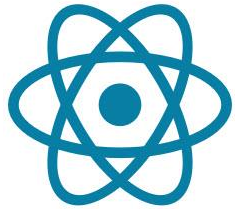

NEXT.JS

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

โค้ชเอก

codingthailand.com

นิยาม

- แนวปฏิบัติที่ดีที่สุด (Best Practices) คือ วิธีการหรือเทคนิคที่ได้รับการยอมรับโดยทั่วไปว่ามีความเหนือกว่าทางเลือกอื่น เนื่องจากมักให้ผลลัพธ์ที่ดีกว่า

- รูปแบบการออกแบบ (Design Patterns) คือ แนวทางแก้ไขปัญหาทั่วไปในงานออกแบบซอฟต์แวร์ โดยแต่ละรูปแบบเปรียบเสมือนพิมพ์เขียวที่สามารถปรับแต่งและใช้ซ้ำได้ เพื่อแก้ปัญหาเฉพาะการออกแบบโค้ดของเรา

#21 จัดโครงสร้าง React Router ให้อ่านและบำรุงรักษาง่าย

- ใช้ฟีเจอร์ของ React Router เพื่อแยกไฟล์ route ให้เป็นสัดส่วน หรือตามฟีเจอร์
- ช่วยให้อ่านโค้ดง่าย และบำรุงรักษาง่าย

#22 ใช้ฟีเจอร์ Lazy Route Modules ใน React Router

- ใช้ฟีเจอร์นี้เพื่อลดขนาดของไฟล์ bundle ทำให้ app เราเล็กลง
- รองรับการทำ code-splitting สำหรับ route

#23 สร้างเพจ Global Error Page ใน React Router

- ใช้ฟีเจอร์ของ react router สร้างเพจเพื่อ handle errors ทั้งหมดใน app

#24 ใช้ lazy() และ <Suspense> ใน React เพื่อเพิ่มประสิทธิภาพ

- ใน React ฟังก์ชัน lazy() และคอมโพเนนต์ <Suspense> ถูกใช้**สำหรับการแยกโค้ด (code splitting) และการจัดการสถานะการโหลด** ซึ่งมีประโยชน์อย่างยิ่งในแอปพลิเคชันขนาดใหญ่ ที่ต้องการเพิ่มประสิทธิภาพในการโหลดครั้งแรกและปรับปรุงประสบการณ์ของผู้ใช้โดยการแสดง loading indicators
- **ฟังก์ชัน lazy() ช่วยให้คุณสามารถนำเข้าคอมโพเนนต์แบบไดนามิก** ซึ่งหมายความว่าคอมโพเนนต์จะถูกโหลดก็ต่อเมื่อมีการใช้งานจริง แทนที่จะโหลดทั้งหมดตั้งแต่เริ่มต้นการเรนเดอร์ วิธีนี้เหมาะสำหรับคอมโพเนนต์ขนาดใหญ่หรือคอมโพเนนต์ที่ยังไม่จำเป็นต้องใช้ทันที เช่น
- **คอมโพเนนต์ <Suspense>** ช่วยให้คุณสามารถกำหนดการโหลดในระหว่างที่กำลังโหลดคอมโพเนนต์ที่ใช้ lazy() (ใช้คู่กัน) ช่วยให้ UX ดีขึ้น คือแสดงสถานะที่โหลดนั่นเอง
- **อย่าใช้ lazy() กับ คอมโพเนนต์เล็ก ๆ หรือที่ใช้งานบ่อย**

#25 ใส่ค่า key ให้ถูกต้องเพื่อช่วยให้ React ทำงานได้มีประสิทธิภาพ

- **ใช้ id ที่ไม่เปลี่ยนแปลง เป็น key** เช่น primary key จากฐานข้อมูล เป็นต้น (แนะนำ)
- หลีกเลี่ยง index ของ array เป็น key เว้นแต่ array จะไม่เปลี่ยนลำดับ
- อย่าใช้ Math.random() เพราะ key เปลี่ยนทุกครั้งเท่ากับว่า React รีเรนเดอร์ใหม่ทั้งหมด

#26 จัดลำดับการนำเข้า (Import) ให้เป็นระเบียบ

- หลีกเลี่ยงการ import ที่ปะปนกันระหว่าง external imports (third-party packages) และ internal imports (เช่น components, utils functions, styles)
- ควรจัดกลุ่มให้ชัดเจน อ่านง่าย ลดข้อผิดพลาด ไม่ต้องไล่หาสลับไปมา
  - Built-in Imports ⟶ เช่น react, redux, react-hook-form
  - Third-Party Libraries ⟶ เช่น @mui/material, axios
  - Internal Imports ⟶ เช่น คอมโพเนนต์, ค่าคงที่ (constants), ฟังก์ชัน (utils)
- ถ้าไม่อยากจัดเอง แนะนำลองใช้ ESLint + Prettier ทำให้อัตโนมัติแทน ลองค้น การใช้ eslint-plugin-import

#27 หลีกเลี่ยงการใช้ useState ที่เยอะเกินไป ให้ใช้ useReducer แทน

- การใช้ useState เยอะๆ ทำให้โค้ดรกและอ่านยาก และต้องเรียก setState หลายครั้ง

- แนะนำให้มาใช้ useReducer แทน โค้ดอ่านง่ายขึ้น จัดการ state ซับซ้อนได้ดี ง่ายต่อการ debug
- https://react.dev/reference/react/useReducer

#28 ใช้ shorthand สำหรับ props ที่เป็น boolean

- ใช้ props ที่มีค่าเป็น true และเขียนแบบสั้น เพื่อโค้ดสะอาดอ่านง่าย เช่น

<Form hasPadding isDisabled />

#29 หลีกเลี่ยงการใช้ curly braces สำหรับ string props

- JSX อนุญาตให้ส่ง string ตรงๆ อยู่แล้ว ไม่ต้องใช้ปีกกา
- ทำให้โค้ดอ่านง่ายขึ้น ดูสะอาด และลดการใช้ปีกกาโดยไม่จำเป็น

#30 ใช้ Fragment เมื่อ div ไม่จำเป็น

- JSX ต้องมี root element เพียงตัวเดียว
- ให้ใช้ <></> แทน และหลีกเลี่ยง ถ้าไม่ได้ใช้
- ไม่เพิ่ม node ของ html โดยไม่จำเป็น ทำให้รกและอ่านยาก
- รักษาโครงสร้างของ HTML ไม่ให้ซับซ้อน (หลีกเลี่ยง div hell)
- อาจทำให้ CSS Layout ซับซ้อนขึ้น

#31 ใช้ Self-Closing Tags เมื่อไม่มี Children

- ระวังการใช้ tag เปิด-ปิด โดยไม่จำเป็น
- ทำให้โค้ดอ่านง่ายขึ้น และลดความยุ่งเหยิง

#32 เรียนรู้การใช้งาน props.children ใน React

- ใช้สร้าง คอมโพเนนต์ที่ยืดหยุ่นและนำกลับมาใช้ใหม่ได้ง่าย

THE END