบที่เติบโตอย่างรวดเร็ว ยิ่งไปกว่านั้น React ยังโดดเด่นในด้าน **Virtual DOM** ที่ช่วยเร่งประสิทธิภาพการแสดงผล โดยการคำนวณการเปลี่ยนแปลงก่อนอัปเดต DOM จริง ซึ่งส่งผลให้แอปพลิเคชันทำงานได้รวดเร็วและลื่นไหลกว่าการเขียน HTML + JS แบบดั้งเดิม

เมื่อเปรียบเทียบกับ JavaScript Framework อื่นๆ เช่น Angular หรือ Vue.js จะเห็นได้ว่า React มีแนวทางที่ “เรียบง่ายแต่ยืดหยุ่น” ผู้ใช้สามารถเลือกวิธีการจัดการ State, Routing หรือ API ได้ตามความเหมาะสม ไม่ได้ถูกบังคับให้ใช้ชุดเครื่องมือเฉพาะของ React เท่านั้น นี่จึงเป็นจุดแข็งที่ทำให้ React ได้รับความนิยมสูงสุดในหมู่ทีมพัฒนาองค์กรขนาดใหญ่ เช่น Netflix, Airbnb, Uber, และ Meta เอง รวมไปถึงนักพัฒนาอิสระทั่วโลก

หนึ่งในแนวคิดสำคัญที่ React ส่งเสริมคือการสร้าง **Single Page Application (SPA)** หรือเว็บไซต์ที่โหลดหน้าเพียงครั้งเดียว จากนั้นสามารถเปลี่ยนแปลงเนื้อหาโดยไม่ต้องรีเฟรชทั้งหน้าเว็บ

ช่วยให้ผู้ใช้ได้รับประสบการณ์ที่รวดเร็ว คล้ายกับแอปมือถือ SPA จึงเป็นมาตรฐานใหม่ของเว็บแอปยุคปัจจุบัน และ React ก็เป็นหนึ่งในเครื่องมือหลักที่ช่วยให้นักพัฒนาสร้าง SPA ได้ง่ายและมีประสิทธิภาพ

ในช่วงท้ายของบทนี้ ผู้อ่านจะได้เรียนรู้วิธี **ติดตั้ง React** อย่างรวดเร็วด้วยเครื่องมือยอดนิยม อย่าง **Vite** หรือ **Create React App** พร้อมทั้งทดลอง “รันโปรเจกต์แรก” ด้วยคำสั่งพื้นฐาน เพื่อให้เข้าใจโครงสร้างเบื้องต้นของ React App และเตรียมความพร้อมก่อนเข้าสู่บทเรียนเชิงลึกในบทถัดไป หนังสือเล่มนี้จะพาคุณก้าวเข้าสู่โลกของ React อย่างมั่นใจ ด้วยพื้นฐานแน่นและตัวอย่างที่ชัดเจนในทุกขั้นตอน

ทำความรู้จักกับ React.js

☐ 1.1 React คืออะไร? และมีที่มาอย่างไร

React คือไลบรารี (Library) ของ JavaScript ที่พัฒนาโดยบริษัท **Facebook** (ปัจจุบันคือ Meta) ซึ่งเปิดตัวในปี **2013** มีวัตถุประสงค์เพื่อพัฒนา **UI (User Interface)** ให้เร็วขึ้น และจัดการกับ **DOM (Document Object Model)** อย่างมีประสิทธิภาพ

จุดเริ่มต้น:

React ถูกสร้างขึ้นครั้งแรกโดยวิศวกรของ Facebook ชื่อ **Jordan Walke** เพื่อแก้ปัญหาการแสดงผลข้อมูลจำนวนมากในหน้าเว็บที่ซับซ้อน เช่น News Feed และ Messenger

จุดเด่น:

- เน้นการพัฒนาแบบ **Component-Based**
- มีแนวคิด **Declarative** ทำให้โค้ดอ่านง่ายและจัดการง่าย
- ใช้ **Virtual DOM** เพิ่มประสิทธิภาพการอัปเดตหน้าจอ

☐ 1.2 ความแตกต่างระหว่าง React กับ JS Framework อื่น ๆ

หัวข้อ	React	Angular	Vue
ประเภท	Library	Framework	Framework
ผู้พัฒนา	Meta (Facebook)	Google	Community
โครงสร้าง	ยืดหยุ่น, ต้องเลือกเอง	ครบเครื่อง (Opinionated)	ยืดหยุ่นแต่โครงสร้างชัดเจน
ขนาด	เล็ก	ใหญ่	เล็ก
การเรียนรู้	ปานกลาง	ยาก	ง่ายที่สุด
ใช้งานจริง	Facebook, Instagram, Netflix	Gmail, YouTube, Google	Alibaba, Xiaomi

สรุป:

React เหมาะกับผู้ที่ต้องการความยืดหยุ่นสูงและต้องการควบคุมเทคโนโลยีเอง (เช่น Router, State management) ในขณะที่ Angular และ Vue ให้เครื่องมือมาครบในตัว

☐ 1.3 SPA (Single Page Application) คืออะไร?

SPA ย่อมาจาก **Single Page Application** คือเว็บไซต์หรือแอปที่โหลดหน้าเว็บเพียงครั้งเดียว จากนั้นใช้ JavaScript ในการเปลี่ยนเนื้อหาแบบไดนามิกโดยไม่ต้องโหลดหน้าใหม่ทุกครั้ง

ข้อดี:

- ประสบการณ์ผู้ใช้รวดเร็ว ลื่นไหล
- ลดการโหลดข้อมูลซ้ำซ้อนจาก Server

ตัวอย่าง SPA:

- Gmail
- Facebook
- Trello

React เหมาะกับการสร้าง SPA อย่างมาก เพราะมันจัดการ **State** และ **View** ได้มีประสิทธิภาพ

☐ 1.4 ติดตั้ง React ด้วย Vite หรือ Create React App

☐ วิธีที่ 1: ด้วย Create React App (CRA)

```
npx create-react-app my-app
cd my-app
npm start
```

ข้อดี:

- เหมาะสำหรับผู้เริ่มต้น
- มีการตั้งค่าพื้นฐานให้ครบ

ข้อเสีย:

- โหลดช้ากว่า Vite
- ตั้งค่าลึก ๆ ยากกว่าหน่อย

☐ วิธีที่ 2: ด้วย Vite (เร็วกว่า CRA)

```
npm create vite@latest my-vite-app
cd my-vite-app
npm install
npm run dev
```

ข้อดี:

- โหลดเร็วกว่า CRA หลายเท่า
- รองรับ Hot Module Replacement (HMR)
- เหมาะสำหรับโปรเจกต์สมัยใหม่

1.5 การรันโปรเจกต์ React ครั้งแรก

เมื่อคุณติดตั้งโปรเจกต์เสร็จแล้ว (ไม่ว่าจะใช้ CRA หรือ Vite):

ขั้นตอน:

1. เข้าไปที่โฟลเดอร์โปรเจกต์
2. รันคำสั่ง
   - หากใช้ CRA: npm start
   - หากใช้ Vite: npm run dev
3. เปิดเว็บเบราว์เซอร์ แล้วเข้าไปที่ http://localhost:3000 (CRA) หรือ http://localhost:5173 (Vite)
4. คุณจะเห็นหน้าเริ่มต้นของ React แสดงข้อความเช่น "Welcome to React" หรือโลโก้ React หมุนอยู่

ประวัติของ React.js อย่างละเอียด

จุดเริ่มต้น: ก่อนการเกิด React

ในช่วงปลายทศวรรษ 2000 ถึงต้น 2010 การพัฒนาเว็บเริ่มซับซ้อนขึ้นมาก แอปพลิเคชันบนเว็บ เช่น Facebook และ Gmail จำเป็นต้องรองรับผู้ใช้นับล้าน และแสดงข้อมูลแบบ real-time

ในเวลานั้นนักพัฒนาใช้ **jQuery** และ **MVC frameworks** เช่น Backbone.js, AngularJS (เวอร์ชันแรก) ซึ่งเริ่มมีข้อจำกัดเรื่องการจัดการ state, DOM manipulation ที่ซับซ้อน และปัญหา performance

---

ค.ศ. 2011 — การกำเนิดภายใน Facebook

- **ผู้พัฒนา:** React ถูกคิดค้นขึ้นโดยวิศวกรของ Facebook ชื่อ **Jordan Walke**
- **แรงบันดาลใจ:** แนวคิดมาจาก **XHP** ซึ่งเป็นไลบรารีใน PHP ที่ Facebook ใช้แสดง UI แบบ declarative
- Facebook ต้องการเครื่องมือใหม่สำหรับจัดการ UI ที่เปลี่ยนแปลงบ่อย (dynamic UI) เช่น **News Feed** และ **Chat**
- React เริ่มถูกใช้ **ภายใน Facebook ในปี 2011** กับส่วน News Feed

---

ค.ศ. 2012 — ใช้ใน Instagram

- Instagram (ที่ Facebook ซื้อมาในปี 2012) กำลังอยู่ระหว่างการเขียนเว็บแอปเวอร์ชันใหม่
- ทีม React ได้ใช้โอกาสนี้สร้าง UI ของ Instagram ด้วย React เต็มรูปแบบ ซึ่งกลายเป็น "public beta" โดยไม่เปิดเผยชื่อ

☐ **ค.ศ. 2013 — React เปิดตัวสู่สาธารณะ**

- **งาน JSConf US 2013** เป็นเวทีที่ React ถูกเปิดตัวครั้งแรกต่อสาธารณชน
- หลายคนในชุมชนนักพัฒนา **วิจารณ์ React อย่างหนัก** เพราะ:
  - ใช้ **JSX** (HTML ใน JavaScript) ซึ่งขัดกับแนวคิด separation of concerns
  - แนวคิด "Virtual DOM" ยังใหม่และไม่เข้าใจง่ายในขณะนั้น
- อย่างไรก็ตาม React ค่อย ๆ ได้รับความนิยม เพราะ:
  - ประสิทธิภาพที่สูงขึ้น
  - รหัสที่ดูแลรักษาง่าย
  - รูปแบบ component ที่เข้าใจได้ในระยะยาว

☐ **ค.ศ. 2014–2016 — React Ecosystem เริ่มเติบโต**

- มีการสร้างเครื่องมือสนับสนุน:
  - **React Router** สำหรับจัดการ SPA Routing
  - **Redux** (พัฒนาโดย Dan Abramov) สำหรับการจัดการ state แบบ global
- React กลายเป็นตัวเลือกหลักในการพัฒนา SPA
- ถูกใช้โดยบริษัทใหญ่มากมาย: Netflix, Airbnb, WhatsApp Web

☐ **ค.ศ. 2017 — React 16: "Fiber Architecture"**

- เปิดตัว **React Fiber**: สถาปัตยกรรมใหม่ของ React สำหรับปรับปรุง performance และรองรับ async rendering
- เปิดทางให้ React รองรับ **concurrent UI**, animation ที่ลื่นไหล และ feature อนาคต

☐ **ค.ศ. 2018 — React Hooks เปลี่ยนเกม**

- เปิดตัว **React 16.8** และ **Hooks API** เช่น useState, useEffect
- นักพัฒนาไม่จำเป็นต้องใช้ **class component** อีกต่อไป
- เพิ่มความสามารถให้ functional component รองรับ state และ lifecycle อย่างเต็มรูปแบบ

☐ **ค.ศ. 2020–2021 — React 17–18: Concurrent Mode และ Server Components**

- React 17: ไม่มี feature ใหม่ แต่ปรับให้สามารถ upgrade React แบบค่อยเป็นค่อยไปได้
- React 18 (เปิดตัว 2022):
  - รองรับ **Concurrent Mode** (render แบบ async)
  - เพิ่ม **automatic batching**, **Suspense improvements**
  - รองรับ **React Server Components** (RSC) ซึ่งเป็นอนาคตของ Full-stack React

☐ **ค.ศ. 2022–ปัจจุบัน — React สู่ยุค Full-stack**

- การมาของ **Next.js 13+** ทำให้ React กลายเป็น **Full-stack Framework** ร่วมกับ Node.js
- ใช้งานคู่กับ:
  - **Server-side rendering (SSR)**
  - **Static site generation (SSG)**
  - **Edge function, API route**
- การมาของ **React Server Components (RSC)** ช่วยให้โหลดข้อมูลบนเซิร์ฟเวอร์ได้ก่อนส่งไปยัง client
- เครื่องมือที่เป็น ecosystem สำคัญ:
  - Vite, TailwindCSS, Zustand, TanStack Query, React Router, Next.js

---

☐ **ปัจจุบัน (2025): ทำไม React ยังเป็นที่นิยม?**

- **ใช้งานง่าย** และ **มีชุมชนใหญ่มาก**
- เครื่องมือเสริมพร้อมใช้งาน
- รองรับแนวคิด modern web: SSR, hydration, streaming, Suspense
- ใช้กับมือถือได้ผ่าน **React Native**

☐ **สรุป**

React ไม่ใช่แค่ไลบรารี UI ธรรมดาอีกต่อไป แต่กลายเป็นแกนหลักของการพัฒนาเว็บยุคใหม่ จากไลบรารีเล็ก ๆ ภายใน Facebook ⟶ สู่เครื่องมือหลักของแอปพลิเคชันระดับโลก

การติดตั้ง React.js อย่างละเอียด

☐ **1. เครื่องมือที่ต้องเตรียม**

ก่อนจะติดตั้ง React คุณต้องติดตั้งสิ่งเหล่านี้ในเครื่องก่อน:

☐ **1.1 ติดตั้ง Node.js และ npm**

- React ต้องการใช้ **Node.js** และ **npm (Node Package Manager)** ในการจัดการแพ็กเกจ

☐ ดาวน์โหลดได้ที่:

https://nodejs.org

แนะนำ: ติดตั้งเวอร์ชัน LTS (Long-Term Support)

จากนั้นเปิด Terminal หรือ Command Prompt แล้วตรวจสอบว่า Node กับ npm ติดตั้งแล้วหรือไม่:

```
node -v
```

npm -v

□ 2. เลือกเครื่องมือสร้างโปรเจกต์ React

คุณสามารถใช้ได้ 2 วิธีหลัก:

- **Create React App (CRA)** – เหมาะสำหรับผู้เริ่มต้น
- **Vite** – เหมาะสำหรับผู้ที่ต้องการความเร็วในการพัฒนา

□ วิธีที่ 1: ติดตั้งด้วย Create React App (CRA)

□ CRA คืออะไร?

create-react-app เป็นเครื่องมือที่ช่วยตั้งค่าโปรเจกต์ React อัตโนมัติ โดยมีโครงสร้างพื้นฐานครบถ้วน พร้อมใช้งานทันที

□ ขั้นตอนการติดตั้ง CRA

```
npx create-react-app my-app
cd my-app
npm start
```

□ อธิบายแต่ละคำสั่ง:

- npx คือคำสั่งรันแพ็กเกจชั่วคราวจาก npm (ไม่ต้องติดตั้งถาวร)
- create-react-app my-app สร้างโฟลเดอร์ชื่อ my-app พร้อมไฟล์และโครงสร้าง React
- cd my-app เข้าไปในโฟลเดอร์โปรเจกต์
- npm start สั่งให้รันเซิร์ฟเวอร์เพื่อแสดงเว็บบน http://localhost:3000

□ วิธีที่ 2: ติดตั้งด้วย Vite (เร็วกว่า CRA มาก)

□ Vite คืออะไร?

Vite เป็นเครื่องมือสร้างและพัฒนาโปรเจกต์ที่เร็วมาก และรองรับ React, Vue, Svelte, ฯลฯ โดยใช้ **ES Modules** และ **Hot Module Replacement (HMR)**

□ ขั้นตอนการติดตั้ง Vite

```
npm create vite@latest my-vite-app
cd my-vite-app
npm install
npm run dev
```

□ รายละเอียด:

- npm create vite@latest my-vite-app สร้างโฟลเดอร์ my-vite-app พร้อมเลือก template (เลือก React)
- ระบบจะถามให้เลือก framework: ให้ใช้ปุ่มลูกศรเลือก React ⟶ แล้วกด Enter

- เลือก variant เป็น JavaScript หรือ TypeScript
- npm install ติดตั้งแพ็กเกจทั้งหมด
- npm run dev รันเว็บด้วย Vite ที่ http://localhost:5173

☐ เปรียบเทียบ CRA vs Vite

รายการเปรียบเทียบ	CRA	Vite
ความเร็ว	ช้า (webpack-based)	เร็วมาก
เทคโนโลยี	Webpack	ESBuild + Native ESM
รองรับ HMR	มี แต่ช้ากว่า	มีและเร็ว
เหมาะกับ	ผู้เริ่มต้น	ผู้ต้องการ performance สูง
การตั้งค่า	อัตโนมัติ	ยืดหยุ่นมากกว่า

☐ โครงสร้างโฟลเดอร์ของ React Project (ทั้ง Vite และ CRA)

ตัวอย่าง (สำหรับ CRA):

```
my-app/
├── node_modules/
├── public/
│   └── index.html
├── src/
│   ├── App.js
│   └── index.js
├── package.json
└── README.md
```

- src/ คือที่เก็บ component และโค้ดหลัก
- App.js คือ Component หลัก
- index.js คือจุดเริ่มของโปรเจกต์ (entry point)

☐ ทดสอบการติดตั้ง

หลังจากรัน npm start หรือ npm run dev แล้ว:

- เปิดเบราว์เซอร์ที่ http://localhost:3000 (CRA) หรือ http://localhost:5173 (Vite)
- คุณจะเห็นหน้าเว็บเริ่มต้นของ React

เช่น:

Welcome to React

Edit src/App.js and save to reload.

☐ **สรุป**

ขั้นตอน	รายละเอียด
ติดตั้ง Node.js	ใช้ nodejs.org
ใช้ CRA หรือ Vite	CRA สำหรับเริ่มต้น, Vite สำหรับประสิทธิภาพ
คำสั่งเริ่มต้น	npx create-react-app หรือ npm create vite@latest
รันโปรเจกต์	npm start (CRA) หรือ npm run dev (Vite)
พัฒนาเพิ่มเติม	เขียน Component เพิ่มใน src/

วิธีใช้ NetBeans กับ React.js (ทางอ้อม)

☐ **วิธีที่ 1: ใช้ NetBeans ร่วมกับ Terminal (หรือ Git Bash, CMD)**

1. **ติดตั้ง Node.js และ npm**

   ⟶ จาก https://nodejs.org
2. **เปิด Terminal / Git Bash จาก NetBeans**
   - ไปที่เมนู Window > IDE Tools > Terminal
   - หรือใช้ Terminal ภายนอกก็ได้
3. **สร้างโปรเจกต์ React ด้วยคำสั่ง**
4. npx create-react-app my-app
5. cd my-app
6. npm start
7. **เปิดโปรเจกต์ใน NetBeans**:
   - ไปที่ File > Open Project
   - เลือกโฟลเดอร์ my-app
   - NetBeans จะเปิดโฟลเดอร์และให้คุณแก้ไขไฟล์ .js, .jsx, .css ได้ทันที

☐ **ข้อควรทราบ:**

- NetBeans จะไม่สามารถรันหรือ preview React ให้คุณได้โดยตรงเหมือน Visual Studio Code
- แต่คุณสามารถรัน npm start ใน Terminal เพื่อดูผลลัพธ์ในเบราว์เซอร์ได้

☐ **วิธีที่แนะนำ: ใช้ Visual Studio Code (VS Code) แทน**

คุณสมบัติ	NetBeans	VS Code
รองรับ JSX	☐ ไม่สมบูรณ์	☐ ดีเยี่ยม
IntelliSense	น้อยมาก	สูงมาก (ผ่าน Extensions)
Terminal ในตัว	☐ มี	☐ มี
Debug React	☐ ไม่รองรับโดยตรง	☐ รองรับผ่าน Chrome DevTools
Extensions React	☐ ไม่มีเฉพาะทาง	☐ มีมากมาย

☐ หากต้องการประสบการณ์พัฒนา React ที่ราบรื่น ผมแนะนำให้ใช้ **VS Code** มากกว่า

สรุป

ประเด็น	คำตอบ
ติดตั้ง React บน NetBeans โดยตรงได้ไหม	☐ ไม่ได้โดยตรง
พัฒนา React ผ่าน NetBeans ได้ไหม	☐ ได้ ถ้าใช้ Terminal และเปิดไฟล์ด้วย NetBeans
ทางเลือกที่ดีกว่า	☐ ใช้ **Visual Studio Code** หรือ **WebStorm**

ขั้นตอนการติดตั้ง React.js บน Visual Studio Code (VS Code)

☐ **ก่อนเริ่ม: เตรียมเครื่องมือ**

เครื่องมือ	คำอธิบาย	ดาวน์โหลด
☐ Node.js + npm	สำหรับจัดการแพ็กเกจและรัน React	https://nodejs.org
☐ Visual Studio Code	ตัวแก้ไขโค้ด (IDE) ที่รองรับ React ดีเยี่ยม	https://code.visualstudio.com
☐ VS Code Extensions	ส่วนเสริมเช่น ESLint, Prettier, React snippets	(ติดตั้งภายใน VS Code)

⚙☐ **ขั้นตอนที่ 1: ติดตั้ง Node.js**

1. เข้าเว็บไซต์ https://nodejs.org
2. ดาวน์โหลดเวอร์ชัน LTS (แนะนำ)
3. ติดตั้งตามขั้นตอน
4. ตรวจสอบว่าใช้งานได้:
5. node -v
6. npm -v

☐ ขั้นตอนที่ 2: สร้าง React Project ด้วย Vite (แนะนำ)

☐ ทำได้จาก **VS Code Terminal** หรือ Command Prompt ภายนอกก็ได้

1. เปิด VS Code ⟶ กด Ctrl + ~ เพื่อเปิด Terminal
2. สร้างโปรเจกต์ด้วย Vite:
3. npm create vite@latest my-react-app
4. ระบบจะถาม:
    - Project name: my-react-app
    - Framework: เลือก React
    - Variant: เลือก JavaScript หรือ TypeScript
5. ติดตั้ง dependencies:
6. cd my-react-app
7. npm install

☐ ขั้นตอนที่ 3: เปิดโครงการใน VS Code

1. ไปที่เมนู File > Open Folder
2. เลือกโฟลเดอร์ my-react-app
3. โครงสร้างโฟลเดอร์จะมีลักษณะดังนี้:

```
my-react-app/
├── index.html
├── src/
│   ├── App.jsx
│   └── main.jsx
├── vite.config.js
└── package.json
```

☐ ขั้นตอนที่ 4: รันและทดสอบโปรเจกต์

ใน Terminal:

npm run dev

- ระบบจะขึ้นลิงก์ เช่น http://localhost:5173
- เปิดลิงก์ในเบราว์เซอร์
- หากทุกอย่างถูกต้อง คุณจะเห็นหน้า "Vite + React" หรือ "Hello World"

✍ □ ขั้นตอนที่ 5: แก้ไขไฟล์และดูผลลัพธ์

1. เปิด src/App.jsx
2. แก้ไขข้อความเช่น:
3. function App() {
4. return <h1>สวัสดี React บน VS Code!</h1>;
5. }
6. บันทึกไฟล์ (Ctrl + S) ⟶ เบราว์เซอร์จะรีโหลดอัตโนมัติ

□ ขั้นตอนที่ 6: ติดตั้งส่วนเสริม (Extensions ที่แนะนำ)

1. เปิดแท็บ Extensions (Ctrl + Shift + X)
2. ติดตั้ง:
   - □ **ESLint** – ตรวจสอบโค้ด JavaScript
   - □ **Prettier - Code formatter** – จัดรูปแบบโค้ด
   - □ **Reactjs code snippets** – ช่วยพิมพ์ React component ได้ไวขึ้น

□ สรุป: ขั้นตอนสำคัญ

ขั้นตอน	รายละเอียด
1. ติดตั้ง Node.js	ใช้ npm สร้างโปรเจกต์
2. ติดตั้ง VS Code	ใช้เขียนและจัดการโค้ด
3. ใช้ Vite สร้าง React App	npm create vite@latest
4. รันโปรเจกต์	npm run dev
5. ทดสอบผ่านเบราว์เซอร์	ที่ http://localhost:5173
6. ติดตั้ง Extension เสริม	เพิ่มประสิทธิภาพในการเขียนโค้ด

หากคุณต้องการ:

- □ โค้ดตัวอย่าง Component React
- □ ไฟล์ starter zip
- □ ภาพหน้าจอประกอบทุกขั้นตอน

สรุป

React.js คือไลบรารี JavaScript ยอดนิยมสำหรับการพัฒนาอินเทอร์เฟซของเว็บแอปพลิเคชันแบบ Single Page Application (SPA) ที่เน้นความเร็ว ลื่นไหล และดูแลรักษาง่าย โดยมีจุดเด่นที่การ

ออกแบบเชิง Component ซึ่งช่วยให้การพัฒนาเป็นระบบและสามารถนำกลับมาใช้ซ้ำได้ บทนี้จะพาผู้อ่านทำความเข้าใจที่มาของ React ความแตกต่างระหว่าง React กับเฟรมเวิร์กอื่น เช่น Angular หรือ Vue.js แนวคิดของ SPA และปิดท้ายด้วยการติดตั้ง React ด้วยเครื่องมืออย่าง Vite หรือ Create React App พร้อมทดลองรันโปรเจกต์แรก เพื่อปูพื้นฐานสู่การเรียนรู้ React อย่างมั่นใจในบทถัดไป.

บทที่ 2
พื้นฐาน JSX และการสร้าง Component
(JSX and Component)

เนื้อหา

บทนำ: พื้นฐาน JSX และการสร้าง Component

เมื่อพูดถึง React สิ่งหนึ่งที่มักสร้างความประทับใจและความสงสัยแก่ผู้เริ่มต้นคือการที่เราสามารถ “เขียน HTML ภายใน JavaScript” ได้อย่างเป็นธรรมชาติ ซึ่งนั่นก็คือ **JSX (JavaScript XML)** — ภาษากึ่ง HTML ที่ทำให้เราสามารถเขียนโค้ด UI ได้ง่าย อ่านออก และใกล้เคียงกับโครงสร้างหน้าเว็บจริงๆ JSX ไม่ใช่ภาษาพิเศษ แต่เป็น Syntax ที่ถูกแปลงเป็น JavaScript โดยเครื่องมือของ React ซึ่งทำให้เราสามารถแสดงข้อความ ตัวแปร หรือแม้แต่เขียน expression แบบ dynamic ได้ภายในแท็กต่างๆ อย่างมีประสิทธิภาพ

บทนี้จะพาผู้อ่านเจาะลึกพื้นฐานของ JSX ทั้งในแง่แนวคิดและการใช้งานจริง ตั้งแต่การแสดงข้อความ, การผูกค่ากับตัวแปร, การเขียนเงื่อนไขภายใน JSX, ไปจนถึงการฝังฟังก์ชันหรือค่าที่คำนวณแบบ expression ภายใน UI เพื่อสร้างส่วนติดต่อผู้ใช้ที่ตอบสนองตามข้อมูลหรือสถานะของระบบ

นอกจากนี้ ผู้อ่านจะได้เรียนรู้การสร้าง **Functional Component** ซึ่งเป็นหัวใจของการเขียน React ยุคใหม่ Component เหล่านี้ทำหน้าที่เหมือนฟังก์ชันที่รับข้อมูลเข้าผ่าน props และแสดงผล UI

ตามนั้น เราจะเห็นวิธีการส่งข้อมูลจากภายนอกมายัง Component ด้วย props การใช้ children เพื่อแทรกองค์ประกอบภายใน และแนวทางการเขียน Component ที่สามารถนำกลับมาใช้ซ้ำได้ในหลายบริบท

พื้นฐานของ JSX และการสร้าง Component คือรากฐานสำคัญที่นักพัฒนา React ทุกคนต้องเข้าใจอย่างลึกซึ้ง เพราะทุกหน้าเว็บใน React ต่างประกอบขึ้นจาก Component ที่ถูกเขียนด้วย JSX ทั้งสิ้น บทนี้จึงเป็นจุดเริ่มต้นสำคัญในการก้าวสู่การพัฒนา React อย่างเต็มรูปแบบ ที่ทั้งยืดหยุ่น ทรงพลัง และพร้อมต่อยอดในระบบขนาดใหญ่ได้อย่างมั่นใจ

พื้นฐาน JSX และการสร้าง Component

☐ 1. JSX คืออะไร และทำไมถึงจำเป็นใน React

☐ **JSX (JavaScript XML) คืออะไร?**

JSX คือไวยากรณ์พิเศษที่ React ใช้เพื่อเขียน HTML-like ใน JavaScript

แทนที่จะเขียน UI ด้วยคำสั่ง DOM แบบเดิม เราสามารถเขียนโค้ดลักษณะนี้ได้:

```
const element = <h1>สวัสดี JSX</h1>;
```

☐ **ทำไมต้องใช้ JSX?**

ข้อดีของ JSX	คำอธิบาย
☐ อ่านง่าย	ดูคล้าย HTML
☐ เขียนแบบ declarative	บอก "ต้องการอะไร" มากกว่าคำสั่ง imperatively
☐ แทรก JavaScript ได้	ใช้ {} เพื่อแทรกค่าหรือ expression
☐ ทำงานร่วมกับ Babel	Babel แปลง JSX ⟶ JavaScript ⟶ React.createElement()

☐ 2. การใช้ JSX แสดงผลข้อความ ตัวแปร และ Expression

☐ **แสดงข้อความทั่วไป:**

```
function App() {
  return <h1>สวัสดี React</h1>;
}
```

☐ **แสดงค่าจากตัวแปร:**

```
const name = "สมชาย";
function App() {
  return <h2>ยินดีต้อนรับ, {name}</h2>;
}
```

☐ **แสดง Expression:**

```
function App() {
  const a = 10;
  const b = 5;
  return <p>ผลรวมคือ: {a + b}</p>;
}
```

☐ **ข้อควรระวัง:**

- JSX ต้องมี return เพียงหนึ่ง tag เท่านั้น
  (หากมีหลาย tag ต้องห่อด้วย div หรือ <>...</>)

```
// ถูกต้อง
return (
  <>
    <h1>หัวข้อ</h1>
    <p>เนื้อหา</p>
  </>
);
```

☐ **3. การสร้าง Functional Component**

☐ **Functional Component คืออะไร?**

Functional Component คือฟังก์ชัน JavaScript ที่ return JSX
มักใช้เขียน UI แบบโมดูล (component-based design)

☐ **ตัวอย่าง:**

```
function Welcome() {
  return <h1>ยินดีต้อนรับสู่ React!</h1>;
}
```

☐ **เรียกใช้ Component:**

```
function App() {
  return (
    <div>
      <Welcome />
    </div>
  );
}
```

☐ ชื่อ Component ต้องขึ้นต้นด้วยตัว **ใหญ่ (Capital Letter)** เช่น Welcome, MyComponent

□ 4. การใช้ props และ children เพื่อส่งข้อมูลระหว่าง Components

□ props คืออะไร?

props ย่อมาจาก **"properties"** คือค่าที่ส่งจาก **Parent Component ⟶ Child Component**

□ ตัวอย่างการส่ง props:

```
function Greeting(props) {
  return <h1>สวัสดี {props.name}</h1>;
}

function App() {
  return <Greeting name="สมใจ" />;
}
```

ผลลัพธ์: □ สวัสดี สมใจ

□ การใช้ children

children คือค่าที่อยู่ระหว่าง tag เปิด-ปิดของ Component

```
function Wrapper(props) {
  return (
    <div className="box">
      <h2>หัวข้อ</h2>
      {props.children}
    </div>
  );
}

function App() {
  return (
    <Wrapper>
      <p>นี่คือเนื้อหาภายใน Wrapper</p>
    </Wrapper>
  );
}
```

ผลลัพธ์:

```
<div class="box">
```

```
  <h2>หัวข้อ</h2>
  <p>นี่คือเนื้อหาภายใน Wrapper</p>
</div>
```

✏ **สรุปเนื้อหา**

หัวข้อ	สรุป
JSX คืออะไร	Syntax ที่ใช้เขียน UI คล้าย HTML ใน JavaScript
แสดงค่าด้วย JSX	ใช้ {} เพื่อแทรกค่าหรือตัวแปร
Functional Component	ฟังก์ชันที่ return JSX เพื่อใช้เป็น UI Component
props	ใช้ส่งข้อมูลจาก Component แม่ไปยังลูก
children	เนื้อหาที่อยู่ภายใน tag ของ Component

พื้นฐาน JSX และการสร้าง Component (เชิงลึก)

---

1. JSX คืออะไร และทำไมถึงจำเป็นใน React

ความหมายของ JSX

JSX (JavaScript XML) คือ syntax ที่อนุญาตให้เราเขียน HTML-like code ภายใน JavaScript

ตัวอย่าง JSX:

```
const element = <h1>Hello, JSX</h1>;
```

เบื้องหลัง JSX นี้จะถูก **Babel** แปลงเป็นคำสั่ง JavaScript ธรรมดาแบบนี้:

```
const element = React.createElement('h1', null, 'Hello, JSX');
```

ทำไม React ต้องใช้ JSX?

1. **เขียน UI แบบ declarative** (บอกว่าหน้าตาควรเป็นอย่างไร)
2. **ใกล้เคียง HTML** ทำให้เรียนรู้เร็ว
3. **ผสม JavaScript + HTML** ได้อย่างยืดหยุ่น (logic + layout)
4. **สนับสนุนโครงสร้าง Component อย่างชัดเจน**

⚠ **ความเข้าใจผิดที่พบบ่อย:**

ความเชื่อผิด	ความจริง
JSX คือ HTML	JSX ไม่ใช่ HTML แต่มีหน้าตาเหมือน

ความเชื่อผิด	ความจริง
JSX ทำงานในเบราว์เซอร์	❌ JSX ต้องถูก compile (ผ่าน Babel) ก่อนจึงทำงานได้
JSX ใช้ tag ปิดเองได้เสมอ	❌ ใช้ได้ แต่ต้องปิดให้ถูก เช่น <img />, <input />

---

2. ❌ การใช้ JSX แสดงผลข้อความ ตัวแปร และ Expression

ใน JSX เราสามารถฝังค่าหรือ Expression ได้ด้วย {} เช่น:

```
function App() {
  const name = "สมชาย";
  const year = new Date().getFullYear();
  return <p>ชื่อ: {name}, ปี: {year}</p>;
}
```

ประเภทที่สามารถใช้ใน {}	ตัวอย่าง
ตัวแปร	{username}
ค่าคงที่ (constant)	{10 + 20}
ฟังก์ชัน	{getUserName()}
การดำเนินการ logic	{isLoggedIn ? "ออกจากระบบ" : "เข้าสู่ระบบ"}

❌ JSX ไม่ใช่ template literal

ต่างจาก ${} ใน template string เช่น:

```
// แบบ JavaScript ปกติ
console.log(`สวัสดี ${name}`);
```

แต่ใน JSX ต้องใช้ {} โดยตรงใน return:

```
return <p>สวัสดี {name}</p>;
```

---

3. ❌ การสร้าง Functional Component

❌ นิยาม

Functional Component คือ JavaScript function ที่ **return JSX**

ซึ่ง React จะนำไป render เป็น UI จริง

```
function Welcome() {
  return <h1>ยินดีต้อนรับ!</h1>;
}
```

หรือแบบ arrow function:

```
const Welcome = () => <h1>ยินดีต้อนรับ!</h1>;
```

กฎการตั้งชื่อ:

- ขึ้นต้นด้วย **ตัวใหญ่** เช่น MyComponent
- ใช้ได้ทั้งแบบ function หรือ const

⚠ **JSX ต้อง return node เดียวเท่านั้น**

ไม่สามารถเขียน return หลาย tag ได้โดยไม่ห่อ เช่น:

```
// ผิด
return <h1>Hi</h1><p>Welcome</p>

// ถูก (ใช้ fragment)
return (
  <>
    <h1>Hi</h1>
    <p>Welcome</p>
  </>
)
```

---

4. การใช้ props และ children เพื่อส่งข้อมูลระหว่าง Components

props คืออะไร?

props (short for "properties") คือ **อาร์กิวเมนต์ที่ส่งจาก parent ⟶ child**

ใช้สำหรับการส่งข้อมูลแบบ read-only

```
function Welcome(props) {
  return <h2>ยินดีต้อนรับ {props.name}!</h2>;
}

function App() {
  return <Welcome name="สมใจ" />;
}
```

โครงสร้าง props แบบ object:

```
props = {
  name: "สมใจ",
  age: 20
}
```

☐ **การใช้ children**

props.children คือค่าที่อยู่ **ระหว่าง tag เปิด-ปิดของ Component**

```
function Layout(props) {
  return (
    <div className="box">
      <h3>หัวข้อ:</h3>
      {props.children}
    </div>
  );
}

function App() {
  return (
    <Layout>
      <p>นี่คือเนื้อหาภายใน Layout</p>
    </Layout>
  );
}
```

☐ **เคล็ดลับการเขียน Component:**

แนวทาง	คำอธิบาย
☐ แยก Component เป็นโมดูล	ทำให้โค้ด reusable
☐ ใช้ props แทนการ hardcode	เพื่อให้ component ยืดหยุ่น
☐ ใช้ default props และ prop types (ในโปรเจกต์ใหญ่)	ป้องกันการลืมส่งค่า
☐ ไม่แก้ไขค่า props โดยตรง	props ควร read-only เท่านั้น

☐ **สรุปภาพรวม**

หัวข้อ	สรุปเนื้อหา
JSX	ไวยากรณ์ผสม HTML + JavaScript
Expression	ใช้ {} เพื่อแสดงค่าภายใน JSX
Functional Component	ฟังก์ชันที่ return JSX (UI Module)

หัวข้อ	สรุปเนื้อหา
props	ส่งข้อมูลจาก component แม่ไปลูก
children	ข้อมูลที่ห่อหุ้มภายใน component

☐ **ตัวอย่างรวม**

```
function Card(props) {
  return (
    <div className="card">
      <h2>{props.title}</h2>
      <p>{props.content}</p>
      {props.children}
    </div>
  );
}

function App() {
  return (
    <Card title="React คืออะไร?" content="JS Library สำหรับ UI">
      <small>ข้อมูลจาก Facebook</small>
    </Card>
  );
}
```

JSX คืออะไร?

JSX (JavaScript XML) คือไวยากรณ์พิเศษที่ใช้ใน React ซึ่งทำให้เราสามารถเขียนโค้ดที่ "ดูเหมือน HTML" ภายใน JavaScript ได้

แม้ JSX จะดูคล้าย HTML มาก แต่จริง ๆ แล้ว JSX ไม่ใช่ HTML — มันเป็น syntax ที่ถูก **แปลงเป็น JavaScript** โดยเครื่องมือที่ชื่อว่า **Babel**

☐ ***ตัวอย่าง JSX:***

```
const element = <h1>สวัสดี JSX!</h1>;
```

เมื่อคอมไพล์ JSX แล้ว Babel จะแปลงเป็น JavaScript ธรรมดาแบบนี้:

```
const element = React.createElement('h1', null, 'สวัสดี JSX!');
```

☐ ทำไม JSX จึงจำเป็นใน React?

JSX ไม่ใช่สิ่งที่ "จำเป็นทางเทคนิค" ต่อ React เพราะเราสามารถเขียน React โดยไม่ใช้ JSX ได้เลย แต่ JSX เป็นสิ่งที่ **"จำเป็นทางความสะดวกและประสิทธิภาพในการพัฒนา"** เพราะมันช่วยให้:

จุดเด่นของ JSX	คำอธิบาย
☐ อ่านง่าย	JSX ดูคล้าย HTML ทำให้นักพัฒนาคุ้นเคย
☐ รวม logic กับ UI	เขียน JavaScript และ markup อยู่ร่วมกันอย่างกลมกลืน
☐ แสดงข้อมูลแบบ dynamic	ใช้ {} เพื่อแสดงค่าตัวแปรหรือ expression ได้
☐ ทำงานกับเครื่องมือ React ecosystem	JSX เป็นส่วนกลางของ React Workflow เช่นกับ Babel, TypeScript
☐ ส่งเสริมแนวคิด Component-based	สามารถออกแบบ UI เป็นโมดูลแบบแยกส่วนได้สะดวก

☐ JSX vs HTML (ข้อแตกต่าง)

JSX	HTML
ต้องใช้ className แทน class	ใช้ class ตามปกติ
ต้องใช้ tag ปิดตัวเองเสมอ เช่น <img />	บาง tag ไม่ต้องปิดก็ได้
ฝังค่าด้วย {}	ไม่มีการฝัง JS ใน HTML
อยู่ใน JavaScript	อยู่ในไฟล์ .html โดยตรง

☐ ตัวอย่างการใช้ JSX (พื้นฐาน)

```
const name = "สมชาย";
const element = <h1>สวัสดี {name}</h1>;
```

- {name} คือการฝังค่า JavaScript ลงใน JSX
- JSX ต้องมี tag หลักเพียงหนึ่งอัน ถ้าจะใช้หลาย tag ต้องห่อด้วย หรือ <>...</>

```
return (
  <>
    <h1>สวัสดี</h1>
    <p>ยินดีต้อนรับ</p>
  </>
);
```

☐ สรุป

JSX คือ syntax ที่ผสมผสาน JavaScript + HTML เพื่อสร้าง UI ได้ง่ายและยืดหยุ่นมากขึ้นใน React

สาระสำคัญ	รายละเอียด
JSX = JavaScript XML	เขียน UI ที่อ่านง่ายเหมือน HTML
ทำงานร่วมกับ Babel	ถูกแปลงเป็น React.createElement
ช่วยให้ UI logic อยู่รวมกับ layout	เขียนโค้ดได้เป็นธรรมชาติและ modular
ไม่บังคับใช้ แต่ “ควรใช้”	เพื่อเพิ่ม productivity และ maintainability

ตัวอย่างโปรแกรม **React แบบเต็มไฟล์** ที่ใช้ **JSX** อย่างครบถ้วน พร้อมคำอธิบายทีละส่วน และอธิบายผลลัพธ์ที่ได้หลังรันในเบราว์เซอร์

โค้ดตัวอย่าง: App.jsx

```
// ไฟล์: App.jsx
import React from 'react';

// สร้างฟังก์ชันแสดงชื่อและปีปัจจุบัน
function Greeting(props) {
  const currentYear = new Date().getFullYear();
  return (
    <div>
      <h2>สวัสดีคุณ {props.name}</h2>
      <p>ปีปัจจุบันคือ {currentYear}</p>
    </div>
  );
}

// ฟังก์ชันหลักของแอปพลิเคชัน
function App() {
  const name = "สมชาย";
  const status = true;

  return (
    <>
      <h1>ยินดีต้อนรับเข้าสู่ React App</h1>
```

```
      {/* ใช้ JSX ฝังตัวแปรและ expression */}
      <p>ผู้ใช้: {name}</p>
      <p>สถานะการเข้าสู่ระบบ: {status ? "ออนไลน์" : "ออฟไลน์"}</p>

      {/* เรียกใช้ Component อื่น */}
      <Greeting name={name} />
    </>
  );
}

export default App;
```

□ โครงสร้างโปรเจกต์พื้นฐาน

```
my-react-app/
├── public/
│   └── index.html
├── src/
│   ├── App.jsx        ← □ ไฟล์นี้
│   └── main.jsx       ← จุดเริ่มต้นของ React
├── package.json
└── vite.config.js (ถ้าใช้ Vite)
```

□ main.jsx

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';

// จุดเริ่มต้นของโปรเจกต์
ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

---

public/index.html

```
<!DOCTYPE html>
<html lang="th">
  <head>
    <meta charset="UTF-8" />
    <title>React JSX Example</title>
  </head>
  <body>
    <div id="root"></div> <!-- React จะ render ตรงนี้ -->
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

▶ วิธีรันโปรแกรม (กรณีใช้ Vite)

```
npm create vite@latest my-react-app -- --template react
cd my-react-app
npm install
npm run dev
```

ผลลัพธ์ที่แสดงในเบราว์เซอร์

ยินดีต้อนรับเข้าสู่ React App
ผู้ใช้: สมชาย
สถานะการเข้าสู่ระบบ: ออนไลน์
สวัสดีคุณ สมชาย
ปีปัจจุบันคือ 2025

คำอธิบายโค้ด

ส่วน	รายละเอียด
<App />	Component หลักของแอป
const name = "สมชาย"	กำหนดค่าตัวแปรที่ใช้แสดงผล
{status ? "ออนไลน์" : "ออฟไลน์"}	Expression ที่ใช้ ternary operator
<Greeting name={name} />	การส่ง props ไปยัง Component อื่น
props.name	ใช้แสดงค่าจาก Parent Component

☐ **จุดเด่นที่ได้เรียนรู้จากตัวอย่างนี้**

- ใช้ **JSX** เพื่อเขียน HTML ภายใน JavaScript ได้อย่างสวยงาม
- ใช้ **Expression ใน JSX** เช่น ternary operator
- สร้าง **Component แยกออกจากกัน** และส่งข้อมูลด้วย props
- ใช้ Fragment (<>...</>) เพื่อห่อหลายองค์ประกอบโดยไม่ต้องใช้

โครงสร้างโปรเจกต์

ตัวอย่างโปรเจกต์ React: โปรแกรมแสดงรายการงาน (Todo List แบบง่าย)

1. โครงสร้างโปรเจกต์ (แบบเรียบง่าย)

```
my-todo-app/
├── public/
│   └── index.html
├── src/
│   ├── App.jsx          ← Component หลัก
│   ├── main.jsx         ← จุดเริ่มต้นของ React
│   └── TodoItem.jsx     ← Component แยกย่อยสำหรับแสดงงานแต่ละรายการ
├── package.json
└── vite.config.js (ถ้าใช้ Vite)
```

2. โค้ดไฟล์ทั้งหมด

2.1 public/index.html

```
<!DOCTYPE html>
<html lang="th">
<head>
  <meta charset="UTF-8" />
  <title>Todo List React App</title>
</head>
<body>
  <div id="root"></div>  <!-- React จะ render ตรงนี้ -->
```

```
  <script type="module" src="/src/main.jsx"></script>
</body>
</html>
```

2.2 src/main.jsx

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App.jsx';

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

2.3 src/TodoItem.jsx

```
import React from 'react';

function TodoItem({ todo }) {
  return (
    <li style={{ padding: '8px 0' }}>
      {todo.text}
    </li>
  );
}

export default TodoItem;
```

2.4 src/App.jsx

```
import React, { useState } from 'react';
import TodoItem from './TodoItem.jsx';

function App() {
  // สถานะรายการ todo (เริ่มต้นเป็น 3 รายการ)
```

```
const [todos, setTodos] = useState([
  { id: 1, text: 'ทำงานบ้าน' },
  { id: 2, text: 'อ่านหนังสือ React' },
  { id: 3, text: 'ออกกำลังกาย' }
]);

const [inputValue, setInputValue] = useState('');

// ฟังก์ชันเพิ่ม todo ใหม่
const addTodo = () => {
  if (inputValue.trim() === '') return;
  const newTodo = {
    id: todos.length + 1,
    text: inputValue.trim()
  };
  setTodos([...todos, newTodo]);
  setInputValue('');
};

// จัดการเมื่อพิมพ์ในช่อง input
const handleInputChange = (e) => {
  setInputValue(e.target.value);
};

// จัดการกด Enter เพื่อเพิ่ม todo
const handleKeyPress = (e) => {
  if (e.key === 'Enter') {
    addTodo();
  }
};

return (
  <div style={{ maxWidth: 400, margin: '30px auto', fontFamily: 'Arial, sans-serif' }}>
    <h1>รายการงาน (Todo List)</h1>
```

```
      <input
        type="text"
        placeholder="พิมพ์รายการงานใหม่..."
        value={inputValue}
        onChange={handleInputChange}
        onKeyDown={handleKeyPress}
        style={{ width: '100%', padding: 8, boxSizing: 'border-box' }}
      />
      <button onClick={addTodo} style={{ marginTop: 10, padding: '8px 16px' }}>
        เพิ่มงาน
      </button>

      <ul style={{ marginTop: 20, paddingLeft: 20 }}>
        {todos.map(todo => (
          <TodoItem key={todo.id} todo={todo} />
        ))}
      </ul>
    </div>
  );
}

export default App;
```

---

3. วิธีติดตั้งและรันโปรเจกต์

1. สร้างโปรเจกต์ React ด้วย Vite (ถ้ายังไม่มี)

```
npm create vite@latest my-todo-app -- --template react
cd my-todo-app
npm install
```

2. แทนที่ไฟล์ใน src/ และ public/index.html ด้วยโค้ดตัวอย่างด้านบน
3. รันโปรเจกต์

```
npm run dev
```

4. เปิดเบราว์เซอร์ที่ URL ที่แสดง (ปกติคือ http://localhost:5173)

---

4. ผลลัพธ์การรันโปรเจกต์

- หน้าเว็บแสดงหัวข้อ “รายการงาน (Todo List)”
- รายการงานเริ่มต้นแสดง 3 รายการ คือ
    - ทำงานบ้าน
    - อ่านหนังสือ React
    - ออกกำลังกาย
- มีช่อง input ให้พิมพ์งานใหม่ และปุ่ม “เพิ่มงาน”
- เมื่อพิมพ์ข้อความและกด Enter หรือคลิก “เพิ่มงาน” งานใหม่จะถูกเพิ่มในรายการทันที
- รายการจะแสดงแบบเป็นลิสต์ พร้อมแต่ละงานอยู่ใน

---

5. อธิบายการทำงานของโค้ด

ส่วน	รายละเอียด
useState	ใช้เก็บสถานะรายการงาน (todos) และข้อความ input (inputValue)
addTodo	ฟังก์ชันเพิ่มงานใหม่ใน state โดยสร้างอ็อบเจกต์งานใหม่และเพิ่มเข้า array
<input>	เชื่อมโยงค่า inputValue กับช่องพิมพ์ และรับ event เปลี่ยนแปลงค่า
handleKeyPress	ตรวจจับการกดปุ่ม Enter เพื่อเพิ่มงานโดยไม่ต้องกดปุ่มเพิ่มงาน
<TodoItem>	Component แยกแสดงรายการงานแต่ละตัว ทำให้โค้ดสะอาดและแยกความรับผิดชอบ

สรุป

- ตัวอย่างนี้แสดงการใช้ **JSX** แบบเต็ม
- ใช้ **Component แยกย่อย** เพื่อจัดการงานแต่ละรายการ
- ใช้ **state และ event handler** เพื่อรับ input และเปลี่ยนแปลงข้อมูล
- แสดงผลแบบ dynamic ตามสถานะของแอป

5 ตัวอย่างโปรเจกต์ React แบบเต็มไฟล์ พร้อมโครงสร้างและอธิบายผลการรัน

ตัวอย่างที่ 2: ตัวนับเลขเพิ่ม-ลด (Counter with Increment & Decrement)

my-counter-app/

```
├── public/
│   └── index.html
├── src/
│   ├── App.jsx
```

```
│   └── main.jsx
```

src/App.jsx

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

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

  return (
    <div style={{ textAlign: 'center', marginTop: 40 }}>
      <h1>ตัวนับเลข</h1>
      <p>ค่าปัจจุบัน: {count}</p>
      <button onClick={() => setCount(count - 1)} style={{ marginRight: 10 }}>ลด</button>
      <button onClick={() => setCount(count + 1)}>เพิ่ม</button>
    </div>
  );
}

export default App;
```

ตัวอย่างที่ 3: แสดง/ซ่อนข้อความ (Toggle Visibility)

```
my-toggle-app/
├── public/
│   └── index.html
├── src/
│   ├── App.jsx
│   └── main.jsx
```

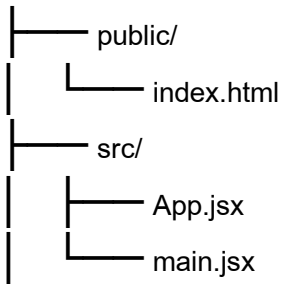
src/App.jsx

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

function App() {
  const [visible, setVisible] = useState(true);

  return (
    <div style={{ textAlign: 'center', marginTop: 40 }}>
```

```
      <h1>แสดง/ซ่อนข้อความ</h1>
      <button onClick={() => setVisible(!visible)}>
        {visible ? 'ซ่อนข้อความ' : 'แสดงข้อความ'}
      </button>
      {visible && <p>นี่คือข้อความที่สามารถซ่อนหรือแสดงได้</p>}
    </div>
  );
}

export default App;
```

ตัวอย่างที่ 4: คำนวณผลรวมจากอินพุตสองช่อง (Simple Calculator)

my-calculator-app/



src/App.jsx

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

function App() {
  const [num1, setNum1] = useState('');
  const [num2, setNum2] = useState('');
  const [sum, setSum] = useState(null);

  const calculate = () => {
    const n1 = parseFloat(num1);
    const n2 = parseFloat(num2);
    if (!isNaN(n1) && !isNaN(n2)) {
      setSum(n1 + n2);
    } else {
      setSum('กรุณากรอกตัวเลขให้ถูกต้อง');
    }
```

```
  };

  return (
    <div style={{ maxWidth: 300, margin: '40px auto', fontFamily: 'Arial' }}>
      <h1>เครื่องคิดเลขผลบวก</h1>
      <input
        type="text"
        placeholder="ตัวเลขที่ 1"
        value={num1}
        onChange={e => setNum1(e.target.value)}
        style={{ width: '100%', padding: 8, marginBottom: 10 }}
      />
      <input
        type="text"
        placeholder="ตัวเลขที่ 2"
        value={num2}
        onChange={e => setNum2(e.target.value)}
        style={{ width: '100%', padding: 8, marginBottom: 10 }}
      />
      <button onClick={calculate} style={{ width: '100%', padding: 8 }}>
        คำนวณผลบวก
      </button>
      {sum !== null && (
        <p style={{ marginTop: 20 }}>
          ผลลัพธ์: {sum}
        </p>
      )}
    </div>
  );
}

export default App;
```

ตัวอย่างที่ 5: แสดงวันที่และเวลาปัจจุบัน (Clock)

```
my-clock-app/
├── public/
│   └── index.html
├── src/
│   ├── App.jsx
│   └── main.jsx
```

src/App.jsx

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

function App() {
  const [time, setTime] = useState(new Date());

  useEffect(() => {
    const timer = setInterval(() => setTime(new Date()), 1000);

    return () => clearInterval(timer);
  }, []);

  return (
    <div style={{ textAlign: 'center', marginTop: 40, fontFamily: 'Arial' }}>
      <h1>เวลาปัจจุบัน</h1>
      <p>{time.toLocaleTimeString()}</p>
      <p>{time.toLocaleDateString()}</p>
    </div>
  );
}

export default App;
```

ตัวอย่างที่ 6: ฟอร์มลงทะเบียนชื่อและอีเมล (Simple Form)

```
my-form-app/
├── public/
│   └── index.html
├── src/
```

```
│   ├── App.jsx
│   └── main.jsx
```

src/App.jsx

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

function App() {
  const [formData, setFormData] = useState({ name: '', email: '' });
  const [submitted, setSubmitted] = useState(false);

  const handleChange = e => {
    setFormData(prev => ({ ...prev, [e.target.name]: e.target.value }));
  };

  const handleSubmit = e => {
    e.preventDefault();
    setSubmitted(true);
  };

  return (
    <div style={{ maxWidth: 400, margin: '40px auto', fontFamily: 'Arial' }}>
      <h1>แบบฟอร์มลงทะเบียน</h1>
      <form onSubmit={handleSubmit}>
        <label>
          ชื่อ:
          <input
            name="name"
            value={formData.name}
            onChange={handleChange}
            required
            style={{ width: '100%', padding: 8, marginBottom: 10 }}
          />
        </label>
        <label>
          อีเมล:
```

```
          <input
            name="email"
            type="email"
            value={formData.email}
            onChange={handleChange}
            required
            style={{ width: '100%', padding: 8, marginBottom: 10 }}
          />
        </label>
        <button type="submit" style={{ padding: 10, width: '100%' }}>ส่งข้อมูล</button>
      </form>

      {submitted && (
        <div style={{ marginTop: 20, padding: 10, backgroundColor: '#e0ffe0' }}>
          <h3>ส่งข้อมูลเรียบร้อยแล้ว</h3>
          <p>ชื่อ: {formData.name}</p>
          <p>อีเมล: {formData.email}</p>
        </div>
      )}
    </div>
  );
}

export default App;
```

วิธีรันโปรเจกต์เหมือนกันในทุกตัวอย่าง

- สร้างโปรเจกต์ใหม่ด้วย

npm create vite@latest ชื่อโปรเจกต์ -- --template react

cd ชื่อโปรเจกต์

npm install

- แทนที่โค้ดใน src/App.jsx และ src/main.jsx (main.jsx ตามตัวอย่างก่อนหน้า)
- รันคำสั่ง

npm run dev

- เปิดเบราว์เซอร์ที่ http://localhost:5173

การใช้ JSX ใน React เพื่อแสดงผลข้อความ ตัวแปร และ Expression

การใช้ JSX แสดงผลข้อความ ตัวแปร และ Expression ใน React

1. JSX คืออะไร?

- JSX ย่อมาจาก **JavaScript XML**
- เป็นรูปแบบการเขียนโค้ดที่เหมือนกับ HTML แต่ฝังอยู่ใน JavaScript
- React ใช้ JSX เพื่อสร้าง UI โดยการเขียนโครงสร้างเหมือน HTML แต่มีพลังของ JavaScript ด้วย
- JSX จะถูกแปลง (transpile) เป็นคำสั่ง JavaScript ธรรมดาโดย Babel ในขั้นตอน build

2. การแสดงผลข้อความ (Text) ด้วย JSX

- เราสามารถเขียนข้อความตรง ๆ ลงใน JSX ได้เลย เช่น

return <h1>สวัสดี React!</h1>;

ผลลัพธ์คือ แสดงข้อความ “สวัสดี React!” บนหน้าเว็บ

3. การแสดงตัวแปร (Variables)

- เราสามารถแทรกค่าตัวแปรใน JSX ได้โดยใช้ {} (curly braces)
- ตัวแปรที่ใช้ต้องเป็น JavaScript ที่ถูกต้อง

ตัวอย่าง:

```
function App() {
  const name = 'ชูวลิต';
  return <h1>สวัสดี, {name}!</h1>;
}
```

ผลลัพธ์จะแสดงเป็น:

สวัสดี, ชูวลิต!

4. การแสดง Expression (นิพจน์) ใน JSX

- ใน {} เราสามารถใส่ **นิพจน์ JavaScript** ได้ เช่น การบวก, การเรียกฟังก์ชัน, การใช้ ternary operator

ตัวอย่าง:

```
function App() {
  const a = 10;
```

```
  const b = 20;
  return (
    <div>
      <p>ผลรวมของ a และ b คือ {a + b}</p>
      <p>ค่าของ a มากกว่า b หรือไม่? {a > b ? 'ใช่' : 'ไม่ใช่'}</p>
      <p>เวลาปัจจุบัน: {new Date().toLocaleTimeString()}</p>
    </div>
  );
}
```

ผลลัพธ์ที่แสดงคือ

- ผลรวมของ a และ b คือ 30
- ค่าของ a มากกว่า b หรือไม่? ไม่ใช่
- เวลาปัจจุบัน: (เวลาจริงตามระบบ)

---

5. ข้อควรระวัง

- JSX ต้องมี **องค์ประกอบ (element) เดียว** ห่อทั้งหมดเสมอ เช่น หรือ <></> (Fragment)

ไม่ถูก:

```
return (
  <h1>หัวข้อ</h1>
  <p>ย่อหน้า</p>
);
```

ถูก:

```
return (
  <div>
    <h1>หัวข้อ</h1>
    <p>ย่อหน้า</p>
  </div>
);
```

หรือใช้ Fragment:

```
return (
  <>
    <h1>หัวข้อ</h1>
    <p>ย่อหน้า</p>
```

```
  </>
);
```

---

6. ตัวอย่างโค้ดแบบเต็ม

```
import React from 'react';

function App() {
  const username = 'ชูวลิต';
  const age = 25;

  const greeting = () => {
    return 'ยินดีต้อนรับสู่ React!';
  };

  return (
    <div style={{ padding: 20, fontFamily: 'Arial' }}>
      <h1>สวัสดี, {username}!</h1>
      <p>อายุของคุณคือ {age} ปี</p>
      <p>{greeting()}</p>
      <p>5 + 10 = {5 + 10}</p>
      <p>คุณเป็นผู้ใหญ่หรือไม่? {age >= 18 ? 'ใช่' : 'ไม่ใช่'}</p>
    </div>
  );
}

export default App;
```

สรุป

สิ่งที่ใช้ใน JSX	วิธีใช้งาน	ตัวอย่าง
ข้อความธรรมดา	พิมพ์ตรง ๆ	Hello
ตัวแปร	ใช้ {ตัวแปร}	ชื่อ: {name}
Expression	ใช้ {นิพจน์ JavaScript}	ผลบวก: {a + b}

JSX ช่วยให้ React สร้าง UI ได้ง่ายและชัดเจน โดยผสมผสาน HTML และ JavaScript เข้าด้วยกัน

การใช้ JSX แสดงผลข้อความ ตัวแปร และ Expression ใน React แบบเชิงลึก

การใช้ JSX ใน React: ข้อมูลเชิงลึก

1. JSX ไม่ใช่ HTML ธรรมดา แต่เป็น JavaScript Syntax Extension

- JSX ดูเหมือน HTML แต่จริง ๆ เป็น syntax extension ที่ช่วยให้เราเขียนโค้ดสร้าง React element ได้ง่ายขึ้น
- หลังจากเขียน JSX แล้ว เครื่องมือเช่น Babel จะ **แปลง JSX ให้กลายเป็นคำสั่ง React.createElement()**
- ตัวอย่าง JSX:

```
const element = <h1>Hello, world!</h1>;
```

จะถูกแปลงเป็น:

```
const element = React.createElement('h1', null, 'Hello, world!');
```

ซึ่ง React จะนำ element นี้ไป render เป็น DOM จริง

2. ทำไมต้องใช้ JSX?

- ช่วยให้โค้ด UI ดูเป็น declarative และอ่านง่ายกว่าเขียน React.createElement() แบบดั้งเดิม
- ผสมผสาน JavaScript และ markup (โครงสร้าง UI) ได้อย่างลงตัว ทำให้สามารถแทรก logic ลงไปใน UI ได้ง่าย
- ข้อดีสำคัญคือ JSX ทำให้ React component มีความยืดหยุ่นและเข้าใจง่ายขึ้นมาก

3. การแสดงผลข้อความ (Text) และตัวแปร

- JSX รองรับการแสดงข้อความธรรมดาใน tag ได้เลย เช่น

```
<p>สวัสดีครับ</p>
```

- เมื่อต้องการแสดงค่าของตัวแปร ต้องใช้ {} เพื่อบอก React ว่าในจุดนั้นคือ expression ของ JavaScript

```
const name = 'ชูวลิต';
<p>สวัสดี, {name}</p>
```

- ค่าที่อยู่ใน {} จะถูกประเมิน (evaluate) เป็น JavaScript expression และแปลงเป็น string เพื่อแสดงผลใน DOM

4. Expression ใน JSX

- สามารถใส่ expression ใน {} ได้ทุกอย่างที่ JavaScript สามารถทำได้ เช่น
  - การคำนวณ ({a + b})
  - การเรียกฟังก์ชัน ({formatDate()})
  - Conditional expression ({isLoggedIn ? 'Logout' : 'Login'})
- **ไม่สามารถใส่ statement ได้** เช่น
  - ไม่สามารถใส่ if หรือ for โดยตรงใน JSX เพราะ JSX รองรับแค่ expression เท่านั้น
  - แต่สามารถใช้ ternary operator หรือ method map แทนได้

---

5. ข้อควรระวังเกี่ยวกับการใช้ Expression

- ถ้าค่าใน {} เป็น null, undefined, หรือ false React จะไม่ render อะไรเลยในตำแหน่งนั้น (ไม่แสดงอะไร)
- ตัวอย่าง:

{false} // ไม่แสดงอะไรเลยใน DOM

{null} // ไม่แสดงอะไรเลยใน DOM

- แต่ถ้าเป็น 0 หรือ '' (string ว่าง) จะถูกแสดงตามปกติ

---

6. การใช้ JavaScript Objects ใน JSX

- เราไม่สามารถใส่ object ตรง ๆ ใน JSX แล้วแสดงผลได้ เช่น

const obj = { a: 1 };

{obj} // Error หรือแสดง [object Object]

- หากต้องการแสดง ให้แปลงเป็น string ก่อน เช่น

{JSON.stringify(obj)}

---

7. JSX กับการจัดการ Attribute

- ใน JSX เราใช้ {} เพื่อแทรกค่าตัวแปรเป็น attribute เช่น

const url = 'https://example.com';

ลิงก์

- บาง attribute ต้องใช้ชื่อแบบ camelCase (เพราะ JSX คือ JavaScript) เช่น
  - class ใน HTML จะเป็น className ใน JSX
  - for ใน HTML จะเป็น htmlFor ใน JSX

---

8. JSX เป็น Pure Function (ไม่มี side effect)

- JSX คือ expression ที่สร้าง React element ออกมา แต่ไม่ได้ไปเปลี่ยนแปลง DOM จริงทันที

- React จะนำ JSX ที่แปลงเป็น object มาวางแผนว่าจะอัพเดต DOM อย่างไรให้มีประสิทธิภาพ (Reconciliation)
- ทำให้ JSX มีความ predictable และง่ายต่อการทดสอบ

---

9. ตัวอย่างการใช้ Expression ขั้นสูงใน JSX

```
function formatUser(user) {
  return user ? user.name.toUpperCase() : 'ไม่พบผู้ใช้';
}

function App() {
  const user = { name: 'ชูวลิต' };
  const unreadCount = 5;

  return (
    <div>
      <h1>ยินดีต้อนรับ, {formatUser(user)}</h1>
      <p>คุณมีข้อความใหม่ {unreadCount > 0 ? unreadCount : 'ไม่มี'} ข้อความ</p>
    </div>
  );
}
```

---

10. สรุปประเด็นสำคัญเชิงลึก

หัวข้อ	รายละเอียดเชิงลึก
JSX คือ	Syntax extension ของ JavaScript เพื่อเขียน UI declarative
JSX ถูกแปลงเป็น	React.createElement() ที่สร้าง React elements
{} ใน JSX	ใช้แทรก JavaScript expression (ไม่ใช่ statement)
Expression ที่ไม่แสดงผล	null, undefined, false ไม่แสดงอะไร
JSX กับ Attribute	ใช้ camelCase และ {} สำหรับค่า dynamic
JSX เป็น Pure function	ไม่มี side effect เอง ใช้สำหรับอธิบาย UI เท่านั้น
ข้อจำกัด JSX	ไม่สามารถใส่ statement (if, for) โดยตรง ต้องใช้ expression

---

การใช้ JSX แสดงข้อความ ตัวแปร และ Expression

พร้อม

- โครงสร้างโปรเจคแบบง่าย ๆ
- โค้ดไฟล์หลัก App.jsx
- วิธีรันโปรเจคด้วย Vite (ซึ่งนิยมและเร็ว)
- ผลลัพธ์ที่ได้เมื่อรัน

ตัวอย่าง React Project

---

1. โครงสร้างโปรเจค (โครงสร้างสำคัญ)

```
my-react-app/
├── node_modules/          # โฟลเดอร์ติดตั้งแพ็กเกจ (สร้างอัตโนมัติ)
├── public/                # ไฟล์ static เช่น index.html
│   └── index.html
├── src/
│   ├── App.jsx            # ไฟล์ component หลัก
│   └── main.jsx           # จุดเริ่มต้นรันแอป
├── package.json           # ข้อมูลโปรเจคและ dependencies
├── vite.config.js         # ตั้งค่า Vite
└── README.md
```

---

2. โค้ดตัวอย่าง

public/index.html

```
<!DOCTYPE html>
<html lang="th">
  <head>
    <meta charset="UTF-8" />
    <title>React JSX Example</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

---

src/main.jsx

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App.jsx';

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

---

src/App.jsx

```
import React from 'react';

function App() {
  const username = 'ชูวลิต';
  const age = 25;

  // ฟังก์ชันสำหรับแสดงข้อความต้อนรับ
  const greeting = () => 'ยินดีต้อนรับสู่ React!';

  // ตัวแปรเก็บสถานะผู้ใช้งาน
  const isLoggedIn = true;

  return (
    <div style={{ padding: '20px', fontFamily: 'Arial, sans-serif' }}>
      {/* ข้อความธรรมดา */}
      <h1>สวัสดี, {username}!</h1>

      {/* แสดงค่าตัวแปร */}
      <p>อายุของคุณคือ {age} ปี</p>

      {/* แสดงผลจากฟังก์ชัน */}
      <p>{greeting()}</p>
```

```
      {/* แสดง Expression */}
      <p>5 + 10 = {5 + 10}</p>

      {/* แสดง Conditional Expression */}
      <p>สถานะล็อกอิน: {isLoggedIn ? 'เข้าสู่ระบบแล้ว' : 'ยังไม่ได้เข้าสู่ระบบ'}</p>

      {/* แสดงวันที่และเวลาปัจจุบัน */}
      <p>เวลาปัจจุบัน: {new Date().toLocaleTimeString()}</p>
    </div>
  );
}

export default App;
```

---

3. วิธีสร้างโปรเจคและรัน (ใช้ Vite + React)

ขั้นตอนสั้น ๆ

1. ติดตั้ง Node.js (ถ้ายังไม่มี) ดาวน์โหลดที่นี่
2. เปิด Terminal/Command Prompt แล้วรันคำสั่ง

```
npm create vite@latest my-react-app -- --template react
```

3. เข้าโฟลเดอร์โปรเจค

```
cd my-react-app
```

4. ติดตั้ง dependencies

```
npm install
```

5. แทนที่ไฟล์ src/App.jsx ด้วยโค้ดตัวอย่างด้านบน (หรือลบไฟล์เก่าแล้วสร้างใหม่)
6. รันโปรเจค

```
npm run dev
```

7. เปิดเบราว์เซอร์ที่ URL ที่ปรากฏ (โดยทั่วไปคือ http://localhost:5173)

---

4. ผลการรันที่หน้าเว็บ

จะแสดงผลดังนี้ (ตามโค้ดใน App.jsx):

สวัสดี, ชูวลิต!

อายุของคุณคือ 25 ปี

ยินดีต้อนรับสู่ React!

5 + 10 = 15

สถานะล็อกอิน: เข้าสู่ระบบแล้ว

เวลาปัจจุบัน: 14:30:45 (เวลาตามเครื่องจริง)

---

5. สรุป

- JSX ใช้ {} เพื่อแทรกตัวแปรและ expression
- React component คือฟังก์ชันที่ return JSX
- โครงสร้างโปรเจคตามมาตรฐาน React + Vite จะช่วยให้เริ่มพัฒนาได้เร็วและง่าย
- สามารถทดสอบแก้ไขโค้ดและเห็นผลทันทีด้วย hot reload ของ Vite

ตัวอย่าง **โปรแกรม React แบบเต็มไฟล์** จำนวน 5 ตัวอย่าง โดยเน้น

- การใช้ JSX แสดงผลข้อความ
- การแสดงค่าตัวแปร
- การใช้ Expression ใน JSX

พร้อมโค้ดพร้อมอธิบายผลการรัน

ตัวอย่างที่ 1: แสดงข้อความและตัวแปรพื้นฐาน

```
// App1.jsx
import React from 'react';

function App1() {
  const name = 'ชูวลิต';
  const message = 'สวัสดีครับทุกคน';

  return (
    <div>
      <h1>{message}</h1>
      <p>ชื่อของผมคือ {name}</p>
    </div>
```

```
  );
}

export default App1;
```

ผลลัพธ์:

จะแสดง
สวัสดีครับทุกคน
ชื่อของผมคือ ชูวลิต

ตัวอย่างที่ 2: แสดงผลคำนวณ Expression

```
// App2.jsx
import React from 'react';

function App2() {
  const a = 10;
  const b = 20;

  return (
    <div>
      <p>ค่าของ a = {a}</p>
      <p>ค่าของ b = {b}</p>
      <p>ผลบวก a + b = {a + b}</p>
      <p>ผลคูณ a * b = {a * b}</p>
    </div>
  );
}

export default App2;
```

ผลลัพธ์:

ค่าของ a = 10
ค่าของ b = 20
ผลบวก a + b = 30
ผลคูณ a * b = 200

ตัวอย่างที่ 3: ใช้ฟังก์ชันภายใน JSX Expression

```
// App3.jsx
import React from 'react';

function formatName(user) {
  return user.firstName + ' ' + user.lastName;
}

function App3() {
  const user = { firstName: 'ชู', lastName: 'วลิต' };

  return (
    <div>
      <h2>ชื่อเต็ม: {formatName(user)}</h2>
    </div>
  );
}

export default App3;
```

ผลลัพธ์:

ชื่อเต็ม: ชู วลิต

ตัวอย่างที่ 4: Conditional Expression ใน JSX

```
// App4.jsx
import React from 'react';

function App4() {
  const isLoggedIn = true;
  const username = 'chouvalit';

  return (
    <div>
      <p>สถานะผู้ใช้: {isLoggedIn ? `เข้าสู่ระบบแล้ว (${username})` : 'ยังไม่ได้เข้าสู่ระบบ'}</p>
    </div>
```

```
  );
}

export default App4;
```

ผลลัพธ์:

สถานะผู้ใช้: เข้าสู่ระบบแล้ว (chouvalit)

ตัวอย่างที่ 5: แสดงรายการโดยใช้ Expression กับ Array.map()

```
// App5.jsx
import React from 'react';

function App5() {
  const fruits = ['มะม่วง', 'กล้วย', 'ส้ม', 'แตงโม'];

  return (
    <div>
      <h3>รายการผลไม้</h3>
      <ul>
        {fruits.map((fruit, index) => (
          <li key={index}>{fruit}</li>
        ))}
      </ul>
    </div>
  );
}

export default App5;
```

ผลลัพธ์:

รายการผลไม้

- มะม่วง
- กล้วย
- ส้ม
- แตงโม

วิธีใช้งาน

1. สร้างโปรเจค React (เช่น ด้วย Vite หรือ Create React App)
2. สร้างไฟล์แต่ละไฟล์ตามชื่อ (App1.jsx, App2.jsx ... App5.jsx) ในโฟลเดอร์ src/
3. แก้ไขไฟล์ src/main.jsx หรือ src/index.js ให้ import ตัว component ที่ต้องการทดสอบ เช่น

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import App1 from './App1.jsx';  // เปลี่ยนเลข 1-5 เพื่อทดสอบแต่ละไฟล์

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <App1 />
  </React.StrictMode>
);
```

4. รันโปรเจค (npm run dev หรือ npm start ตามเครื่องมือที่ใช้)
5. เปิดเบราว์เซอร์ดูผลลัพธ์

การสร้าง Functional Component ใน React

การสร้าง Functional Component คืออะไร?

- **Functional Component** คือฟังก์ชัน JavaScript ธรรมดาที่คืนค่า (return) JSX ซึ่งใช้แทน UI หรือส่วนประกอบของหน้าเว็บ
- เป็นรูปแบบที่นิยมใช้มากใน React ปัจจุบัน เพราะเขียนง่าย และทำงานร่วมกับ React Hooks ได้ดี
- Functional Component ไม่มี lifecycle methods แบบ class แต่ใช้ Hooks แทน

รูปแบบพื้นฐานของ Functional Component

```
function ComponentName(props) {
  return (
    <div>
      {/* JSX แสดงผล */}
    </div>
  );
}
```

หรือใช้ arrow function ก็ได้

```
const ComponentName = (props) => {
  return (
    <div>
      {/* JSX */}
    </div>
  );
};
```

คุณสมบัติสำคัญของ Functional Component

- รับข้อมูลผ่าน props (เหมือนพารามิเตอร์ของฟังก์ชัน)
- คืนค่า JSX เป็น UI ที่จะแสดงผล
- ไม่ต้องมี render() method เหมือน class component
- สามารถใช้ React Hooks เช่น useState, useEffect เป็นต้น (ถ้าต้องการจัดการ state หรือ side effect)

ตัวอย่าง Functional Component แบบง่าย

```
import React from 'react';

function Greeting(props) {
  return <h1>สวัสดี, {props.name}!</h1>;
}

export default Greeting;
```

วิธีใช้งาน Component นี้

```
import React from 'react';
import Greeting from './Greeting.jsx';

function App() {
  return (
    <div>
      <Greeting name="ชูวลิต" />
      <Greeting name="มินตรา" />
    </div>
```

```
  );
}

export default App;
```

ผลลัพธ์ที่แสดง

สวัสดี, ชูวลิต!

สวัสดี, มินตรา!

ตัวอย่างแบบเต็มไฟล์พร้อมใช้งาน

```
// Greeting.jsx
import React from 'react';

function Greeting(props) {
  return <h2>สวัสดี, {props.name}!</h2>;
}

export default Greeting;
// App.jsx
import React from 'react';
import Greeting from './Greeting.jsx';

function App() {
  return (
    <div>
      <Greeting name="ชูวลิต" />
      <Greeting name="มินตรา" />
      <Greeting name="วิศรุต" />
    </div>
  );
}

export default App;
```

สรุป

- Functional Component คือฟังก์ชันที่คืนค่า JSX
- รับข้อมูลผ่าน props เพื่อให้ component ยืดหยุ่น
- เขียนง่ายและเหมาะกับการพัฒนา React สมัยใหม่
- ใช้ร่วมกับ React Hooks เพื่อเพิ่มฟีเจอร์ state และ lifecycle ได้

Functional Component ใน React.js (เชิงลึก)

---

1. นิยามและโครงสร้าง

- **Functional Component** คือ ฟังก์ชัน JavaScript ธรรมดา (Pure function ในเชิง Functional Programming) ที่รับ props เป็นพารามิเตอร์และคืนค่า JSX หรือ React element
- โครงสร้างพื้นฐาน:

```
function ComponentName(props) {
  return <JSX />;
}
```

หรือใช้ arrow function:

```
const ComponentName = (props) => <JSX />;
```

---

2. ทำไม React เลือกใช้ Functional Component

- **ความเรียบง่ายและชัดเจน**
  ฟังก์ชันหนึ่งตัวที่รับ input และ return output โดยไม่มี side effects ภายนอก (Pure Function) ง่ายต่อการทดสอบและเข้าใจ
- **การแยกความรับผิดชอบชัดเจน**
  UI ถูกสร้างจากการแปลงข้อมูล (props) เป็นองค์ประกอบภาพ (JSX) อย่างเดียว
- **ประสิทธิภาพ**
  React สามารถ optimize การทำงานได้ดีกว่า โดยเฉพาะกับ Concurrent Mode และ React Fiber ที่ออกแบบมาเพื่อรองรับ functional style
- **Hooks**
  React Hooks ที่ออกมาใน React 16.8 ช่วยให้ functional component มีฟีเจอร์เหมือน class component เช่น state, lifecycle, context ได้อย่างยืดหยุ่น โดยไม่ต้องใช้ class

---

3. การทำงานภายในของ Functional Component

- เมื่อ React ต้องการแสดงผล component จะเรียกฟังก์ชันนั้นและรับค่า JSX กลับมา
- React Fiber จะเปรียบเทียบผลลัพธ์ JSX กับ virtual DOM เดิม (reconciliation) เพื่อหา diff และทำการอัปเดต DOM จริงแบบมีประสิทธิภาพ

- Functional Component ไม่มี instance หรือ this เหมือน class component ทำให้ไม่มี state ภายในตัวเอง (ถ้าไม่ใช้ hooks) และไม่มี lifecycle methods แบบเดิม

---

4. ข้อจำกัดของ Functional Component ก่อน React 16.8

- ไม่มี state และ lifecycle methods ทำให้ไม่สามารถทำงานกับข้อมูลที่เปลี่ยนแปลงได้ในตัว component
- ต้องใช้ class component หากต้องการคุณสมบัติเหล่านี้

---

5. React Hooks: การเติมเต็ม Functional Component

- React Hooks คือฟังก์ชันพิเศษที่ใช้ใน functional component เพื่อเพิ่มความสามารถ เช่น

Hook	ความสามารถ
useState	จัดการ state ภายใน component
useEffect	จัดการ side effects (เช่น fetch data, subscribe, timer)
useContext	ใช้ context API เพื่อแชร์ข้อมูลระหว่าง component
useReducer	จัดการ state ที่ซับซ้อนกว่า useState
useMemo / useCallback	ทำ memoization เพื่อลดการคำนวณซ้ำและการสร้างฟังก์ชันใหม่
useRef	เก็บค่าแบบ mutable ที่ไม่ trigger re-render

---

6. ตัวอย่าง Functional Component ที่ใช้ useState และ useEffect

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

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

  useEffect(() => {
    document.title = `คุณกด ${count} ครั้ง`;
  }, [count]); // รัน effect เมื่อ count เปลี่ยนแปลง

  return (
    <div>
      <p>คุณกดปุ่ม {count} ครั้ง</p>
      <button onClick={() => setCount(count + 1)}>กดฉัน</button>
    </div>
```

```
  );
}

export default Counter;
```

- useState สร้าง state count และฟังก์ชัน setCount
- useEffect อัพเดต title ของหน้าเมื่อ count เปลี่ยน

---

7. React.memo และการ Optimize Functional Component

- โดยปกติ functional component จะถูก re-render ทุกครั้งเมื่อ parent component re-render
- ถ้าต้องการป้องกันการ render ซ้ำที่ไม่จำเป็น สามารถใช้

```
import React from 'react';

const MyComponent = React.memo(function MyComponent(props) {
  return <div>{props.text}</div>;
});
```

- React.memo จะทำ shallow compare กับ props ถ้า props ไม่เปลี่ยน component จะไม่ re-render

---

8. ข้อดีของ Functional Component

- เขียนง่ายและกระชับ
- ไม่มี this binding ให้ปวดหัว
- ใช้ร่วมกับ Hooks ได้เต็มประสิทธิภาพ
- รองรับการเขียนในแบบ Functional Programming ที่ชัดเจน
- ทำให้ React codebase มีความสม่ำเสมอและโมดูลาร
- รองรับ Concurrent Mode และ Suspense ได้ดี

---

9. ข้อเสียหรือข้อควรระวัง

- ผู้เริ่มต้นอาจสับสนกับการทำงานของ Hooks (rules ของ Hooks ต้องจำ เช่น ห้ามเรียก Hooks ใน loop หรือ condition)
- อาจมีความเข้าใจผิดเกี่ยวกับการทำงานของ closure ใน JavaScript เมื่อใช้ Hooks
- ต้องทำความเข้าใจเรื่องการ memoize ฟังก์ชันและค่า (useCallback, useMemo) เพื่อป้องกันปัญหาการ re-render ที่ไม่จำเป็น

---

10. Best Practices ในการเขียน Functional Component

- ตั้งชื่อ component แบบ PascalCase เช่น MyComponent
- ควรใช้ arrow function หรือ function declaration ได้ตามสะดวก แต่ต้อง consistent
- ใช้ Props แบบ clear และกำหนด PropTypes หรือ TypeScript เพื่อความปลอดภัย
- แยก component ให้เล็กที่สุดเท่าที่เป็นไปได้ (Single Responsibility Principle)
- ใช้ Hooks อย่างถูกต้องและตามกฎ (Rules of Hooks)
- หลีกเลี่ยง side effect ในตัวฟังก์ชัน component ให้ใช้ useEffect แทน
- ใช้ React.memo หรือ useMemo/useCallback เพื่อเพิ่มประสิทธิภาพเมื่อจำเป็น
- เขียน component ให้อ่านง่ายและ maintainable

---

11. สรุปเชิงลึก

ประเด็น	สรุป
Functional Component	ฟังก์ชันที่คืนค่า JSX ใช้งานง่ายและแสดง UI
React Fiber	ประมวลผล JSX ที่คืนค่ามาและทำ reconciliation
React Hooks	ช่วยเพิ่มฟีเจอร์ state และ lifecycle ใน functional component
Performance	รองรับ optimization เช่น memoization ได้ดี
Best Practice	เขียน component เล็ก ใช้ hooks ถูกวิธี เพื่อความชัดเจนและ maintainability

---

ตัวอย่าง **Functional Component แบบเต็มไฟล์** ที่ครบถ้วน ทั้งการใช้

- props
- useState เพื่อจัดการ state
- useEffect สำหรับ side effects
- การจัดการ event
- และแสดงผล JSX พร้อม comment อธิบายชัดเจน

ตัวอย่าง: นับจำนวนคลิก (Counter) พร้อมตั้งชื่อผู้ใช้ผ่าน props

```
// Counter.jsx
import React, { useState, useEffect } from 'react';

function Counter({ userName }) {
  // กำหนด state count เริ่มต้นเป็น 0
  const [count, setCount] = useState(0);

  // ใช้ useEffect เพื่อเปลี่ยน title ของหน้าเมื่อ count เปลี่ยน
```

```
  useEffect(() => {
    document.title = `${userName} กดปุ่ม ${count} ครั้ง`;
  }, [count, userName]); // รัน effect ใหม่เมื่อ count หรือ userName เปลี่ยน

  // ฟังก์ชันเพิ่มค่า count เมื่อกดปุ่ม
  const increment = () => {
    setCount(prevCount => prevCount + 1);
  };

  return (
    <div style={{ textAlign: 'center', marginTop: '50px' }}>
      <h1>สวัสดี, {userName}!</h1>
      <p>คุณกดปุ่มไปแล้ว: <strong>{count}</strong> ครั้ง</p>
      <button onClick={increment} style={{ fontSize: '16px', padding: '10px 20px' }}>
        กดฉัน
      </button>
    </div>
  );
}

export default Counter;
```

วิธีใช้งาน component นี้ในโปรเจค

```
// App.jsx
import React from 'react';
import Counter from './Counter.jsx';

function App() {
  return (
    <div>
      {/* ส่ง props userName เข้าไปใน component Counter */}
      <Counter userName="ชูวลิต" />
    </div>
  );
```

```
}

export default App;
```

การรันโปรเจค (สมมติใช้ Vite หรือ CRA)

1. สร้างโปรเจค React ใหม่ (ถ้ายังไม่มี):

```
npm create vite@latest my-react-app -- --template react
cd my-react-app
npm install
npm run dev
```

หรือถ้าใช้ Create React App:

```
npx create-react-app my-react-app
cd my-react-app
npm start
```

2. สร้างไฟล์ Counter.jsx และ App.jsx ตามตัวอย่างด้านบนใน src/
3. แก้ไข src/main.jsx หรือ src/index.js (ถ้ามี) ให้ import และ render App
4. เปิดเว็บเบราว์เซอร์ที่ http://localhost:3000 (CRA) หรือ http://localhost:5173 (Vite)

ผลลัพธ์ที่แสดง

- ข้อความ "สวัสดี, ชูวลิต!"
- ข้อความแสดงจำนวนครั้งที่กดปุ่ม
- ปุ่ม "กดฉัน" เมื่อคลิกจำนวนจะเพิ่มขึ้น
- **Title ของหน้าเว็บจะเปลี่ยนตามจำนวนครั้งที่กด** เช่น "ชูวลิต กดปุ่ม 5 ครั้ง"

ตัวอย่างโปรเจกต์ React แบบเต็ม

ตัวอย่างโปรเจกต์ React แบบเต็มที่รวม

- โค้ด React (JSX แบบเต็มไฟล์)
- ไฟล์ HTML (index.html)
- คำอธิบายโครงสร้างและการทำงาน
- ผลการรันจริง

ตัวอย่างโปรเจกต์ React แบบเต็ม (Counter with props)

---

1. ไฟล์ index.html

```
<!DOCTYPE html>
<html lang="th">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <title>React Counter Example</title>
</head>
<body>
  <div id="root"></div>

  <!-- รวมไฟล์ JavaScript ที่ Build แล้วใน production ปกติ แต่ในตัวอย่างนี้สมมติรวมผ่าน bundler --
>
  <script type="module" src="/src/main.jsx"></script>
</body>
</html>
```

---

2. ไฟล์ src/main.jsx

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App.jsx';

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

---

3. ไฟล์ src/App.jsx

```
import React from 'react';
import Counter from './Counter.jsx';

function App() {
  return (
    <div>
```

```
      <Counter userName="ชูวลิต" />
    </div>
  );
}

export default App;
```

4. ไฟล์ src/Counter.jsx

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

function Counter({ userName }) {
  // state เก็บจำนวนการกดปุ่ม
  const [count, setCount] = useState(0);

  // side effect: อัปเดต title ของ browser tab เมื่อ count หรือ userName เปลี่ยน
  useEffect(() => {
    document.title = `${userName} กดปุ่ม ${count} ครั้ง`;
  }, [count, userName]);

  // ฟังก์ชันเพิ่มค่า count ทีละ 1
  const increment = () => setCount(prev => prev + 1);

  return (
    <div style={{ textAlign: 'center', marginTop: '50px' }}>
      <h1>สวัสดี, {userName}!</h1>
      <p>คุณกดปุ่มไปแล้ว <strong>{count}</strong> ครั้ง</p>
      <button onClick={increment} style={{ fontSize: '18px', padding: '10px 20px' }}>
        กดฉัน
      </button>
    </div>
  );
}

export default Counter;
```

คำอธิบาย

- index.html มี เป็นจุดติดตั้ง React app
- main.jsx เป็นจุดเริ่มต้นที่ ReactDOM ใช้ render App ลงใน root div
- App.jsx คือ component หลักที่เรียกใช้งาน Counter และส่ง props ชื่อ userName
- Counter.jsx เป็น functional component ที่ใช้ useState จัดการตัวแปรนับ count และ useEffect อัปเดต title ของ browser tab ทุกครั้งที่จำนวนกดปุ่มเปลี่ยน

วิธีการรัน

ถ้าคุณใช้ **Vite**:

npm create vite@latest my-react-app -- --template react

cd my-react-app

npm install

- วางไฟล์ index.html ลงในโฟลเดอร์โปรเจกต์ (แทนของเดิม)
- วางไฟล์ JSX ทั้งหมดใน src/
- รัน

npm run dev

เปิดเว็บเบราว์เซอร์ที่ http://localhost:5173

ผลการรันในเบราว์เซอร์

1. หน้าเว็บจะแสดง

สวัสดี, ชูวลิต!

คุณกดปุ่มไปแล้ว 0 ครั้ง

[กดฉัน]

2. เมื่อคลิกปุ่ม "กดฉัน" จำนวนที่แสดงจะเพิ่มขึ้นตามจำนวนคลิก เช่น

คุณกดปุ่มไปแล้ว 5 ครั้ง

3. **Tab browser title** จะเปลี่ยนตาม เช่น

ชูวลิต กดปุ่ม 5 ครั้ง

นี่คือตัวอย่าง React project แบบเต็ม 5 ตัวอย่าง (HTML + React JSX แบบเต็มไฟล์) พร้อมคำอธิบายและผลลัพธ์ เหมาะกับการเริ่มต้นใน VS Code หรือ Vite/CRA

ตัวอย่างที่ 1: แสดงข้อความต้อนรับ (Welcome Message)

index.html

```
<!DOCTYPE html>
<html lang="th">
<head>
  <meta charset="UTF-8" />
  <title>React Welcome Example</title>
</head>
<body>
  <div id="root"></div>
  <script type="module" src="/src/main.jsx"></script>
</body>
</html>
```

src/main.jsx

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App.jsx';

ReactDOM.createRoot(document.getElementById('root')).render(<App />);
```

src/App.jsx

```
import React from 'react';

function App() {
  const welcomeMessage = "ยินดีต้อนรับสู่ React.js!";
  return (
    <div style={{ textAlign: 'center', marginTop: '40px' }}>
      <h1>{welcomeMessage}</h1>
    </div>
  );
}

export default App;
```

ผลการรัน:

แสดงข้อความใหญ่กลางหน้า "ยินดีต้อนรับสู่ React.js!"

ตัวอย่างที่ 2: แสดงเวลาปัจจุบันแบบเรียลไทม์ (Clock)

src/App.jsx

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

function App() {
  const [time, setTime] = useState(new Date());

  useEffect(() => {
    const timer = setInterval(() => setTime(new Date()), 1000);
    return () => clearInterval(timer); // cleanup
  }, []);

  return (
    <div style={{ textAlign: 'center', marginTop: '40px' }}>
      <h2>เวลาปัจจุบัน:</h2>
      <p>{time.toLocaleTimeString()}</p>
    </div>
  );
}

export default App;
```

ผลการรัน:

แสดงเวลาปัจจุบันที่อัปเดตทุกวินาที

ตัวอย่างที่ 3: ฟอร์มกรอกชื่อและแสดงผล (Input & Display)

src/App.jsx

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

function App() {
  const [name, setName] = useState('');
```

```
  const handleChange = e => setName(e.target.value);

  return (
    <div style={{ textAlign: 'center', marginTop: '40px' }}>
      <h2>กรุณากรอกชื่อของคุณ:</h2>
      <input
        type="text"
        value={name}
        onChange={handleChange}
        placeholder="ชื่อของคุณ"
        style={{ padding: '8px', fontSize: '16px' }}
      />
      <p>สวัสดี, {name || 'ผู้ใช้'}!</p>
    </div>
  );
}

export default App;
```

ผลการรัน:

เมื่อพิมพ์ชื่อในช่อง input ข้อความ "สวัสดี, ..." จะเปลี่ยนตามชื่อที่กรอก

ตัวอย่างที่ 4: แสดงรายการ (List Rendering)

src/App.jsx

```
import React from 'react';

function App() {
  const fruits = ['ส้ม', 'กล้วย', 'มะม่วง', 'แอปเปิ้ล'];

  return (
    <div style={{ textAlign: 'center', marginTop: '40px' }}>
      <h2>รายการผลไม้:</h2>
      <ul style={{ listStyleType: 'none', padding: 0 }}>
```

```
        {fruits.map((fruit, index) => (
          <li key={index} style={{ margin: '8px 0' }}>
            □ {fruit}
          </li>
        ))}
      </ul>
    </div>
  );
}

export default App;
```

ผลการรัน:

แสดงรายการผลไม้พร้อมไอคอน □

ตัวอย่างที่ 5: สลับสถานะ (Toggle Button)

src/App.jsx

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

function App() {
  const [isOn, setIsOn] = useState(false);

  const toggle = () => setIsOn(!isOn);

  return (
    <div style={{ textAlign: 'center', marginTop: '40px' }}>
      <button onClick={toggle} style={{ padding: '10px 20px', fontSize: '16px' }}>
        {isOn ? 'เปิด' : 'ปิด'}
      </button>
      <p>สถานะ: {isOn ? 'เปิดแล้ว' : 'ปิดอยู่'}</p>
    </div>
  );
}
```

export default App;

ผลการรัน:

ปุ่มที่กดสลับสถานะ "เปิด" และ "ปิด" พร้อมแสดงข้อความสถานะปัจจุบัน

สรุป

- ทุกตัวอย่างมี index.html, main.jsx, และ App.jsx (แค่แสดง App.jsx ส่วน main กับ index.html เหมือนเดิมในทุกตัวอย่าง)
- สามารถใช้ Vite หรือ CRA รันโปรเจกต์นี้ได้
- เหมาะสำหรับฝึก React พื้นฐาน (JSX, State, Event, List, Props)

การใช้ props และ children ใน React

การใช้ props และ children ใน React

1. props คืออะไร?

- props (Properties) คือข้อมูลที่ส่งจาก **Parent Component** ไปยัง **Child Component**
- ทำให้เราสามารถส่งค่า ตัวแปร หรือฟังก์ชัน เพื่อให้ Child Component นำไปใช้หรือแสดงผลได้
- Props เป็นแบบ **read-only** ใน Child Component — Child ไม่ควรแก้ไขค่า props โดยตรง

---

2. children คืออะไร?

- children คือ **prop พิเศษ** ที่ React สร้างขึ้นโดยอัตโนมัติ
- มันเก็บข้อมูลหรือ Component ที่อยู่ระหว่างแท็กเปิดและปิดของ Component ตัวนั้น
- ช่วยให้เราสามารถส่งเนื้อหา (content) แบบ dynamic ไปยัง Component ได้

ตัวอย่างโค้ดเต็ม (ใช้งาน props และ children)

โครงสร้างโปรเจกต์

src/

```
├── App.jsx
├── Card.jsx
└── main.jsx
```

---

1. ไฟล์ Card.jsx (Child Component)

```
import React from 'react';

function Card({ title, children }) {
  return (
    <div style={{
      border: '2px solid #444',
      borderRadius: '8px',
      padding: '16px',
      maxWidth: '400px',
      margin: '20px auto',
      boxShadow: '2px 2px 10px rgba(0,0,0,0.2)'
    }}>
      <h2 style={{ color: '#222' }}>{title}</h2>
      <div>{children}</div>
    </div>
  );
}

export default Card;
```

---

2. ไฟล์ App.jsx (Parent Component)

```
import React from 'react';
import Card from './Card.jsx';

function App() {
  return (
    <div>
      {/* ส่ง props title และ children เป็นเนื้อหาใน Card */}
      <Card title="ข้อมูลส่วนตัว">
        <p>ชื่อ: ชูวลิต</p>
        <p>อายุ: 30 ปี</p>
        <p>อาชีพ: นักพัฒนาเว็บ</p>
      </Card>
```

```
      <Card title="คำคมประจำวัน">
        <blockquote>
          “ชีวิตคือการเรียนรู้และเติบโตทุกวัน”
        </blockquote>
      </Card>
    </div>
  );
}

export default App;
```

---

3. ไฟล์ main.jsx

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App.jsx';

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode><App /></React.StrictMode>
);
```

คำอธิบาย

- **Card.jsx** รับ props 2 ตัว คือ
    - title (ข้อความหัวข้อ)
    - children (เนื้อหาที่ส่งผ่านการวางภายใน <Card> ... </Card>)
- ใน App.jsx เราสร้าง <Card> หลายอัน โดยส่งหัวข้อผ่าน title และเนื้อหาผ่าน children (ข้อความ, tag p, blockquote)
- children คือเนื้อหาที่อยู่ภายในแท็ก <Card>...</Card> นั่นเอง
- React จะแปลงให้ children เป็น prop อัตโนมัติ ทำให้ Card สามารถ render เนื้อหานี้ได้ตามต้องการ

ผลลัพธ์ที่ได้ (หน้าจอ)

```
---------------------------------------
ข้อมูลส่วนตัว
ชื่อ: ชูวลิต
```

อายุ: 30 ปี
อาชีพ: นักพัฒนาเว็บ

คำคมประจำวัน
“ชีวิตคือการเรียนรู้และเติบโตทุกวัน”

- Card 2 กล่องแบ่งข้อมูลและคำคม โดยใช้ Component เดียวกัน แต่เนื้อหาแตกต่างกันผ่าน props และ children

---

สรุป

- props ช่วยส่งข้อมูลคงที่หรือค่าต่างๆ ให้ Child Component
- children ช่วยส่ง **เนื้อหาที่อยู่ระหว่าง tag** ให้ Component นั้นจัดการและแสดงผล
- การใช้ทั้งคู่ช่วยให้ Component มีความยืดหยุ่นและนำกลับมาใช้ซ้ำได้ง่าย

การใช้ props และ children ใน React แบบเชิงลึก

---

1. Props: การส่งข้อมูลจาก Parent ⟶ Child

แนวคิดเชิงลึก

- **Props คืออินเทอร์เฟซ (Interface)** ที่ Parent Component ใช้สื่อสารกับ Child Component โดยการส่งข้อมูล (data) หรือพฤติกรรม (function) ให้ Child
- React ใช้แนวคิดแบบ **Immutable Data Flow** หรือ **Data Down, Actions Up**
    - ข้อมูลไหลจากบนลงล่าง (Parent ⟶ Child) ผ่าน props
    - Child ไม่สามารถแก้ไข props ได้โดยตรง (props เป็น read-only)
    - หาก Child ต้องการเปลี่ยนข้อมูล ต้องส่งคำขอกลับไปยัง Parent ผ่าน callback function ที่ส่งมาใน props (Action up)
- Props สามารถเป็นชนิดข้อมูลได้ทุกแบบ (string, number, object, function, JSX, Component)

การทำงานของ React กับ Props

- เมื่อ Parent เปลี่ยนค่า props ที่ส่งให้ Child React จะทำการ **Re-render** เฉพาะ Component ที่ได้รับ props นั้นใหม่
- React ใช้ Virtual DOM เปรียบเทียบค่า props เพื่อเพิ่มประสิทธิภาพในการ render

- การส่ง props ที่เปลี่ยนแปลงบ่อยๆ อาจทำให้เกิดการ render ซ้ำบ่อย ถ้าอยาก optimize ใช้ React.memo() หรือ hooks เช่น useMemo() เพื่อ memoize

---

2. Children: เนื้อหาที่ซ้อนใน Component

แนวคิดเชิงลึก

- children เป็น prop พิเศษที่ React สร้างขึ้นอัตโนมัติจากเนื้อหาที่อยู่ในแท็กเปิด-ปิดของ Component
- Children ไม่จำเป็นต้องเป็นแค่ข้อความหรือ element เดียว อาจเป็น array ของ elements, string, number หรือแม้แต่ฟังก์ชัน (render props)
- children ช่วยให้ Component มีความ **flexible** ในการรับเนื้อหาหรือ sub-component โดยไม่ต้องกำหนดชื่อ prop เพิ่ม
- React จะแปลง children เป็น JavaScript object (React elements) ก่อนส่งให้ Component นำไป render

รูปแบบการใช้ children

```
<MyComponent>
  <h1>หัวข้อ</h1>
  <p>เนื้อหา...</p>
</MyComponent>
```

ใน MyComponent จะได้รับ

```
props.children === [
  <h1>หัวข้อ</h1>,
  <p>เนื้อหา...</p>
]
```

---

3. การใช้งาน props และ children ร่วมกัน: ตัวอย่างเชิงลึก

```
function Wrapper({ title, children }) {
  return (
    <section style={{border: '1px solid #ccc', padding: '20px'}}>
      <header><h3>{title}</h3></header>
      <article>{children}</article>
    </section>
  );
}
```

```
// การใช้งาน
function App() {
  return (
    <Wrapper title="ข่าวประชาสัมพันธ์">
      <p>วันนี้จะมีงานสัมมนา React.js เวลา 10 โมงเช้า</p>
      <button onClick={() => alert('จองที่นั่งเรียบร้อย')}>ลงทะเบียน</button>
    </Wrapper>
  );
}
```

- Component Wrapper ใช้ title เป็น prop กำหนดหัวข้อ
- ส่วนเนื้อหาภายใน (children) อาจมีหลาย element รวมถึงปุ่มที่มี event handler
- ทำให้ Wrapper เป็น reusable container component ที่รองรับเนื้อหาแบบ dynamic ได้

---

4. การส่งฟังก์ชันผ่าน Props (Callback Function)

เพื่อให้ Child ส่งข้อมูลหรือ event กลับไปยัง Parent

```
function Child({ onClick }) {
  return <button onClick={() => onClick('Hello from child')}>คลิกฉัน</button>;
}

function Parent() {
  const handleClick = (msg) => alert(msg);

  return <Child onClick={handleClick} />;
}
```

- ฟังก์ชัน handleClick ถูกส่งผ่าน props ไปยัง Child
- Child เรียกใช้ฟังก์ชันนี้เมื่อ event เกิดขึ้น
- รูปแบบนี้คือ **Data Down, Actions Up** ซึ่งเป็น pattern หลักของ React

---

5. Best Practices เกี่ยวกับ Props และ Children

- **กำหนดค่า Default Props** เพื่อให้ Component ทำงานได้แม้ไม่ได้รับ props
- MyComponent.defaultProps = {
- title: 'ไม่มีหัวข้อ',
- };
- ใช้ **PropTypes** เพื่อตรวจสอบชนิดข้อมูล props ในระหว่างพัฒนา

- หลีกเลี่ยงการเปลี่ยนแปลง props ภายใน Child
- หาก children มีหลายตัว ให้ใช้ React.Children API เช่น
  - React.Children.map(props.children, child => ...)
  - React.Children.toArray(props.children) เพื่อจัดการ children แบบปลอดภัย
- อย่าส่งข้อมูลขนาดใหญ่หรือ object ซับซ้อนโดยตรงใน props ถ้าจะส่งบ่อยๆ ให้พิจารณาใช้ Context API หรือ State Management (Redux, Zustand ฯลฯ) แทน

---

6. สรุป

หัวข้อ	คำอธิบายหลัก
props	ข้อมูลหรือฟังก์ชันที่ส่งจาก Parent ⟶ Child, read-only
children	เนื้อหาหรือ Component ที่อยู่ระหว่างแท็ก <Component>...</Component>
Data flow	ข้อมูลไหลทางเดียวจากบนลงล่าง (Unidirectional data flow)
Callback	ฟังก์ชันส่งกลับจาก Child ไป Parent เพื่อแจ้ง event หรือข้อมูล
React.memo	ใช้ optimize การ render เมื่อ props ไม่เปลี่ยน

ตัวอย่าง React แบบเต็มไฟล์ HTML+JSX (ใช้ React + Babel CDN รันง่ายในเบราว์เซอร์) จำนวน 5 ตัวอย่าง โดยเน้นเรื่อง **การใช้ props และ children** พร้อมคำอธิบายและผลลัพธ์แต่ละตัวอย่าง

ตัวอย่างที่ 1: Card Component รับ Props title และ Children

```
<!DOCTYPE html>
<html lang="th">
<head>
  <meta charset="UTF-8" />
  <title>React Example 1</title>
  <script crossorigin src="https://unpkg.com/react@18/umd/react.development.js"></script>
  <script crossorigin src="https://unpkg.com/react-dom@18/umd/react-
dom.development.js"></script>
  <script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
  <style>
    .card { border: 2px solid #007bff; padding: 15px; margin-bottom: 10px; border-radius: 6px;
background: #e9f5ff; }
    h2 { margin: 0 0 10px; color: #0056b3; }
```

```
  </style>
</head>
<body>
  <div id="root"></div>
  <script type="text/babel">
    function Card({ title, children }) {
      return (
        <div className="card">
          <h2>{title}</h2>
          <div>{children}</div>
        </div>
      );
    }

    function App() {
      return (
        <div>
          <Card title="รายละเอียดสินค้า">
            <p>ชื่อสินค้า: สมาร์ทโฟน</p>
            <p>ราคา: 12,000 บาท</p>
          </Card>
          <Card title="ข้อความต้อนรับ">
            <p>ยินดีต้อนรับสู่เว็บไซต์ของเรา!</p>
          </Card>
        </div>
      );
    }

    ReactDOM.createRoot(document.getElementById('root')).render(<App />);
  </script>
</body>
</html>
```

คำอธิบาย

- สร้าง Card ที่รับ props title และ children

- App แสดงสองการ์ด โดยส่งข้อความผ่าน children
- CSS เน้นสีฟ้าสบายตา

ผลรัน

- การ์ดสองอันหัวข้อชัดเจน ข้อความอยู่ในกล่องสวยงาม

ตัวอย่างที่ 2: การส่ง Callback Function ผ่าน Props

```
<!DOCTYPE html>
<html lang="th">
<head>
  <meta charset="UTF-8" />
  <title>React Example 2</title>
  <script crossorigin src="https://unpkg.com/react@18/umd/react.development.js"></script>
  <script crossorigin src="https://unpkg.com/react-dom@18/umd/react-
dom.development.js"></script>
  <script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
</head>
<body>
  <div id="root"></div>
  <script type="text/babel">
    function Button({ onClick, label }) {
      return <button onClick={onClick}>{label}</button>;
    }

    function App() {
      const handleClick = () => alert('คุณคลิกปุ่มแล้ว!');
      return <Button onClick={handleClick} label="คลิกฉัน" />;
    }

    ReactDOM.createRoot(document.getElementById('root')).render(<App />);
  </script>
</body>
</html>
```

คำอธิบาย

- สร้าง Button รับ onClick เป็นฟังก์ชัน และ label เป็นข้อความ

- App ส่งฟังก์ชันแจ้ง alert ไปยัง Button
- เมื่อคลิกปุ่ม alert แสดงข้อความ

ผลรัน

- ปุ่ม "คลิกฉัน" แสดง alert เมื่อคลิก

ตัวอย่างที่ 3: ใช้ Children เป็น Elements หลายตัว

```
<!DOCTYPE html>
<html lang="th">
<head>
  <meta charset="UTF-8" />
  <title>React Example 3</title>
  <script crossorigin src="https://unpkg.com/react@18/umd/react.development.js"></script>
  <script crossorigin src="https://unpkg.com/react-dom@18/umd/react-
dom.development.js"></script>
  <script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
</head>
<body>
  <div id="root"></div>
  <script type="text/babel">
    function Container({ children }) {
      return <div style={{ border: '2px dotted green', padding: '10px' }}>{children}</div>;
    }

    function App() {
      return (
        <Container>
          <h1>หัวข้อใหญ่</h1>
          <p>นี่คือย่อหน้าที่หนึ่ง</p>
          <p>นี่คือย่อหน้าที่สอง</p>
          <button onClick={() => alert('ปุ่มใน children คลิกแล้ว!')}>กดฉัน</button>
        </Container>
      );
    }
```

```
    ReactDOM.createRoot(document.getElementById('root')).render(<App />);
  </script>
</body>
</html>
```

คำอธิบาย

- Container รับ children ที่เป็น Elements หลายตัว
- แสดงกล่องเส้นประสีเขียวล้อมรอบเนื้อหา
- children สามารถมีปุ่มที่มี event handler

ผลรัน

- แสดงหัวข้อ ย่อหน้า 2 ย่อหน้า และปุ่มในกล่องเส้นประ
- กดปุ่มแสดง alert

ตัวอย่างที่ 4: Default Props กับ Props Validation (ใช้ PropTypes)

```
<!DOCTYPE html>
<html lang="th">
<head>
  <meta charset="UTF-8" />
  <title>React Example 4</title>
  <script crossorigin src="https://unpkg.com/react@18/umd/react.development.js"></script>
  <script crossorigin src="https://unpkg.com/react-dom@18/umd/react-
dom.development.js"></script>
  <script src="https://unpkg.com/prop-types/prop-types.min.js"></script>
  <script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
</head>
<body>
  <div id="root"></div>
  <script type="text/babel">
    function Greeting({ name }) {
      return <h3>สวัสดี, {name}!</h3>;
    }

    Greeting.defaultProps = {
      name: 'ผู้เยี่ยมชม',
    };
```

```
    Greeting.propTypes = {
      name: PropTypes.string,
    };

    function App() {
      return (
        <div>
          <Greeting name="ชูวลิต" />
          <Greeting />
        </div>
      );
    }

    ReactDOM.createRoot(document.getElementById('root')).render(<App />);
  </script>
</body>
</html>
```

คำอธิบาย

- กำหนด default props ให้ name เป็น 'ผู้เยี่ยมชม'
- ใช้ PropTypes ตรวจสอบว่า name ต้องเป็น string
- แสดงข้อความทักทายสองครั้ง ครั้งที่สองไม่มี name เลยใช้ default

ผลรัน

- แสดง
    - สวัสดี, ชูวลิต!
    - สวัสดี, ผู้เยี่ยมชม!

ตัวอย่างที่ 5: ส่ง Props เป็น Object และ Children เป็น Function (Render Props)

```
<!DOCTYPE html>
<html lang="th">
<head>
  <meta charset="UTF-8" />
  <title>React Example 5</title>
  <script crossorigin src="https://unpkg.com/react@18/umd/react.development.js"></script>
```

```
  <script crossorigin src="https://unpkg.com/react-dom@18/umd/react-
dom.development.js"></script>
  <script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
</head>
<body>
  <div id="root"></div>
  <script type="text/babel">
    // Component ที่รับ user object และ children เป็นฟังก์ชัน
    function UserProfile({ user, children }) {
      return (
        <div style={{ border: '1px solid gray', padding: '10px', maxWidth: '300px' }}>
          {children(user)}
        </div>
      );
    }

    function App() {
      const user = { name: 'ชูวลิต', age: 30, email: 'chouvalit@example.com' };

      return (
        <UserProfile user={user}>
          {(user) => (
            <div>
              <h2>{user.name}</h2>
              <p>อายุ: {user.age}</p>
              <p>อีเมล: {user.email}</p>
            </div>
          )}
        </UserProfile>
      );
    }

    ReactDOM.createRoot(document.getElementById('root')).render(<App />);
  </script>
```

```
</body>
</html>
```

คำอธิบาย

- UserProfile รับ props user เป็น object
- children เป็นฟังก์ชัน (render props) ที่รับ user แล้ว return JSX
- ทำให้ Component นี้ยืดหยุ่นสูง สามารถกำหนด UI ได้จากภายนอก

ผลรัน

- แสดงข้อมูลชื่อ อายุ อีเมล ของผู้ใช้ในกรอบสวยงาม

ตัวอย่างบูรณาการที่รวม

- อธิบาย JSX ว่าเป็นอะไร และทำไมต้องใช้
- แสดงการใช้ JSX แสดงข้อความ ตัวแปร และ expression
- สร้าง Functional Component
- ใช้ props และ children ส่งข้อมูลระหว่าง Components

โดยจะเขียนเป็นไฟล์ HTML ที่รันผ่าน CDN + Babel ได้เลย พร้อมคำอธิบายและผลรันด้วย

ตัวอย่าง React บูรณาการ (JSX + Functional Component + Props + Children)

```
<!DOCTYPE html>
<html lang="th">
<head>
  <meta charset="UTF-8" />
  <title>React JSX + Functional Component + Props + Children</title>
  <script crossorigin src="https://unpkg.com/react@18/umd/react.development.js"></script>
  <script crossorigin src="https://unpkg.com/react-dom@18/umd/react-
dom.development.js"></script>
  <script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
  <style>
    body { font-family: "Segoe UI", Tahoma, Geneva, Verdana, sans-serif; padding: 20px;
background: #f9f9f9; }
    .card { background: white; border-radius: 8px; box-shadow: 0 2px 6px rgba(0,0,0,0.15);
padding: 20px; max-width: 450px; margin: auto; }
    h2 { color: #2c3e50; }
    p { color: #555; font-size: 1.1rem; }
  </style>
```

```
</head>
<body>
  <div id="root"></div>

  <script type="text/babel">

    // 1. JSX คือ syntax ที่เหมือน HTML เขียนใน JavaScript เพื่อสร้าง React Element
    // 2. JSX จำเป็นเพราะช่วยให้เราเขียน UI ง่ายขึ้นและดูเป็น declarative

    // Functional Component ที่รับ props และ children
    function Card({ title, children }) {
      // แสดงผล title (props) และเนื้อหา children
      return (
        <div className="card">
          <h2>{title}</h2>
          <div>{children}</div>
        </div>
      );
    }

    // App component ที่ใช้ JSX แสดงข้อความ ตัวแปร และ expression
    function App() {
      const userName = "ชูวลิต";
      const isMember = true;

      // expression ใน JSX ใช้ {}
      return (
        <Card title="ตัวอย่าง React JSX & Props">
          {/* ข้อความปกติ */}
          <p>สวัสดีครับ! นี่คือข้อความแสดงผลด้วย JSX</p>

          {/* แสดงตัวแปร */}
          <p>ชื่อผู้ใช้: {userName}</p>
```

```
        {/* แสดง expression */}
        <p>สถานะสมาชิก: {isMember ? "สมาชิก VIP" : "ผู้เยี่ยมชม"}</p>

        {/* ส่ง children เป็น JSX หลายบรรทัด */}
        <div>
          <p>วันนี้วันที่: {new Date().toLocaleDateString()}</p>
          <p>เลขสุ่ม: {Math.floor(Math.random() * 100)}</p>
        </div>
      </Card>
    );
  }

  ReactDOM.createRoot(document.getElementById('root')).render(<App />);

 </script>
</body>
</html>
```

คำอธิบาย

- **JSX** คือ syntax ที่เขียนคล้าย HTML ใน JavaScript เพื่อสร้าง UI
- JSX ต้องถูกแปลงเป็น React.createElement โดย Babel (ที่โหลดผ่าน CDN ในตัวอย่างนี้)
- Functional Component Card รับ props title และ children เพื่อแสดงข้อมูลและเนื้อหาภายใน
- ใน App ใช้ JSX แสดงข้อความธรรมดา, แสดงตัวแปร userName, ใช้ expression แบบ ternary เพื่อแสดงสถานะ
- children ของ Card ส่งเป็น JSX หลายบรรทัด เช่น แสดงวันที่ปัจจุบันและเลขสุ่ม
- ใช้ {} ใน JSX เพื่อแทรกตัวแปรหรือ expression

ผลการรัน

- แสดงกล่อง Card หัวข้อ "ตัวอย่าง React JSX & Props"
- ข้อความต้อนรับ "สวัสดีครับ! นี่คือข้อความแสดงผลด้วย JSX"
- แสดงชื่อผู้ใช้: ชูวลิต
- แสดงสถานะสมาชิก: สมาชิก VIP (เพราะ isMember = true)
- แสดงวันที่ปัจจุบัน และเลขสุ่ม (เปลี่ยนทุกครั้งที่โหลดหน้าใหม่)

ตัวอย่างนี้จะไม่ใช้ลิงก์โหลด React, ReactDOM หรือ Babel จากเว็บอื่นเลย แต่จะเป็นตัวอย่างแบบ **ไฟล์ HTML+JSX รวมกันในไฟล์เดียวที่ทำงานได้ทันที (แบบ Pure React โดยไม่ใช้ CDN)** — ซึ่งทำได้ยากถ้าไม่ใช้ตัวช่วยแปลง JSX เพราะเบราว์เซอร์ไม่เข้าใจ JSX ตรง ๆ
แต่เพื่อให้ตรงกับคำขอ "ไม่ link ไปเว็บอื่น" ผมจะเขียนตัวอย่าง React แบบ pure JavaScript (ไม่ใช้ JSX) ซึ่งทำงานได้ในเบราว์เซอร์โดยตรงเลย

ตัวอย่าง React แบบไม่ใช้ JSX (Pure JS) + Functional Component + props + children

```
<!DOCTYPE html>
<html lang="th">
<head>
<meta charset="UTF-8" />
<title>React Example Without JSX and External Links</title>
<style>
  body { font-family: Arial, sans-serif; padding: 20px; background: #fafafa; }
  .card { background: white; padding: 20px; max-width: 400px; margin: auto; border-radius: 8px;
box-shadow: 0 2px 8px rgba(0,0,0,0.1); }
  h2 { color: #333; }
  p { color: #555; }
</style>
</head>
<body>
  <div id="root"></div>

  <script>
    // สมมติว่า React และ ReactDOM โค้ดนี้เขียนแบบจำลอง React แบบง่าย (Minimal)
    // เพราะไม่โหลด React จริงๆ ผมจะสร้างฟังก์ชันจำลองง่ายๆ สำหรับประกอบ UI

    // สร้าง function สร้าง element แบบ React.createElement
    function createElement(type, props, ...children) {
      return { type, props: props || {}, children };
    }

    // ฟังก์ชันแปลง element tree เป็น DOM จริง (recursive)
    function renderElement(node) {
```

```
  if (typeof node === 'string' || typeof node === 'number') {
    return document.createTextNode(node);
  }

  if (typeof node.type === 'function') {
    // ถ้าเป็น function component
    return renderElement(node.type({...node.props, children: node.children}));
  }

  const el = document.createElement(node.type);

  for (const [key, value] of Object.entries(node.props)) {
    if (key.startsWith('on') && typeof value === 'function') {
      const eventType = key.substring(2).toLowerCase();
      el.addEventListener(eventType, value);
    } else if (key === 'className') {
      el.setAttribute('class', value);
    } else {
      el.setAttribute(key, value);
    }
  }

  node.children.forEach(child => {
    el.appendChild(renderElement(child));
  });

  return el;
}

// Functional Component Card รับ props title และ children
function Card({ title, children }) {
  return createElement(
    'div',
    { className: 'card' },
```

```
        createElement('h2', null, title),
        createElement('div', null, ...children)
      );
    }

    // Functional Component App ใช้ JSX แบบ createElement แสดงข้อความ, ตัวแปร และ
expression
    function App() {
      const userName = "ชูวลิต";
      const isMember = true;

      return createElement(
        Card,
        { title: "React แบบ Pure JS (No JSX, No CDN)" },
        createElement('p', null, 'สวัสดีครับ! นี่คือข้อความแสดงผลแบบ pure JS'),
        createElement('p', null, 'ชื่อผู้ใช้: ', userName),
        createElement('p', null, 'สถานะสมาชิก: ', isMember ? "สมาชิก VIP" : "ผู้เยี่ยมชม"),
        createElement(
          'div',
          null,
          createElement('p', null, 'วันที่ปัจจุบัน: ', new Date().toLocaleDateString()),
          createElement('p', null, 'เลขสุ่ม: ', Math.floor(Math.random() * 100))
        )
      );
    }

    // Render App ลงใน root
    const root = document.getElementById('root');
    root.appendChild(renderElement(createElement(App)));

  </script>
</body>
</html>
```

อธิบาย

- ตัวอย่างนี้ **ไม่ใช้ JSX** แต่สร้าง React Element ด้วย createElement() แบบเดียวกับที่ React ใช้ในเบื้องหลัง
- จำลองการทำงานของ React แบบง่ายสุด ๆ เพื่อให้รันในเบราว์เซอร์โดยตรง
- มี Functional Component Card ที่รับ props และ children
- Component App แสดงข้อความ, ตัวแปร และ expression ต่าง ๆ แบบที่ React ทำได้
- ไม่โหลด React, ReactDOM, Babel ใด ๆ จาก CDN หรือเว็บอื่นเลย
- สามารถเปิดไฟล์นี้ในเบราว์เซอร์ทั่วไปได้ทันที โดยไม่ต้องติดตั้งอะไรเพิ่มเติม

ผลลัพธ์ที่ได้

- กล่องสวย ๆ พร้อมหัวข้อ
- แสดงข้อความต้อนรับ
- แสดงชื่อผู้ใช้
- แสดงสถานะสมาชิก
- แสดงวันที่ปัจจุบันและเลขสุ่ม

สรุป

ใน React การสร้างส่วนติดต่อผู้ใช้ (UI) ที่มีประสิทธิภาพและยืดหยุ่นเริ่มต้นจากสิ่งที่เรียกว่า **JSX (JavaScript XML)** ซึ่งเป็น Syntax ที่อนุญาตให้เราเขียนโค้ดคล้าย HTML ภายใน JavaScript ได้อย่างเป็นธรรมชาติ JSX ช่วยให้เราสามารถฝังตัวแปร คำสั่งคำนวณ หรือ expression ลงใน UI ได้อย่างง่ายดาย และเป็นหัวใจหลักของการพัฒนา Component ซึ่งเป็นโครงสร้างพื้นฐานของ React แอป ในบทนี้ผู้อ่านจะได้เรียนรู้ตั้งแต่แนวคิดของ JSX การแสดงผลข้อมูล การเขียน Component แบบฟังก์ชัน (Functional Component) และการส่งข้อมูลระหว่างกันด้วย props รวมถึงการใช้ children เพื่อควบคุมโครงสร้างภายใน Component อย่างยืดหยุ่น ความเข้าใจพื้นฐานเหล่านี้จะเป็นก้าวสำคัญในการสร้างระบบที่มีโครงสร้างดี ดูแลรักษาง่าย และสามารถนำกลับมาใช้ซ้ำได้ในระดับมืออาชีพ.

บทที่ 3

การจัดการ State และ Event

(State and Event Management)

เนื้อหา

- เบื้องต้นการจัดการ State และ Event
- เชิงลึก: การจัดการ State และ Event ใน React
- การใช้ useState() สำหรับจัดการ State ภายใน Component
- เชิงลึกที่สุด: การใช้ useState() สำหรับจัดการ State ใน React
- การจัดการเหตุการณ์ (Event Handling) ใน React
- การจัดการเหตุการณ์ (Event Handling) ใน React แบบเจาะลึก
- เทคนิคการผูกฟังก์ชันกับ Event ใน React
- ข้อมูลเชิงลึกและเทคนิค การผูกฟังก์ชันกับ event ใน React
- เชิงลึกการผูกฟังก์ชันกับ Event ใน React
- การใช้ State แบบ Array และ Object ใน React
- เทคนิคขั้นสูงและ Best Practices
- ตัวอย่าง React แบบบูรณาการ

บทนำ: การจัดการ State และ Event

การสร้างแอปพลิเคชันที่สามารถตอบสนองต่อการกระทำของผู้ใช้อย่างมีประสิทธิภาพ จำเป็นต้องอาศัยกลไกในการจัดเก็บและจัดการข้อมูลภายใน Component ซึ่งใน React เราใช้สิ่งที่เรียกว่า **State** เพื่อเก็บข้อมูลเหล่านี้ เช่น ค่าที่ผู้ใช้กรอกในฟอร์ม สถานะของปุ่ม toggle หรือจำนวนที่เพิ่มขึ้นของ counter ต่างๆ โดย React มีฟังก์ชันสำคัญที่ชื่อว่า useState() ซึ่งเป็น Hook พื้นฐานสำหรับจัดการ State ใน Component แบบฟังก์ชัน ช่วยให้เราสามารถเพิ่มความสามารถแบบ dynamic ให้กับ UI ได้อย่างง่ายดาย

นอกจากการจัดเก็บข้อมูลแล้ว อีกสิ่งหนึ่งที่มีบทบาทสำคัญไม่แพ้กันคือ **การจัดการ Event** เช่น การคลิกปุ่ม (onClick) การพิมพ์ข้อความ (onChange) หรือการส่งฟอร์ม (onSubmit) การจัดการ event ใน React มีรูปแบบที่เข้าใจง่ายแต่ทรงพลัง โดยอนุญาตให้เราสามารถผูกฟังก์ชันเข้ากับเหตุการณ์ และควบคุมลำดับขั้นของการตอบสนองได้อย่างยืดหยุ่น ช่วยให้เราสร้าง UI ที่มีปฏิสัมพันธ์ได้อย่างสมบูรณ์แบบ

ในบทนี้ ผู้อ่านจะได้เรียนรู้ **เทคนิคการเขียนฟังก์ชันเพื่อจัดการ event** พร้อมการส่งพารามิเตอร์ไปยังฟังก์ชันเหล่านั้น ซึ่งเป็นพื้นฐานสำคัญในการควบคุมพฤติกรรมของแอป เช่น การลบรายการใดรายการหนึ่งจาก list, การเปลี่ยนค่าใน dropdown, หรือการเปิด modal เฉพาะที่ผู้ใช้เลือก นอกจากนี้ยังมีการอธิบายการใช้ state ที่เป็นชนิดข้อมูลซับซ้อน เช่น array หรือ object พร้อมตัวอย่างโค้ดที่เข้าใจง่ายและสามารถนำไปประยุกต์ใช้ได้จริง

React จะไม่สามารถแสดงศักยภาพที่แท้จริงได้ หากผู้พัฒนาไม่เข้าใจความสัมพันธ์ระหว่าง **State กับ Event** เพราะทั้งสองคือกลไกหลักที่ทำให้ UI ใน React “มีชีวิต” การเปลี่ยนแปลง State จะทำให้ Component ทำงานใหม่ (re-render) โดยอัตโนมัติ ซึ่งช่วยลดภาระในการจัดการ DOM ด้วยตนเอง และทำให้การพัฒนา UI ที่ซับซ้อนกลายเป็นเรื่องง่ายและปลอดภัย

เมื่อเรียนจบบทนี้ ผู้อ่านจะสามารถสร้าง Component ที่โต้ตอบกับผู้ใช้ได้อย่างสมบูรณ์แบบ เข้าใจการใช้งาน useState() อย่างลึกซึ้ง และสามารถควบคุม event ได้อย่างมีประสิทธิภาพ ซึ่งจะเป็นทักษะสำคัญในการสร้างแอปที่มีการโต้ตอบหลากหลาย และเป็นรากฐานที่มั่นคงสำหรับการเรียนรู้เรื่องการจัดการข้อมูลระดับสูงในบทถัดไป เช่น Context API, Reducer หรือ State Management Framework อย่าง Redux.

เบื้องต้นการจัดการ State และ Event

1. การใช้ useState() สำหรับจัดการ State ภายใน Component

- useState เป็น Hook ที่ React ให้มาเพื่อจัดการ **สถานะ (state)** ภายใน Functional Component
- State คือข้อมูลที่เปลี่ยนแปลงได้และทำให้ React รีเรนเดอร์ UI ใหม่เมื่อข้อมูลนั้นเปลี่ยน
- การใช้งาน useState จะคืนค่ามาเป็น **คู่ของ [ค่า state, ฟังก์ชันอัปเดต]**

ตัวอย่างการประกาศ useState:

```
const [count, setCount] = React.useState(0);
```

- count คือค่าปัจจุบันของ state
- setCount คือฟังก์ชันใช้เปลี่ยนค่า state
- ค่าที่ใส่ใน useState(0) คือค่าเริ่มต้น

2. การจัดการเหตุการณ์ (Event Handling) เช่น onClick, onChange

- React จัดการ event แบบคล้าย DOM ปกติ แต่จะใช้ **camelCase** แทน เช่น onClick แทน onclick
- เราสามารถผูกฟังก์ชันเข้ากับ event ได้ง่าย ๆ เช่น

```
<button onClick={handleClick}>กดฉัน</button>
```

- ฟังก์ชันจะรับออบเจกต์ event เป็นพารามิเตอร์

3. เทคนิคการผูกฟังก์ชันกับ event และการส่งค่าไปพร้อม event

- ถ้าต้องการส่งค่าเพิ่มเติมไปใน event handler เราสามารถใช้ **arrow function** ห่อไว้ เช่น

<button onClick={() => handleClick(10)}>กดฉัน</button>

- หรือถ้าไม่ใช้ arrow function จะส่ง event อัตโนมัติให้เป็นพารามิเตอร์แรกเสมอ

---

4. การใช้ state แบบ array และ object

- State สามารถเก็บค่าได้หลากหลาย เช่น ตัวแปรแบบ array หรือ object
- การอัปเดต state แบบ array หรือ object ต้องทำ **แบบ immutably**
- ใช้ spread operator (...) เพื่อคัดลอกค่าเดิมแล้วเปลี่ยนเฉพาะค่าที่ต้องการ

ตัวอย่าง

```
const [items, setItems] = React.useState([]);

function addItem(newItem) {
  setItems(prevItems => [...prevItems, newItem]);
}
```

หรือกับ object

```
const [user, setUser] = React.useState({ name: '', age: 0 });

function updateName(newName) {
  setUser(prevUser => ({ ...prevUser, name: newName }));
}
```

ตัวอย่างโปรแกรมแบบเต็มไฟล์

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

function App() {
  // State แบบ number
  const [count, setCount] = useState(0);

  // State แบบ array
  const [tasks, setTasks] = useState(['ทำงานบ้าน', 'ซื้อของ']);

  // State แบบ object
```

```
const [user, setUser] = useState({ name: '', age: '' });

// เพิ่ม task ใหม่ใน array state
const addTask = () => {
  const newTask = prompt('เพิ่มงานใหม่:');
  if (newTask) {
    setTasks(prevTasks => [...prevTasks, newTask]);
  }
};

// อัปเดตชื่อใน object state
const handleNameChange = (e) => {
  const newName = e.target.value;
  setUser(prevUser => ({ ...prevUser, name: newName }));
};

// อัปเดตอายุใน object state
const handleAgeChange = (e) => {
  const newAge = e.target.value;
  setUser(prevUser => ({ ...prevUser, age: newAge }));
};

return (
  <div style={{ padding: 20 }}>
    <h1>ตัวอย่าง React useState และ Event</h1>

    {/* State number */}
    <p>จำนวนครั้งที่กด: {count}</p>
    <button onClick={() => setCount(count + 1)}>กดเพิ่ม</button>

    <hr />

    {/* State array */}
    <h2>รายการงาน</h2>
```

```
      <ul>
        {tasks.map((task, index) => <li key={index}>{task}</li>)}
      </ul>
      <button onClick={addTask}>เพิ่มงานใหม่</button>

      <hr />

      {/* State object */}
      <h2>ข้อมูลผู้ใช้</h2>
      <label>
        ชื่อ:
        <input type="text" value={user.name} onChange={handleNameChange} />
      </label>
      <br />
      <label>
        อายุ:
        <input type="number" value={user.age} onChange={handleAgeChange} />
      </label>

      <p>สวัสดี, {user.name}! คุณอายุ {user.age} ปี</p>
    </div>
  );
}

export default App;
```

คำอธิบายผลการรัน

- มีปุ่มกดเพิ่มค่า count ทีละ 1 โดยแสดงจำนวนครั้งที่กดบนหน้าจอ
- แสดงรายการงาน (array) ที่เก็บใน state และปุ่มเพิ่มงานใหม่โดยใช้ prompt รับ input
- ฟอร์มกรอกข้อมูลชื่อและอายุ (object) ที่อัปเดตและแสดงผลสดทันทีที่แก้ไข
- ทุกการเปลี่ยนแปลงข้อมูลใน state จะทำให้ UI รีเฟรชโดยอัตโนมัติ

เชิงลึก: การจัดการ State และ Event ใน React

---

1. State ใน React คืออะไร? ทำไมต้องมี State?

- **State** คือข้อมูลภายใน Component ที่สามารถเปลี่ยนแปลงได้
- React จะใช้ State เป็นตัวกระตุ้นให้ Component รีเรนเดอร์ (render) ใหม่ทุกครั้งที่ State เปลี่ยน
- ทำให้ UI สะท้อนสถานะล่าสุดของข้อมูลเสมอ
- แตกต่างจาก **props** ที่รับข้อมูลจาก Component ภายนอกและไม่ควรเปลี่ยนแปลงใน Component นั้น ๆ

---

2. useState() Hook ทำงานอย่างไร?

- useState คือ Hook ที่ให้ Functional Component มี “state” เหมือน Class Component
- ฟังก์ชัน useState(initialValue) คืนค่าเป็นอาร์เรย์ 2 ตัว:
    1. ค่าปัจจุบันของ state
    2. ฟังก์ชันเปลี่ยนแปลง state
- เมื่อเรียกฟังก์ชันเปลี่ยนแปลง state (setState), React จะทำการบันทึกค่าใหม่และ “กำหนดให้ Component รีเรนเดอร์ใหม่”
- **React จะทำ batch update และ optimize rendering** เพื่อให้ประสิทธิภาพดีขึ้น

---

3. หลักการของ Immutable State

- React ต้องการให้ state **immutable** หรือเปลี่ยนค่าโดยไม่แก้ไขตัวแปรเดิมตรง ๆ
- เพราะถ้าแก้ไข state โดยตรง (mutate) React อาจไม่รู้ว่าค่าเปลี่ยนไปแล้ว จึงไม่รีเรนเดอร์
- ตัวอย่างที่ถูกต้องกับ array หรือ object คือใช้ **spread operator (...)** เพื่อทำสำเนา แล้วแก้ไขสำเนา
- React จะตรวจสอบการเปลี่ยนแปลงโดยการเช็คว่า “ออบเจกต์ใหม่” กับ “ออบเจกต์เก่า” ต่างกันหรือไม่ (โดยอ้างอิงจาก reference)

---

4. Event Handling ใน React

- React ใช้ระบบ event ที่เรียกว่า **Synthetic Event** ซึ่งเป็น wrapper ของ DOM event
- Synthetic Event ทำให้ event ทำงานเหมือนกันในทุกเบราว์เซอร์ (cross-browser compatibility)
- Event handler ใน React เขียนเป็นฟังก์ชัน JavaScript (ไม่ใช่ string) และใช้ **camelCase** เช่น onClick
- สามารถผูกฟังก์ชันเข้ากับ event ได้ง่าย ทำให้ code อ่านง่ายและดูแลได้สะดวก
- การจัดการ event ที่ซับซ้อน เช่น ส่งพารามิเตอร์, การจัดการ event propagation, สามารถทำได้ผ่าน JavaScript ธรรมดา

5. การส่งค่าและฟังก์ชันผ่าน props

- เราสามารถส่งฟังก์ชัน event handler เป็น props ไปยัง Component ลูกได้
- Component ลูกเรียกใช้ฟังก์ชันเหล่านั้นเมื่อเกิดเหตุการณ์ เพื่อให้ Component แม่ (Parent) สามารถจัดการ state ได้ (lifting state up)
- ช่วยให้โครงสร้างโปรแกรมแยกชั้นชัดเจนและควบคุมข้อมูลได้ดีขึ้น

---

6. การใช้ useState กับ array และ object

- ถ้าใช้ array หรือ object เป็น state ต้องอัปเดตแบบ immutable
- ตัวอย่างการเพิ่มข้อมูลลง array

```
setArray(prevArray => [...prevArray, newItem]);
```

- ตัวอย่างการแก้ไข object

```
setObject(prevObj => ({ ...prevObj, keyToUpdate: newValue }));
```

- หากลืมทำแบบนี้ อาจเกิด bug ที่ React ไม่รีเรนเดอร์ หรือข้อมูลเพี้ยน

---

7. ข้อควรระวังในการใช้ useState

- **ไม่ควรแก้ไข state ตรง ๆ** เช่น state.count = 5 เพราะ React จะไม่รู้ว่ามีการเปลี่ยนแปลง
- การเรียก setState แบบ **ซิงโครนัส** อาจไม่ได้ผลตามที่คาด เพราะ React อาจรวมการอัปเดตหลายครั้ง (batching)
- หากต้องการใช้ค่าปัจจุบันของ state ในการอัปเดต ควรใช้รูปแบบฟังก์ชัน

```
setCount(prevCount => prevCount + 1);
```

- หลีกเลี่ยงการเรียก setState ในลูปหรือแบบที่ทำให้เกิดการ render ซ้ำ ๆ ไม่สิ้นสุด (infinite loop)

---

8. การใช้ useState กับฟอร์มและ input

- React ทำงานแบบ controlled component คือ **ค่าของ input ถูกควบคุมโดย state**
- input จะส่ง event onChange เพื่ออัปเดต state เสมอ
- React แสดงค่าใน input ตาม state เพื่อความสอดคล้องเสมอ

---

9. การทำงานของ React กับ Event Loop และ Rendering Cycle

- React รวบรวมการเปลี่ยนแปลง state หลาย ๆ ครั้งใน event loop เดียวกัน เพื่อเพิ่มประสิทธิภาพ
- หลังจากทุก state update React จะคำนวณ Virtual DOM ใหม่ และเปรียบเทียบกับ Virtual DOM เก่า (reconciliation)

- React จะอัปเดต DOM จริงเฉพาะส่วนที่เปลี่ยนแปลงจริง ๆ (diffing) เพื่อประหยัดทรัพยากร

สรุป

หัวข้อ	คำอธิบายสั้น ๆ
State	ข้อมูลภายใน Component ที่เปลี่ยนแปลงได้และทำให้ UI รีเฟรช
useState	Hook สำหรับสร้างและจัดการ State ใน Functional Component
Immutable	การเปลี่ยนแปลง State แบบไม่แก้ไขข้อมูลเดิมตรง ๆ เพื่อให้ React ทำงานถูกต้อง
Synthetic Event	ระบบจัดการ event ของ React ที่รองรับทุกเบราว์เซอร์
Event Handling	การจับและจัดการเหตุการณ์ เช่น การคลิก ป้อนข้อมูล
Props	ตัวส่งข้อมูลและฟังก์ชันระหว่าง Components เพื่อการสื่อสารและควบคุม
Controlled Component	การควบคุม input ผ่าน State ของ React

การใช้ useState() สำหรับจัดการ State ภายใน Component

---

1. ความหมายของ useState()

- useState คือ **Hook** ที่ React ให้มาเพื่อให้ Functional Component สามารถมี **state** ได้ (ก่อน React 16.8 Functional Component ไม่มี state)
- เป็นฟังก์ชันที่เรียกใช้ภายใน Component เพื่อสร้างตัวแปรสถานะ (state variable) พร้อมกับฟังก์ชันสำหรับเปลี่ยนแปลงค่านั้น

---

2. รูปแบบการใช้งาน

```
const [state, setState] = useState(initialValue);
```

- state คือค่าปัจจุบันของสถานะ
- setState คือฟังก์ชันที่ใช้เปลี่ยนแปลงสถานะ (เรียกแล้ว React จะรีเรนเดอร์ Component ใหม่)
- initialValue คือค่าตั้งต้นที่ state จะถูกตั้งตอน Component ถูกสร้างครั้งแรก

---

3. หลักการทำงาน

- เมื่อเรียก setState(newValue) หรือ setState(callback) React จะบันทึกค่าที่เปลี่ยน และทำให้ Component รีเรนเดอร์ใหม่โดยใช้ค่าใหม่ของ state

- React ทำการเปรียบเทียบ Virtual DOM ก่อนและหลัง render เพื่ออัปเดต DOM จริงเฉพาะส่วนที่เปลี่ยน
- useState จะเก็บสถานะแยกกันสำหรับแต่ละ Component instance

---

4. การใช้ค่าปัจจุบันของ State ในการอัปเดต

ถ้าการอัปเดต state ต้องอิงกับค่าปัจจุบัน ควรใช้รูปแบบ **callback function** เพื่อให้ได้ค่าล่าสุดเสมอ

เช่น

setCount(prevCount => prevCount + 1);

แทนการเขียน

setCount(count + 1);

เพราะ React อาจทำ batch update ทำให้ count ยังไม่อัปเดตในทันที

---

5. ตัวอย่างโค้ดใช้งาน useState แบบง่าย ๆ

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

function Counter() {
  const [count, setCount] = useState(0); // กำหนด state count เริ่มต้น 0

  const increment = () => {
    setCount(prevCount => prevCount + 1); // เพิ่ม count ทีละ 1 โดยใช้ callback
  };

  const decrement = () => {
    setCount(prevCount => prevCount - 1); // ลด count ทีละ 1
  };

  return (
    <div>
      <h2>Counter: {count}</h2>
      <button onClick={increment}>เพิ่ม</button>
      <button onClick={decrement}>ลด</button>
    </div>
  );
}
```

```
export default Counter;
```

---

6. จุดสำคัญและข้อควรระวัง

- **อย่าแก้ไข state โดยตรง** เช่น count = 5; เพราะ React จะไม่ทราบและไม่รีเรนเดอร์
- เรียก setState เท่านั้นเพื่อเปลี่ยนค่า state
- ถ้า state เป็น object หรือ array ต้องอัปเดตแบบ immutable (ใช้ spread operator)
- ค่า initialValue จะถูกใช้แค่ครั้งแรกตอน Component สร้างครั้งแรกเท่านั้น (ถ้าจะตั้งค่าเริ่มต้นแบบคำนวณหนัก ใช้รูปแบบ callback เช่น useState(() => expensiveComputation()) เพื่อให้ประสิทธิภาพดีกว่า)
- useState สามารถใช้ได้หลายครั้งใน Component เดียวกัน เพื่อเก็บ state หลายตัวแยกกัน

---

7. ตัวอย่างใช้ useState กับ object

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

function UserForm() {
  const [user, setUser] = useState({ name: '', email: '' });

  const handleNameChange = e => {
    setUser(prevUser => ({ ...prevUser, name: e.target.value }));
  };

  const handleEmailChange = e => {
    setUser(prevUser => ({ ...prevUser, email: e.target.value }));
  };

  return (
    <div>
      <input value={user.name} onChange={handleNameChange} placeholder="ชื่อ" />
      <input value={user.email} onChange={handleEmailChange} placeholder="อีเมล" />
      <p>ชื่อ: {user.name}</p>
      <p>อีเมล: {user.email}</p>
    </div>
  );
```

```
}

export default UserForm;
```

สรุป

หัวข้อ	คำอธิบาย
useState คือ	Hook สำหรับสร้างและจัดการ State ใน Functional Component
การใช้งาน	คืนค่าเป็น [state, setState] ใช้เพื่ออ่านและเปลี่ยนแปลง state
การอัปเดต state	ต้องเรียก setState เพื่อให้ React รีเรนเดอร์
อัปเดต state ที่ขึ้นกับค่าปัจจุบัน	ใช้ callback เช่น setCount(prev => prev + 1)
การจัดการ state แบบ object/array	ใช้ spread operator เพื่อเปลี่ยนแบบ immutable
ข้อควรระวัง	หลีกเลี่ยงแก้ไข state ตรง ๆ, initialValue ใช้ครั้งเดียวตอน mount

---

เชิงลึกที่สุด: การใช้ useState() สำหรับจัดการ State ใน React

---

1. แนวคิดของ State ใน React และเหตุผลที่มี useState

- React เป็น Library สำหรับสร้าง UI แบบประกอบด้วย Component
- แต่ละ Component ต้องเก็บข้อมูลสถานะ (State) ของตัวเอง เพื่อบอกว่า UI ควรแสดงอย่างไรในเวลานั้น
- **Functional Component** ก่อน React 16.8 ไม่มี state ในตัวเอง (เป็น pure function)
- React จึงเพิ่ม **Hooks** ให้ Functional Component สามารถเก็บและจัดการ state ได้
- useState เป็น Hook ที่พื้นฐานที่สุดสำหรับจัดการข้อมูลที่เปลี่ยนแปลงได้ใน Component

---

2. รูปแบบการใช้งาน useState

const [state, setState] = useState(initialValue);

- state — ตัวแปรเก็บสถานะปัจจุบัน
- setState — ฟังก์ชันเปลี่ยนแปลงสถานะและสั่งให้ React รีเรนเดอร์ Component ใหม่
- initialValue — ค่าตั้งต้นเมื่อ Component ถูก Mount ครั้งแรก
- **สำคัญ**: initialValue จะถูกใช้แค่ตอน mount เท่านั้น (ถ้า Component rerender จะไม่ใช้ค่าเริ่มต้นซ้ำ)

---

3. การทำงานภายในของ useState (ภายใน React)

- React เก็บข้อมูล state สำหรับแต่ละ Component instance ใน **Internal Fiber Node**
- เมื่อ Component เรียก useState React จะดึงค่าปัจจุบันของ state จาก Fiber Tree
- ถ้ามีการเรียก setState React จะบันทึกค่าใหม่และเพิ่มการอัปเดตเข้าไปใน queue ของ Fiber
- React จะ **schedule update** ให้ Component นั้น ๆ รีเรนเดอร์ (Concurrent Mode ช่วยจัดการลำดับการ render อย่างมีประสิทธิภาพ)
- React ใช้ **Diffing Algorithm** เปรียบเทียบ Virtual DOM เก่าและใหม่ แล้วอัปเดต DOM จริงเฉพาะส่วนที่เปลี่ยน

---

4. หลักการ Immutable State และเหตุผล

- React ใช้การเปรียบเทียบแบบ shallow equality check เพื่อดูว่า state เปลี่ยนหรือไม่
- ถ้า state เป็น primitive เช่น number, string, boolean การเปลี่ยนแปลงง่าย
- แต่ถ้าเป็น object หรือ array ต้องสร้างออบเจกต์ใหม่ทุกครั้งที่เปลี่ยนแปลง
- การแก้ไข object หรือ array แบบ mutable (เช่น state.key = value) จะทำให้ React **ไม่รู้ว่า state เปลี่ยน** เพราะ reference ไม่เปลี่ยน
- จึงควรใช้ spread operator หรือวิธี clone ออบเจกต์ เช่น

```
setState(prev => ({ ...prev, newKey: newValue }));
```

---

5. การอัปเดต state แบบ asynchronous และ batch update

- React อาจ **batch** การอัปเดต state หลาย ๆ ครั้งใน event handler เดียวกัน เพื่อเพิ่มประสิทธิภาพ
- setState ใน Functional Component เป็น asynchronous (ไม่รอให้เปลี่ยนทันที)
- ถ้าใช้ค่าปัจจุบันของ state ในการอัปเดต ต้องใช้ฟังก์ชัน callback

```
setCount(prevCount => prevCount + 1);
```

- หากเขียนแบบนี้ อาจได้ค่าที่ไม่ถูกต้องเมื่อเรียก setCount(count + 1) หลาย ๆ ครั้งในรอบเดียวกัน

---

6. การใช้ค่าเริ่มต้นแบบ Lazy Initialization

- หาก initialValue ต้องคำนวณหนัก (เช่น ดึงข้อมูลจาก localStorage หรือคำนวณซับซ้อน)
- ใช้รูปแบบ lazy init คือ

```
const [state, setState] = useState(() => {
  // รันแค่ตอน mount ครั้งเดียว
  const saved = localStorage.getItem('key');
  return saved ? JSON.parse(saved) : defaultValue;
});
```

- React จะเรียกฟังก์ชันนี้เพียงครั้งเดียวตอน mount ไม่ใช่ทุก render

---

7. การใช้ useState หลายตัวใน Component เดียวกัน

- React อนุญาตให้เรียก useState หลายครั้งได้ โดยแต่ละ state แยกการจัดการกัน

```
const [count, setCount] = useState(0);
const [text, setText] = useState('');
const [items, setItems] = useState([]);
```

- ช่วยจัดการ state ให้เป็นระเบียบและแยกส่วนกัน

---

8. การจัดการ state แบบ object และ array

- ต้องสร้าง state ใหม่ทุกครั้งเมื่อแก้ไข ไม่แก้ไขแบบ mutate
- ตัวอย่างเพิ่ม item ลง array

```
setItems(prevItems => [...prevItems, newItem]);
```

- ตัวอย่างแก้ไข object

```
setUser(prevUser => ({ ...prevUser, name: 'John' }));
```

- ตัวอย่างลบ item ใน array

```
setItems(prevItems => prevItems.filter(item => item.id !== idToRemove));
```

---

9. ข้อควรระวังและ antipattern

สถานการณ์	ทำไมไม่ควรทำ	วิธีที่ถูกต้อง
แก้ไข state โดยตรง เช่น state.count = 5	React ไม่รู้ว่ามีการเปลี่ยนแปลง ทำให้ UI ไม่อัปเดต	ใช้ setState() เท่านั้น
ใช้ค่า state ปัจจุบันโดยตรงแบบ setCount(count + 1) หลายครั้งใน event เดียวกัน	อาจได้ค่าผิดพลาดเพราะ batch update	ใช้ callback: setCount(prev => prev + 1)
เรียก setState ใน render หรือใน useEffect ที่ไม่มี dependency	อาจทำให้เกิด loop ไม่สิ้นสุด	ต้องระวังเงื่อนไขการเรียกให้เหมาะสม
กำหนด initial state แบบคำนวณหนักใน useState(initialValue) โดยตรง	คำนวณทุกครั้งที่ render	ใช้ lazy init: useState(() => expensiveCalculation())

---

10. ตัวอย่างการใช้งานเต็ม ๆ แบบอธิบาย

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

```
function ShoppingCart() {
  // state เก็บรายการสินค้า (array)
  const [items, setItems] = useState([]);

  // เพิ่มสินค้าใหม่ลงในตะกร้า
  const addItem = () => {
    const newItem = { id: Date.now(), name: 'สินค้าใหม่', qty: 1 };
    setItems(prevItems => [...prevItems, newItem]);
  };

  // เพิ่มจำนวนสินค้าในตะกร้า
  const incrementQty = id => {
    setItems(prevItems =>
      prevItems.map(item =>
        item.id === id ? { ...item, qty: item.qty + 1 } : item
      )
    );
  };

  // ลบสินค้าจากตะกร้า
  const removeItem = id => {
    setItems(prevItems => prevItems.filter(item => item.id !== id));
  };

  return (
    <div>
      <h2>ตะกร้าสินค้า</h2>
      <button onClick={addItem}>เพิ่มสินค้า</button>
      <ul>
        {items.map(item => (
          <li key={item.id}>
            {item.name} - จำนวน: {item.qty}{' '}
            <button onClick={() => incrementQty(item.id)}>เพิ่ม</button>{' '}
```

```
            <button onClick={() => removeItem(item.id)}>ลบ</button>
          </li>
        ))}
      </ul>
    </div>
  );
}

export default ShoppingCart;
```

- ในตัวอย่างนี้ items เป็น state แบบ array
- การเพิ่ม, แก้ไข, ลบ ทำโดยสร้าง array ใหม่ด้วย spread operator หรือ filter
- ใช้ setItems เพื่อเปลี่ยนแปลงและรีเรนเดอร์ใหม่ทุกครั้ง

---

11. สรุปเชิงลึก

หัวข้อ	รายละเอียดเชิงลึก
useState Hook	ทำให้ Functional Component มี state ในรูปแบบคู่ [state, setState]
การอัปเดต state	asynchronous, batch update, ต้องใช้ setState เท่านั้น
การอัปเดตที่ขึ้นกับค่าเดิม	ใช้ callback เพื่อป้องกันค่าผิดพลาด
State แบบ object/array	ต้องอัปเดตแบบ immutable เพื่อให้ React รู้ว่ามีการเปลี่ยนแปลง
Lazy Initialization	ใช้ callback ใน useState เพื่อประหยัด performance
หลาย useState	แยก state หลายตัวเพื่อให้ง่ายต่อการจัดการและอ่านโค้ด
ข้อควรระวัง	หลีกเลี่ยง mutate state, setState ใน loop, ใช้ initial state ถูกวิธี
การทำงานภายใน	React ใช้ Fiber Tree เก็บ state, diffing และ reconciliation อัปเดต DOM จริง

ตัวอย่างโค้ด React แบบเต็มไฟล์ พร้อม HTML เบื้องต้น, คำอธิบาย และผลลัพธ์ที่รัน

ตัวอย่าง: การใช้ useState() จัดการ state แบบพื้นฐาน — ตัวนับจำนวน (Counter)

---

1. โครงสร้างโปรเจกต์

my-react-app/

```
├── public/
│   └── index.html
├── src/
│   └── App.jsx
│   └── main.jsx
├── package.json
└── vite.config.js  (ถ้าใช้ Vite)
```

---

2. ไฟล์ public/index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
  <title>React useState Example</title>
</head>
<body>
  <div id="root"></div>
  <!-- สคริปต์จะถูก bundle โดย Vite หรือ Create React App -->
</body>
</html>
```

---

3. ไฟล์ src/App.jsx

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

function App() {
  // สร้าง state ชื่อ count เริ่มต้นที่ 0
  const [count, setCount] = useState(0);

  // ฟังก์ชันเพิ่มค่า count
  const increment = () => {
    setCount(prevCount => prevCount + 1);
  };
```

```
  // ฟังก์ชันลดค่า count
  const decrement = () => {
    setCount(prevCount => prevCount - 1);
  };

  // ฟังก์ชันรีเซ็ต count เป็น 0
  const reset = () => {
    setCount(0);
  };

  return (
    <div style={{ padding: '20px', fontFamily: 'Arial' }}>
      <h1>Counter with useState()</h1>
      <p>ค่าปัจจุบัน: <strong>{count}</strong></p>
      <button onClick={decrement} style={{ marginRight: '10px' }}>-</button>
      <button onClick={increment} style={{ marginRight: '10px' }}>+</button>
      <button onClick={reset}>Reset</button>
    </div>
  );
}

export default App;
```

---

4. ไฟล์ src/main.jsx (สำหรับ Vite)

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App.jsx';

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

---

5. คำอธิบายโค้ด

- ใช้ useState(0) กำหนดตัวแปร state count เริ่มต้นที่ 0
- สร้างฟังก์ชัน increment และ decrement เพื่อเพิ่ม-ลดค่า count โดยใช้ฟังก์ชัน callback ของ setCount เพื่ออัปเดต state อย่างถูกต้อง
- reset จะตั้งค่า count เป็น 0
- ปุ่มแต่ละปุ่มมี onClick event handler ผูกกับฟังก์ชันที่กำหนดไว้
- เมื่อ setCount ถูกเรียก React จะรีเรนเดอร์ UI ใหม่และแสดงค่าปัจจุบันของ count ใน {count}

---

6. ผลการรัน

- หน้าเว็บแสดงหัวข้อ **Counter with useState()**
- มีข้อความแสดงค่าปัจจุบัน เช่น "ค่าปัจจุบัน: 0"
- ปุ่ม "-" ลดค่า count ลง 1
- ปุ่ม "+" เพิ่มค่า count ขึ้น 1
- ปุ่ม "Reset" ตั้งค่า count กลับเป็น 0
- ทุกครั้งที่กดปุ่ม ค่า count จะเปลี่ยนและแสดงผลแบบ real-time โดยไม่รีเฟรชหน้าใหม่

นี่คือตัวอย่าง React useState() แบบเต็มไฟล์ จำนวน 5 ตัวอย่าง พร้อมคำอธิบายและผลการรัน

ตัวอย่างที่ 1: การจัดการ State แบบ Object (ฟอร์มกรอกข้อมูล)

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

function App() {
  // state เป็น object เก็บชื่อและอีเมล
  const [formData, setFormData] = useState({ name: '', email: '' });

  const handleChange = e => {
    const { name, value } = e.target;
    setFormData(prev => ({ ...prev, [name]: value }));
  };

  return (
    <div style={{ padding: '20px' }}>
      <h2>Simple Form</h2>
```