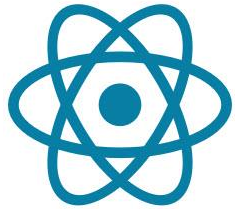

NEXT.JS

หลักสูตร "100+ เทคนิค React/Next.js Design Patterns และ Best Practices"

โค้ชเอก

codingthailand.com

โปรแกรมที่ต้องติดตั้งมาก่อน

1. Git

https://git-scm.com/

2. Node.js เวอร์ชัน LTS

https://nodejs.org/en

3. Visual Studio Code

https://code.visualstudio.com/

ตรวจสอบโปรแกรมที่ติดตั้ง

```
node –v

npm –v


#Upgrade to the latest version of npm

npm install -g npm@latest


git --version
```

การตั้งค่า Git (ทำครั้งเดียวต่อเครื่อง)

- ตรวจสอบการ config user และ default branch ด้วยคำสั่ง

  git config -l

- **git config --global user.name "Your name"**
- **git config --global user.email "Your email"**

- ตั้งค่า default branch ชื่อว่า main ด้วยคำสั่ง ดังนี้ (optional)

  git config --global init.defaultBranch main

สร้างโปรเจคใหม่ React

- https://vite.dev/guide/#scaffolding-your-first-vite-project

สร้างโปรเจคใหม่ Next.js

- https://nextjs.org/docs/app/getting-started/installation

นิยาม

- แนวปฏิบัติที่ดีที่สุด (Best Practices) คือ วิธีการหรือเทคนิคที่ได้รับการยอมรับโดยทั่วไปว่ามีความเหนือกว่าทางเลือกอื่น เนื่องจากมักให้ผลลัพธ์ที่ดีกว่า

- รูปแบบการออกแบบ (Design Patterns) คือ แนวทางแก้ไขปัญหาทั่วไปในงานออกแบบซอฟต์แวร์ โดยแต่ละรูปแบบเปรียบเสมือนพิมพ์เขียวที่สามารถปรับแต่งและใช้ซ้ำได้ เพื่อแก้ปัญหาเฉพาะการออกแบบโค้ดของเรา

#1 เลือกแนวทางการเขียน React ให้ถูกต้อง

- ทีมของ React แนะนำว่า **ถ้าจะขึ้น Project ใหม่ ตอนนี้แนะนำให้ใช้ framework ไปเลย** เช่น Next.js, React Route v7 (framework mode) หรืออื่นๆ ไม่แนะนำให้ขึ้น Project จากศูนย์ด้วยตัวเอง เพราะตอนนี้ framework ต่างๆ สามารถทำได้ทั้ง**แบบ client-side rendering (CSR) / single-page apps (SPA) หรือ static-site generation (SSG) โดยไม่ต้องมี server ก็ได้**
- การใช้ framework นั้นนอกจะช่วยลดระยะเวลาการพัฒนาแล้วยังมีฟีเจอร์มากมาย เช่น routing, data-fetching ซึ่งช่วยให้ app ของเรามีประสิทธิภาพดีขึ้นโดยไม่ต้องเขียนเอง
- ในอนาคตเมื่อ Project อยากใช้ full-stack ก็สามารถขยายและเปิดฟีเจอร์ได้เลยโดยไม่ต้องเขียนใหม่ทั้งหมด

- แต่หากคิดว่าทีมหรือตัวเราสามารถจัดการปัญหา หรือเพิ่มฟีเจอร์ต่างๆ ได้เอง ถ้าอยากเริ่มจากศูนย์ก็สามารถทำได้

#2 ใช้ TypeScript แทน JavaScript

- TypeScript จะช่วยป้องกันข้อผิดพลาดจาก Type ที่ไม่ถูกต้อง
- ช่วยลด bug จากการใช้ type ผิด
- ป้องกันการส่งข้อมูลผิดไปยัง Props ของ component
- ช่วยให้อ่านโค้ดง่าย และมี auto-complete

#3 การตั้งชื่อ Component ที่ดี

- ใช้ PascalCase สำหรับการตั้งชื่อ component เช่น Button.tsx, UserCard.tsx, Dashboard.tsx เป็นต้น

#4 การตั้งชื่อ Hook และ Utility Functions ที่ดี

- ใช้ camelCase สำหรับการตั้งชื่อ Hook เช่น useAuth, useFetch เป็นต้น
- ใช้ camelCase สำหรับการตั้งชื่อ Utility Functions เช่น formatDate.ts, debounce.ts เป็นต้น

#5 การตั้งชื่อ State Variables และ Constant Variables ที่ดี

- สำหรับ State Variable **ค่าที่เป็น boolean** ให้ขึ้นต้นด้วย is, has, หรือ should เป็นต้น

- สำหรับ Constant Variable ให้**ใช้ UPPER_SNAKE_CASE** เช่น API_URL, MAX_RESULT เป็นต้น

#6 การตั้งชื่อ Event Handlers ใน React และชื่อฟังก์ชันใน JavaScript

- การตั้งชื่อ Event ของ React การใช้ camelCase และตั้งชื่อ**ขึ้นต้นด้วยคำว่า handle** เช่น handleButtonClick()

- การชื่อฟังก์ชัน (functions) ใน JavaScript/TypeScript ต้องคำนึงถึงตัวพิมพ์เล็ก-ใหญ่ ควรเริ่มต้นด้วยตัวอักษรพิมพ์เล็ก ใช้ camelCase ในการตั้งชื่อ และควรตั้งชื่อที่มีความหมาย และควรใช้คำกริยา เช่น getUserData(), fetchOrders เป็นต้น

#7 การตั้งชื่อ Higher Order Components

- Higher Order Component (HOC) คือ ฟังก์ชันที่รับ Component เป็น input และ return Component ใหม่เป็น output ใช้เพื่อ reuse logic, เพิ่มฟีเจอร์ต่างๆ ให้กับ component ใหม่ โดยไม่ต้องแก้ไข component เดิม ปัจจุบันในหลายกรณีอาจไม่จำเป็นแล้ว เปลี่ยนมาใช้ Hooks แทน

- ให้ใช้ camelCase และ**ขึ้นด้วย with** เช่น withAuth, withTheme เป็นต้น

#8 ใช้ Functional Components แทน Class Components

- ตั้งแต่ React 16.8+ แนะนำให้ใช้ Functional Components และ Hooks
- โค้ดสั้น อ่านง่ายกว่า เป็นฟังก์ชัน ไม่ต้องใช้ this และ ใช้ Hooks ได้
- ทำงานได้เร็วกว่า ไม่ต้องสร้าง instance ของ class
- รองรับการ reuse โค้ดได้ดี เช่น การเขียน Custom Hook เป็นต้น

#9 จัดรูปแบบ JSX ให้อ่านง่าย

- เว้นบรรทัดแต่ละส่วนให้ชัดเจน
- จัด props ให้อยู่แนวเดียวกัน แต่ถ้ามากกว่า 2-3 ตัวให้ขึ้นบรรทัดใหม่

- แนะนำให้ใช้ไลบรารีเพื่อจัดรูปแบบอัตโนมัติ เช่น Prettier เป็นต้น

#10 ใช้ Props เพื่อทำให้ Component ใช้ซ้ำได้

- ระวังอย่า hardcoded ค่าใน component
- ควรใช้ props เพื่อให้ component กลับมาใช้ใหม่ได้

#11 ใช้ TypeScript เพื่อตรวจสอบ Props

- อย่าพยายามใช้ any สำหรับการกำหนด type ใน props
- ควรสร้าง type สำหรับ props ขึ้นมาเพื่อป้องกันการส่งค่าที่ผิดเข้ามาใน component

- ไม่ควรใช้ PropTypes ในการขึ้นโปรเจคใหม่ ให้ใช้ TypeScript แทน

#12 ใช้ Default Props ลดความเสี่ยงของ undefined props

- ควรตั้งค่าเริ่มต้น (default) ให้ props เพื่อลดความเสี่ยงจาก undefined
- ใน TypeScript สามารถใช้ optional ได้

#13 หลีกเลี่ยงการส่ง Props ที่ไม่จำเป็น

- อย่าส่งข้อมูลที่ไม่จำเป็นเข้ามาที่ props ของ component
- ควรส่งเฉพาะที่ใช้จริงๆ เท่านั้น เพื่อลดการ re-render และทำให้อ่านง่ายขึ้น

#14 ใช้ Destructuring ในการรับ Props

- ใช้ฟีเจอร์ Destructuring เพื่อช่วยให้โค้ดอ่านง่ายขึ้น และลดการใช้ prop. ซ้ำๆ

#15 ใช้ React DOM Attributes Type ของ React

- หากต้องการเขียน component ที่ใช้ props ตามมาตรฐานของ HTML ควรใช้ Type ของ React ที่มีมาให้ เรียกว่า React DOM Attributes Type เช่น button, input, img เป็นต้น

Type	ใช้กับ Element	อธิบาย
`React.ButtonHTMLAttributes<HTMLButtonElement>`	`<button>`	รองรับ props ของ `<button>` ทั้งหมด
`React.InputHTMLAttributes<HTMLInputElement>`	`<input>`	รองรับ props เช่น `type`, `value`, `onChange`
`React.AnchorHTMLAttributes<HTMLAnchorElement>`	`<a>`	รองรับ props เช่น `href`, `target`
`React.FormHTMLAttributes<HTMLFormElement>`	`<form>`	รองรับ props เช่น `onSubmit`, `method`
`React.ImgHTMLAttributes<HTMLImageElement>`	`<img>`	รองรับ props เช่น `src`, `alt`

#16 ใช้ Rest Props เพื่อให้ Component ยืดหยุ่น

- ใช้ Rest Props เพื่อช่วยให้ Component รองรับ props อื่นๆ ได้ โดยไม่ต้องกำหนดทุกค่าแบบตายตัว
- Component ที่สร้างจะยืดหยุ่นขึ้น เพราะสามารถส่ง props มาได้ทั้งหมด
  (ของ React DOM Attributes Type)

#17 แยก Component ให้เป็น Single Responsibility Principle (SRP)

- Component ควรทำงานแค่ **หนึ่งอย่าง เท่านั้น**
- ถ้ามีหลายหน้าที่ แยกออกเป็น Components ย่อย

#18 แยก Logic ออกจาก UI ด้วย Custom Hooks

- แยก Business Logic ออกจาก UI โดยใช้ Custom Hooks เพื่อให้สามารถใช้ซ้ำได้ในหลายๆที่

#19 ระวังการใช้ Inline Styles

- ระวังการใช้ inline styles เพราะจะทำให้โค้ดอ่านยาก ประสิทธิภาพต่ำ
- ไม่สามารถใช้ฟีเจอร์ของ CSS บางตัวได้

#20 ใช้ Arrow Functions ใน React

- Syntax สั้นกว่า ฟังก์ชันปกติ
- อ่านง่ายขึ้น โดยเฉพาะกับ callback หรือ event handler
- ช่วยให้โค้ดสะอาดและดูแลรักษาง่าย

THE END