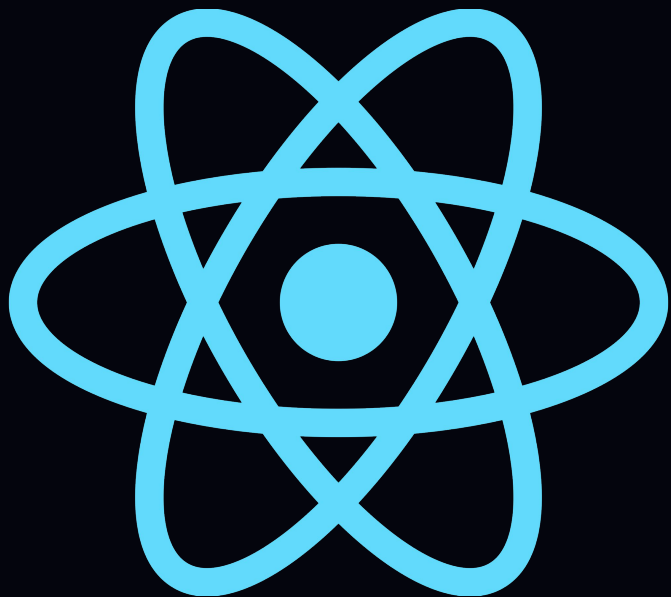


พัฒนาเว็บแอปพลิเคชันด้วย React
(อัปเดตล่าสุด)

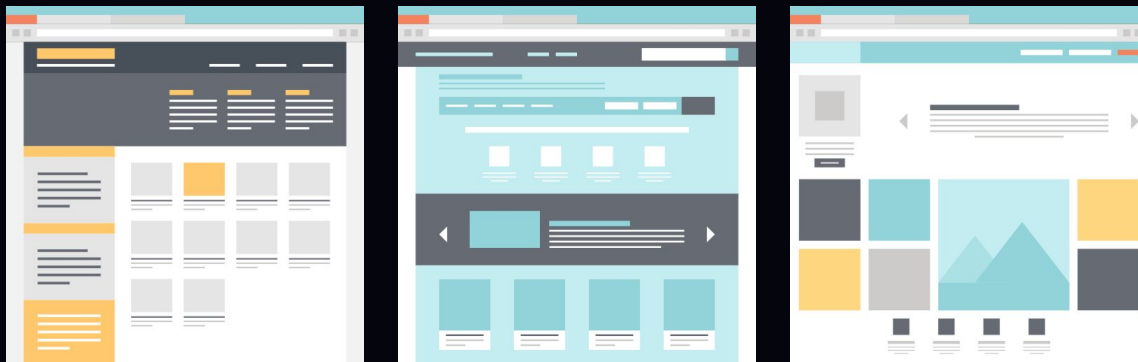
รู้จักกับ React

React คืออะไร



คือ ไลบรารีของภาษา JavaScript
ที่ใช้สำหรับสร้างส่วนติดต่อกับผู้ใช้งาน
(User Interface : UI) หรือการสร้าง
หน้าเว็บให้สวยงามและใช้งานง่าย

แนวคิดของ React

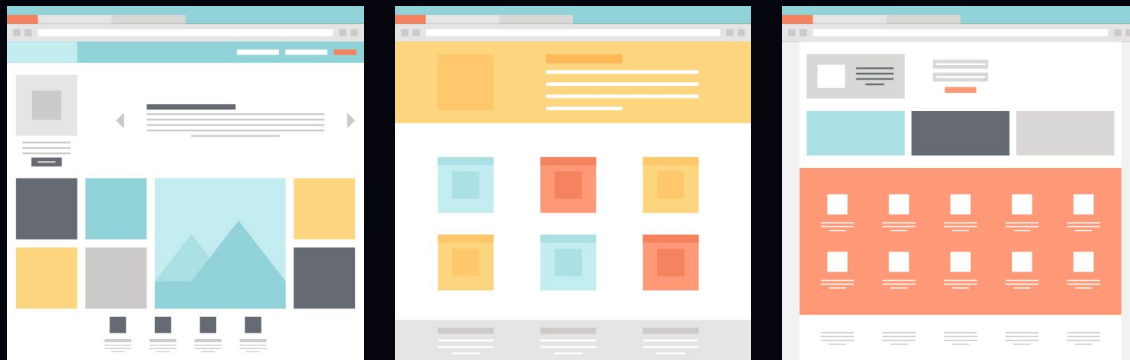


การสร้างเว็บไซต์แบบดั้งเดิม ผู้พัฒนาเว็บไซต์ต้องเขียนโค้ดหน้าเว็บทั้งหมดเก็บไว้ในไฟล์เดียว ส่งผลให้เว็บไซต์ทำงานช้าเนื่องจากต้องโหลดเนื้อหาใหม่ทั้งหมดเมื่อมีการเปลี่ยนแปลงการทำงานภายในหน้าเว็บ

แนวคิดของ React

React ถูกพัฒนาขึ้นมาเพื่อนำมาใช้สร้างหน้าเว็บ โดยมีแนวคิดคือการแบ่งส่วนแสดงผลออกเป็น **ชิ้นส่วนย่อยหลายๆส่วน** โดยไม่ต้องเขียนโค้ดเก็บไว้ในไฟล์เดียว เพื่อให้ง่ายต่อการจัดการ จากนั้นจึงค่อยนำส่วนย่อยดังกล่าวมาประกอบรวมกันในภายหลัง เราจะเรียกองค์ประกอบที่แบ่งออกเป็นชิ้นส่วนย่อย ๆ นี้ว่า **“คอมโพเนนต์ (Component)”**

แนวคิดของ React



ข้อดีของคอมโพเนนต์ คือ สามารถที่จะออกแบบส่วนประกอบต่างๆ แล้วนำกลับมาใช้งานในภายหลังได้ โดยที่ไม่ต้องเสียเวลาเขียนใหม่

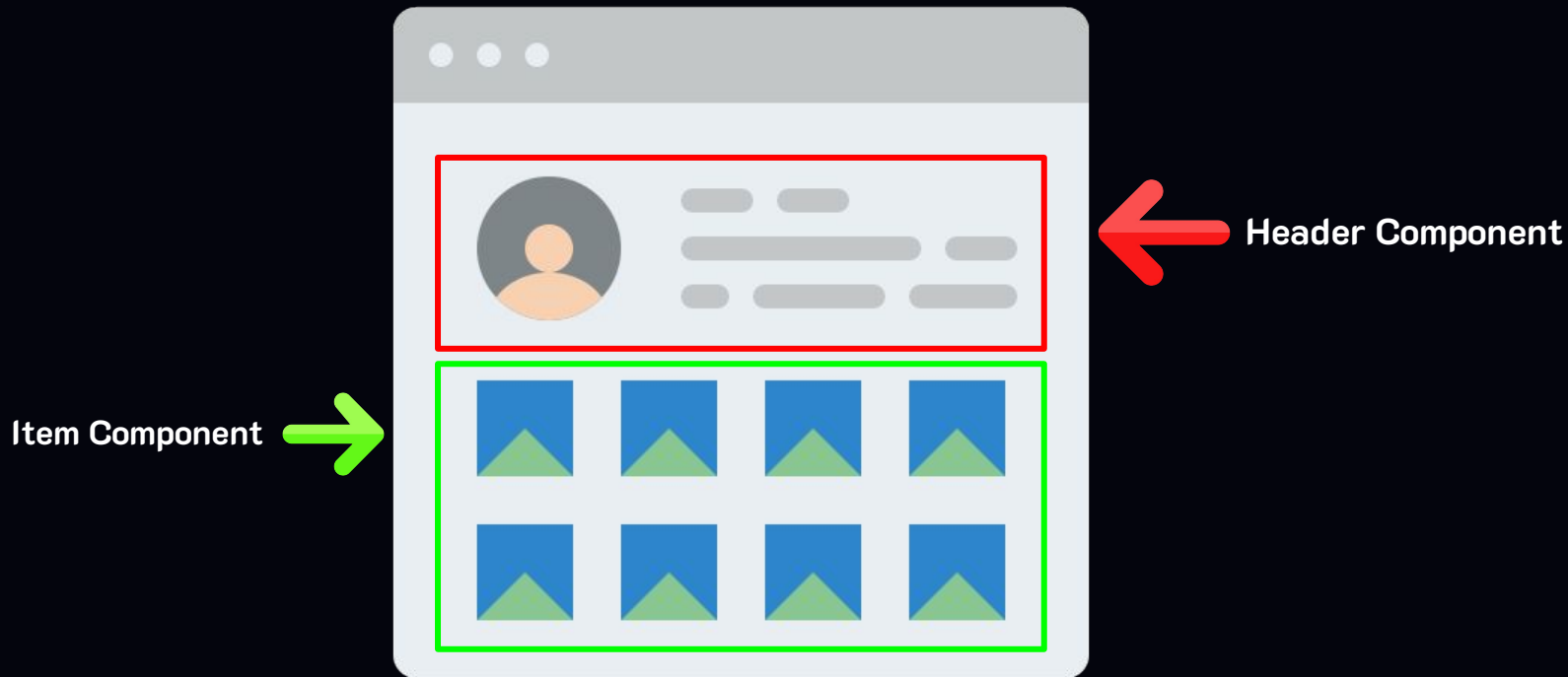
แนวคิดของ React



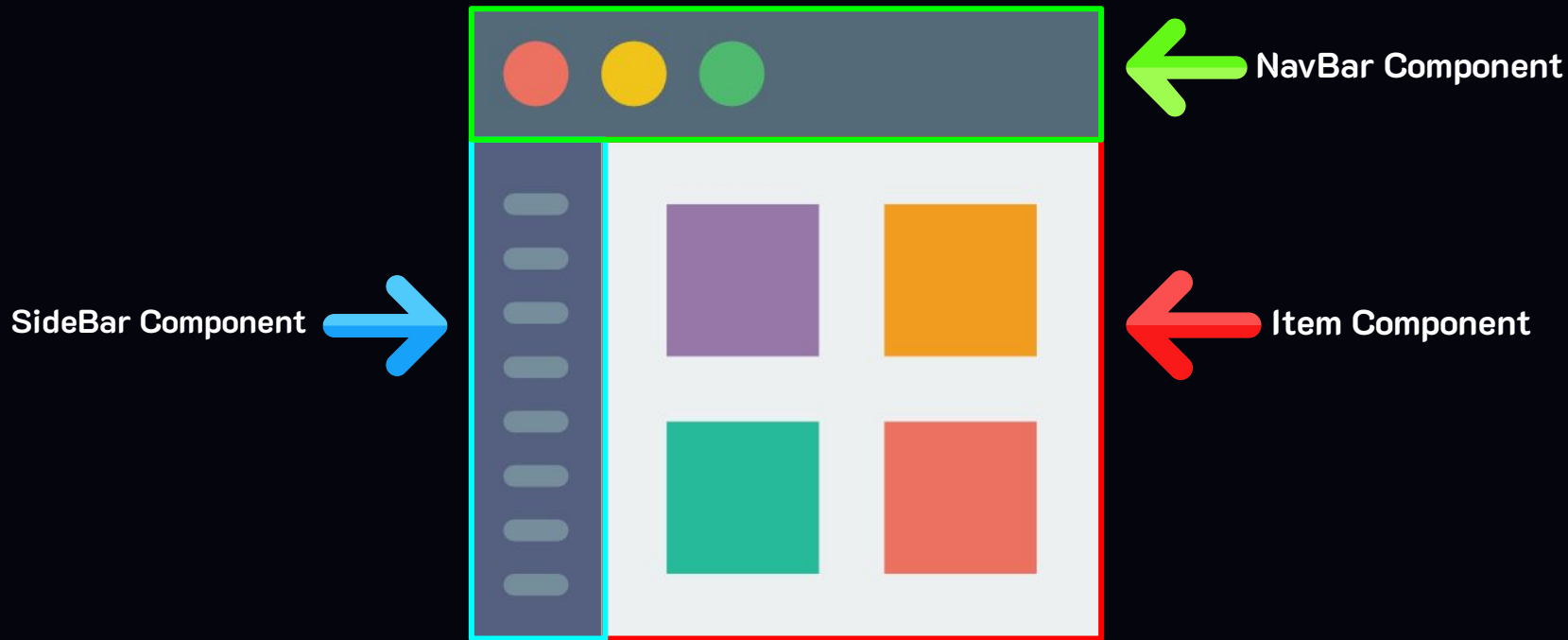
แนวคิดของ React



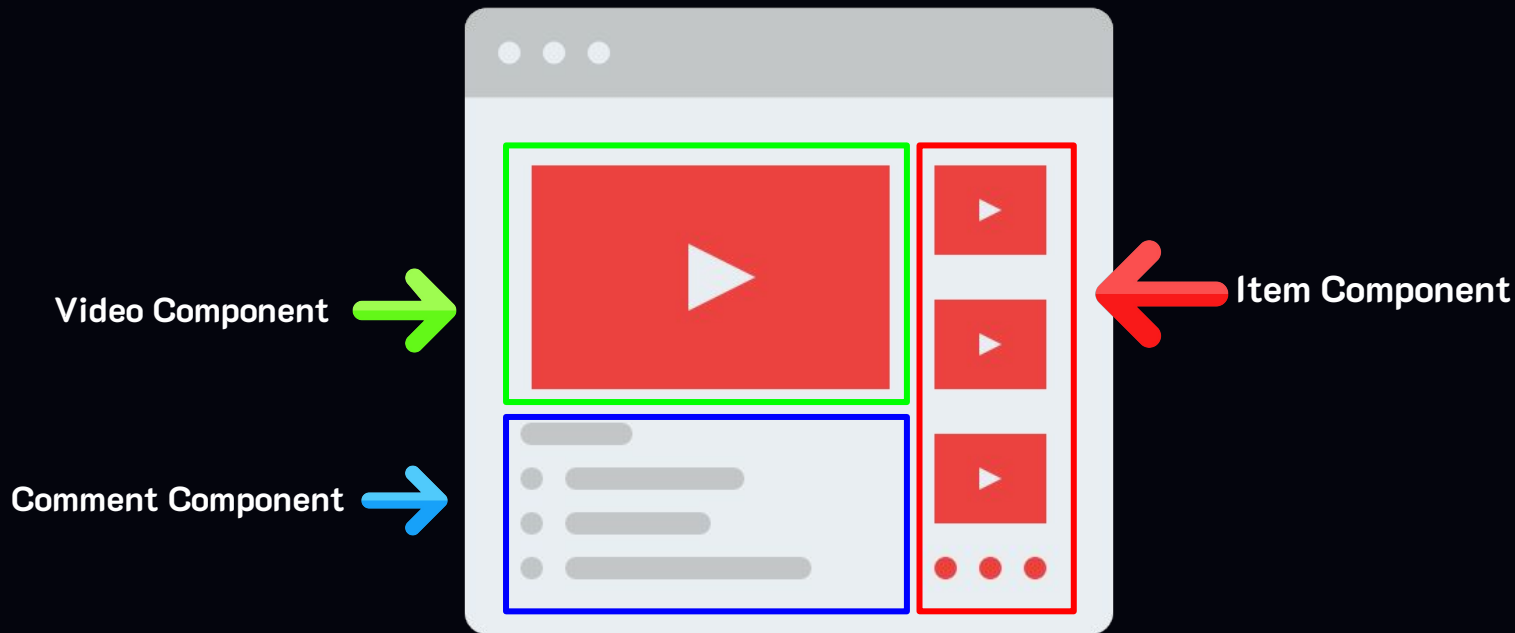
แนวคิดของ React



แนวคิดของ React



แนวคิดของ React



แนวคิดของ React

- **Component** คือ ชิ้นส่วนต่างๆที่ถูกนำมาประกอบรวมกันเป็นหน้าเว็บ (คล้ายๆ สร้าง tag ขึ้นมาใช้เอง เช่น `<Navbar/>`) สามารถนำกลับมาใช้ซ้ำได้
- **State** คือ ข้อมูลที่ถูกเก็บไว้ใน **Component** สามารถเปลี่ยนแปลงได้ตามการกระทำของผู้ใช้งาน เช่น สถานะการล็อกอิน ข้อมูลที่กรอกในฟอร์ม จำนวนครั้งที่กดปุ่ม เป็นต้น เมื่อข้อมูล State เปลี่ยน React จะอัปเดตหน้าเว็บให้อัตโนมัติ

แนวคิดของ React

- **Props (Properties)** คือ ข้อมูลที่ถูกส่งจาก Component หนึ่งไปยังอีก Component หนึ่ง ทำให้แต่ละ Component สามารถแลกเปลี่ยนข้อมูลหรือรับค่ามาแสดงผลหรือใช้งานได้หลากหลายตามที่เราต้องการได้

ต้องมีพื้นฐานอะไรบ้าง

ต้องมีพื้นฐานอะไรบ้าง



- มีพื้นฐาน HTML5
- มีพื้นฐาน CSS3
- มีพื้นฐาน JavaScript
- มีพื้นฐานการใช้งาน Visual Studio Code

เครื่องมือพื้นฐาน

- Node.js
- Visual Studio Code
- Google Chrome
- React Developer Tools (Extension)



VSCode Extension

- Auto Rename Tag
- Color Highlight
- Prettier – Code formatter
- Material Icon Theme *(Optional)*



สร้างโปรเจกต์ React

รู้จักกับ Vite



Vite คือ เครื่องมือสำหรับการพัฒนาเว็บไซต์สมัยใหม่มี
จุดเด่นในเรื่องการ Run & Build แอปพลิเคชันได้อย่างรวดเร็ว
 อีกทั้งยังสามารถปรับแต่งได้ง่าย เหมาะสำหรับนำมาใช้ในการ
สร้างโปรเจกต์ React , Vue และอื่นๆ

จุดเด่นของ Vite

- ติดตั้งและใช้งานง่าย ใช้คำสั่งไม่กี่คำสั่งก็สามารถสร้างโปรเจกต์ได้ทันทีและมีโครงสร้างโปรเจกต์ที่เข้าใจง่าย
 - รองรับหลาย Framework เช่น React, Vue, Svelte อื่นๆ (รองรับ JavaScript และ TypeScript)
- ** ทำงานได้เร็วกว่า Create-React-App (แบบดั้งเดิม)**

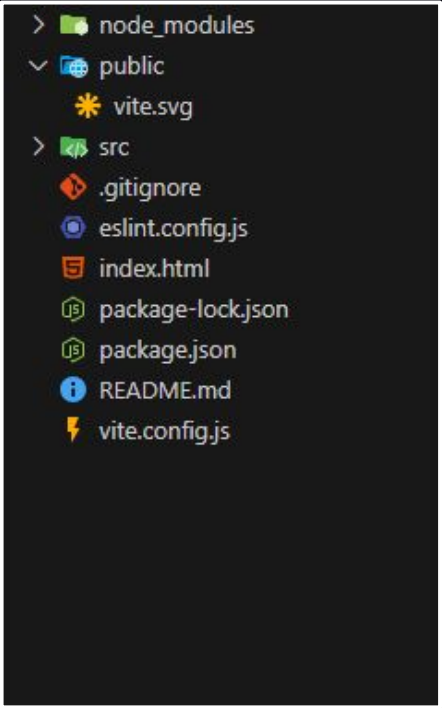
จุดเด่นของ Vite

- **พัฒนาโปรเจกต์ได้อย่างมีประสิทธิภาพ** สามารถอัปเดตโค้ดและดูผลลัพธ์ได้ทันที โดยจะทำการโหลดเฉพาะโค้ดที่จำเป็นหรือโค้ดที่มีการเปลี่ยนแปลงการทำงานเท่านั้นในระหว่างการพัฒนาโปรเจกต์
- **Build & Deploy ได้อย่างรวดเร็ว** สามารถจัดการ Assets ต่างๆได้ง่าย

สร้างโปรเจกต์ React

- `npm create vite@latest <ชื่อโปรเจกต์>`
- `cd <ชื่อโปรเจกต์>`
- `npm install`
- `npm run dev`

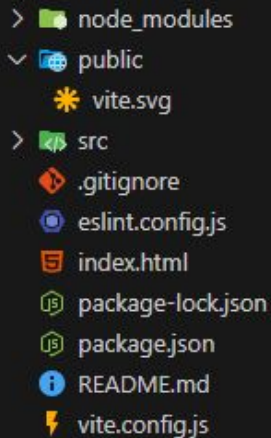
โครงสร้างโปรเจกต์



```
> node_modules
✓ public
  vite.svg
> src
  .gitignore
  eslint.config.js
  index.html
  package-lock.json
  package.json
  README.md
  vite.config.js
```

- **node_modules** คือ โฟลเดอร์สำหรับจัดเก็บโมดูลหรือไลบรารีที่นำมาทำงานภายในโปรเจกต์
- **package.json** คือ ไฟล์ที่เก็บข้อมูลพื้นฐานต่างๆ รวมถึง package หรือไลบรารีที่จะนำมาใช้ทำงานภายในโปรเจกต์

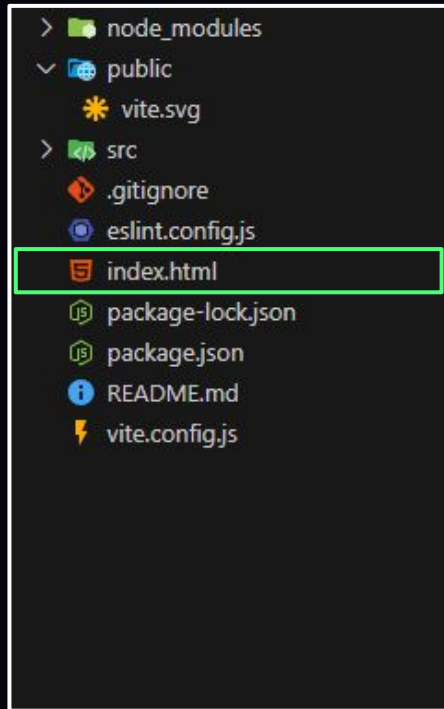
โครงสร้างโปรเจกต์



```
> node_modules
✓ public
  vite.svg
> src
  .gitignore
  eslint.config.js
  index.html
  package-lock.json
  package.json
  README.md
  vite.config.js
```

- **public** คือ โฟลเดอร์ที่จัดเก็บไฟล์ต่างๆที่จะนำใช้งานในโปรเจกต์ เช่น รูปภาพ เสียง วิดีโอ
- **vite.config.js** คือ ไฟล์สำหรับตั้งค่าการทำงานเพิ่มเติมของ Vite เช่น Plugin ต่างๆ เป็นต้น

โครงสร้างโปรเจกต์



- ไฟล์ `index.html` เป็นไฟล์ HTML สำหรับแสดงผลลัพธ์ใน Browser ซึ่งเนื้อหาที่นำมาแสดงผลนั้นมาจากคอมโพเนนต์ (Components)

สิ่งที่เราสนใจใน index.html ก็คือ คำสั่งที่อยู่ใน `<body>` ตรงส่วนของพื้นที่ `<div id="root"></div>` โดยนำไฟล์ `main.jsx` เข้ามาใช้งาน และดึงเนื้อหาที่อยู่ใน `App.jsx` มาแสดงผลในพื้นที่ดังกล่าว

```
<body>
  <div id="root"></div>
  <script type="module" src="/src/main.jsx"></script>
</body>
```

index.html



```
src > main.jsx
1  import { StrictMode } from 'react'
2  import { createRoot } from 'react-dom/client'
3  import './index.css'
4  import App from './App.jsx'
5
6  createRoot(document.getElementById('root')).render(
7    <StrictMode>
8      <App />
9    </StrictMode>,
10  )
```

main.jsx ทำการเชื่อม App Component
(App.jsx) เข้ากับ index.html

สิ่งที่เราสนใจใน index.html ก็คือ คำสั่งที่อยู่ใน <body> ตรงส่วนของพื้นที่
<div id="root"></div> โดยนำไฟล์ main.jsx เข้ามาใช้งาน และดึงเนื้อหาที่อยู่ใน

App.jsx มาแสดงผลในพื้นที่ดังกล่าว

```
<body>
  <div id="root"></div>
  <script type="module" src="/src/main.jsx"></script>
</body>
```

index.html



src > main.jsx

```
1 import { StrictMode } from 'react'
2 import { createRoot } from 'react-dom/client'
3 import './index.css'
4 import App from './App.jsx'
5
6 createRoot(document.getElementById('root')).render(
7   <StrictMode>
8     <App />
9   </StrictMode>,
10 )
```

main.jsx ทำการเชื่อม App Component
(App.jsx) เข้ากับ index.html

โครงสร้างโปรเจกต์

src คือ โฟลเดอร์ที่เก็บไฟล์คอมโพเนนต์หรือโครงสร้างหลักของแอปพลิเคชัน ซึ่งประกอบด้วยไฟล์และโฟลเดอร์สำคัญ ดังนี้

- **App.css** เป็นไฟล์ที่เก็บโค้ด CSS สำหรับนำไปใช้งานในคอมโพเนนต์เริ่มต้น
- **App.jsx** เป็นไฟล์คอมโพเนนต์เริ่มต้นของแอปพลิเคชัน
- **main.jsx** คือไฟล์หน้าแรกของแอปพลิเคชัน (เชื่อมโยงกับไฟล์ *index.html*)
- **index.css** คือไฟล์ CSS ที่ใช้งานในหน้าแรก (ใช้คู่กับไฟล์ *main.jsx*)
- **assets** คือ โฟลเดอร์ที่จัดเก็บไฟล์ต่างๆที่จะนำมาใช้งานในคอมโพเนนต์

React JSX

แนวคิดของ React

การสร้างหน้าเว็บขึ้นมาได้นั้น React จะใช้สิ่งที่เรียกว่า **คอมโพเนนต์ (Component)** หรือ ชิ้นส่วนย่อยแต่ละส่วนของหน้าเว็บนำมาประกอบรวมกัน ซึ่งจะเขียนด้วยภาษา JavaScript เพื่อออกแบบและทำหน้าตาแต่ละส่วนของเว็บไซต์

แนวคิดของ React

ใน React จะไม่เขียน HTML ในไฟล์ HTML โดยตรง แต่จะเขียนใน JavaScript แทน ซึ่งจะอาศัยสิ่งที่เรียกว่า **JSX (JavaScript XML)** ที่ทำให้สามารถใส่ HTML เข้าไปใน JavaScript ได้

ดังนั้นการใช้งาน *React* ก็คือ การสร้างหน้าเว็บด้วยภาษา *JavaScript* ที่มี *HTML* แทรกอยู่นั่นเอง !

React JSX คืออะไร

JSX (JavaScript XML) คือ การเขียนแท็ก HTML ภายในโค้ดของ JavaScript สามารถเขียนใน `<div>` section / article / Fragments `<>` ก็ได้และต้องมีการกำหนด Tag เปิด - ปิด ทุกครั้ง

ตัวอย่าง JSX

HTML Tags	JSX
<code>
</code>	<code>
</code>
<code><hr></code>	<code><hr /></code>
<code><img...></code>	<code><img... /></code>
<code><input...></code>	<code><input... /></code>

ตัวอย่าง JSX

HTML Attribute	JSX
class	className
maxlength	maxLength
onclick	onClick
onmousedown	onMouseDown

*การใส่ Class Style ที่เป็น Attribute ใน JSX จะใช้คำว่า `className` แทน คำว่า `class` เนื่องจากคำว่า `class` เป็น keyword ในภาษา JavaScript

การใช้งาน JSX

```
function App(){  
  return (  
    //พื้นที่เขียน HTML  
  );  
}
```



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

การใช้งาน JSX & JavaScript

```
function App() {
```

```
  const name="ก้องรักสยาม"
```

```
  return (
```

```
    <h1>สวัสดีครับผมชื่อ : {name}</h1>
```

```
  );
```

```
}
```

ใช้ปีกกาสำหรับแทรกคำสั่ง
JavaScript ลงไปใน HTML

การใช้งาน JSX & JavaScript

```
function App() {  
  const name="ก้องรักสยาม"  
  return (  
    <h1>สวัสดีครับผมชื่อ : {name}</h1>  
    <p>ที่อยู่ : กรุงเทพมหานคร</p>  
  );  
}
```

การใช้งาน JSX & JavaScript

```
function App() {  
  const name="ก้องรักสยาม"  
  return (  
    <h1>สวัสดีครับผมชื่อ : {name}</h1>  
    <p>ที่อยู่ : กรุงเทพมหานคร</p>  
  );  
}
```



การใช้งาน JSX & JavaScript

```
function App() {  
  const name="ก้องรักสยาม"  
  return (  
    <div>  
      <h1>สวัสดีครับผมชื่อ : {name}</h1>  
      <p>ที่อยู่ : กรุงเทพมหานคร</p>  
    </div>  
  );  
}
```

การใช้งาน JSX & JavaScript

```
function App() {  
  const name="ก้องรักสยาม"  
  
  return (  
    <div>  
      <h1>สวัสดีครับผมชื่อ : {name}</h1>  
      <p>ที่อยู่ : กรุงเทพมหานคร</p>  
    </div>  
  );  
}
```



รูปแบบการเขียนแบบ div

```
function Hello(){  
  return (  
    <div>  
      <div><h3>สวัสดี React </h3></div>  
    </div>  
  );  
}
```

รูปแบบการเขียนแบบ Section / Article

```
function Hello(){  
  return (  
    <section>  
      <article><h3>สวัสดี React </h3></article>  
    </section>  
  );  
}
```

รูปแบบการเขียนแบบ Fragments <>

```
function Hello(){  
  return (  
    <>  
    <h3>สวัสดี React </h3>  
    </>  
  );  
}
```

รู้จักกับ State

รู้จักกับ State

State คือ ข้อมูลที่ถูกเก็บไว้ในคอมโพเนนต์ สามารถเปลี่ยนแปลงได้
ตามการกระทำของผู้ใช้งาน เช่น สถานะการล็อกอิน ข้อมูลที่กรอกในฟอร์ม
จำนวนครั้งที่กดปุ่ม เป็นต้น

เมื่อมีการเปลี่ยนค่าข้อมูลใน State ก็จะส่งผลให้ คอมโพเนนต์ที่เป็น
เจ้าของ State นั้นอัปเดตหรือแสดงผลหน้าเว็บใหม่ทันที (*re-render*)

การสร้าง State

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

```
[ชื่อ State , ฟังก์ชันที่ใช้เปลี่ยนแปลงข้อมูลใน State ] = useState(ค่าเริ่มต้นของ State)
```

จะได้ Array ที่ Destructuring จาก useState

ตัวอย่างการสร้าง State

```
const [age, setAge] = useState(20)
```

- **useState** มีการส่งค่ากลับมาเป็นอาร์เรย์ที่มีข้อมูล 2 จำนวนและใช้วิธีแยกข้อมูลดังกล่าวด้วย Arrays Destructing ประกอบด้วย
 - **age** คือ ตัวแปรที่เก็บค่าสถานะปัจจุบัน (เริ่มต้นที่ 20)
 - **setAge** ฟังก์ชันที่ใช้สำหรับอัปเดตหรือเปลี่ยนแปลงข้อมูลในตัวแปร age

ตัวอย่างการสร้าง State

```
const [name, setName] = useState ("kongruksiam")
```

```
const [age, setAge] = useState(30)
```

```
const [status, setStatus] = useState(true)
```


การสร้าง State (Array)

รู้จักกับคีย์ (Key)

Keys หมายถึง Property พิเศษที่ทำงานอยู่ใน JSX โดย **Keys** จะมีค่าที่ไม่ซ้ำกัน ถูกกำหนดขึ้นมาเพื่อให้ React สามารถแยกแยะและตรวจสอบได้ว่ามีคอมโพเนนต์ใดบ้างที่มีการเปลี่ยนแปลงการทำงาน เช่น เพิ่ม ลบ แก้ไขหรือสลับตำแหน่ง ทำให้สามารถจัดการการอัปเดต DOM ได้อย่างมีประสิทธิภาพ (ส่วนใหญ่นำมาใช้งานกับ List ของ Component)

รู้จักกับคีย์ (Key)

React จะทำการ Update และ Render เฉพาะคอมโพเนนต์
ที่เปลี่ยนแปลงเท่านั้น (ไม่ได้ *Render* ใหม่ทั้งหมด) ส่งผลให้
สถานะ (State) ของคอมโพเนนต์แต่ละตัวที่อยู่ในรายการนั้นๆ
ไม่สูญหายในระหว่างที่มีการเปลี่ยนแปลงการทำงาน

การสร้าง Component

รูปแบบการสร้างคอมโพเนนต์

- Class Component (แบบดั้งเดิม)
- Functional Component

(โดยทั้งคู่จะเขียน JSX ด้านในส่วนของคำสั่ง Return)

Functional Component

คือ สร้างคอมโพเนนต์ในรูปแบบฟังก์ชันสามารถสร้างใน
ลักษณะของฟังก์ชันแบบปกติ หรือ Arrow Function ได้และจะต้อง
กำหนดให้ตัวอักษรตัวแรกของชื่อฟังก์ชันเป็นตัวพิมพ์ใหญ่เสมอ

Functional Component

ตัวอย่าง

```
function Header() {  
  return <h1>สวัสดี React </h1>;  
}  
  
export default Header;
```

Functional Component

ฟังก์ชันแบบปกติ (1)

```
function Header() {  
    return <h1>สวัสดี React </h1>;  
}  
  
export default Header;
```


Functional Component

ฟังก์ชันแบบปกติ (2)

```
export default function Header() {  
  return <h1>สวัสดี React </h1>;  
}
```

Functional Component

Arrow Function

```
const Header=()=> {  
    return <h1>สวัสดี React </h1>;  
}  
  
export default Header;
```

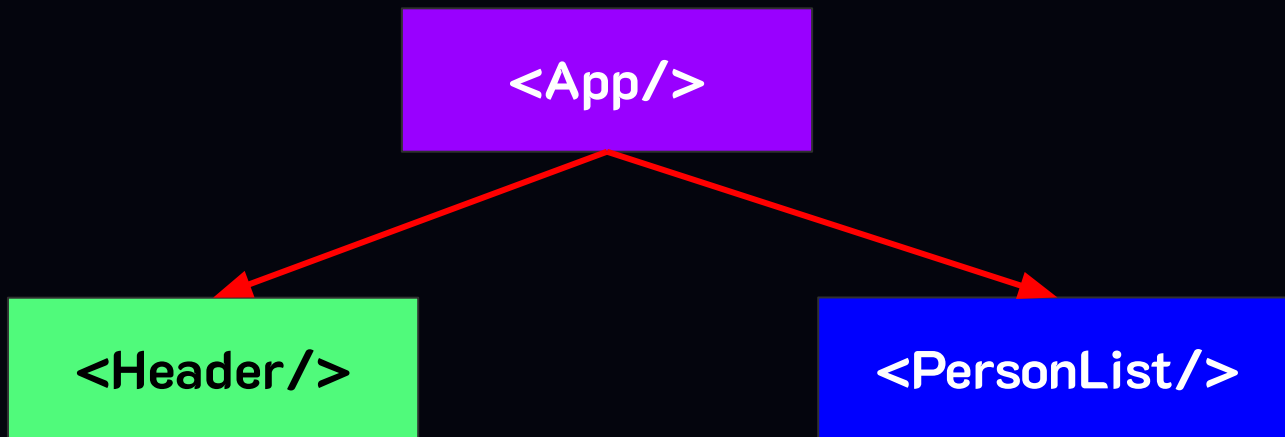
External Component

การสร้างคอมโพเนนต์ไว้เป็นไฟล์ด้านนอกที่มีนามสกุล **.jsx** จากนั้นนำไฟล์ดังกล่าวไปเรียกใช้งานในส่วนอื่นๆที่ต้องการผ่านคำสั่ง `import`

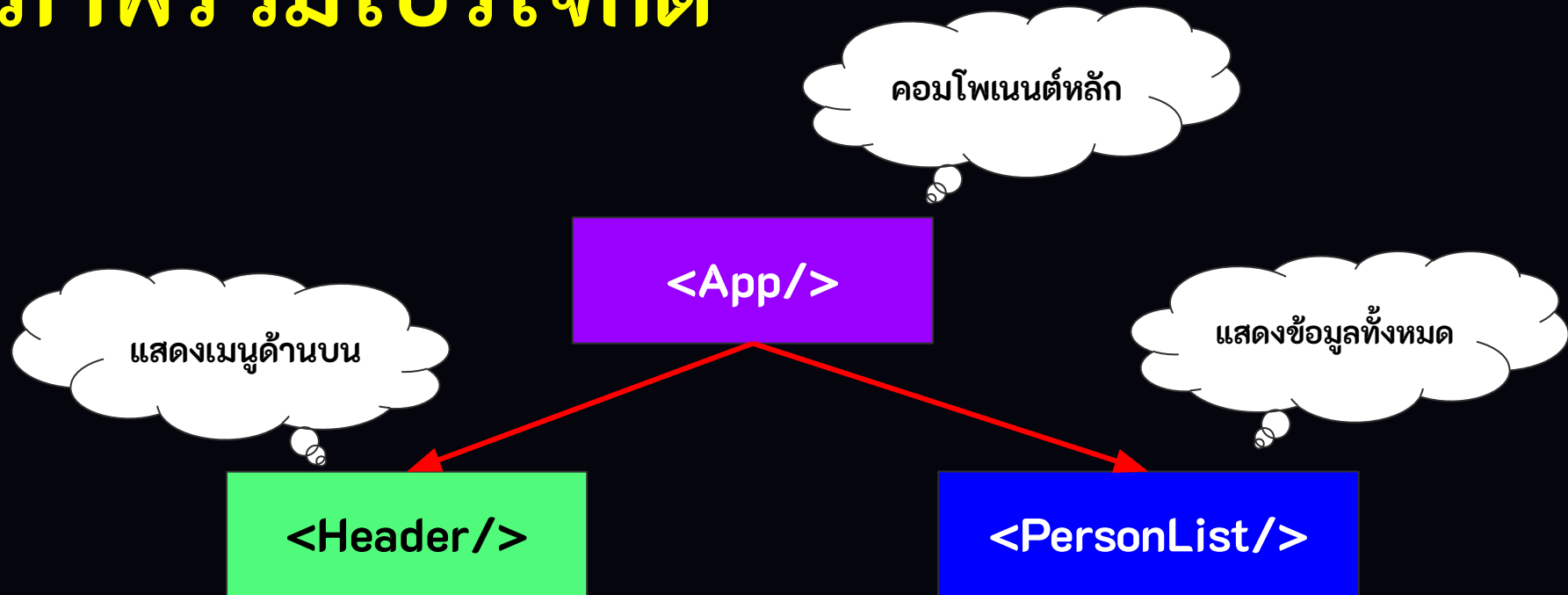
`components/Header.jsx`

```
export default function Header(){  
  
  return <h1>สวัสดี Component</h1>  
  
}
```

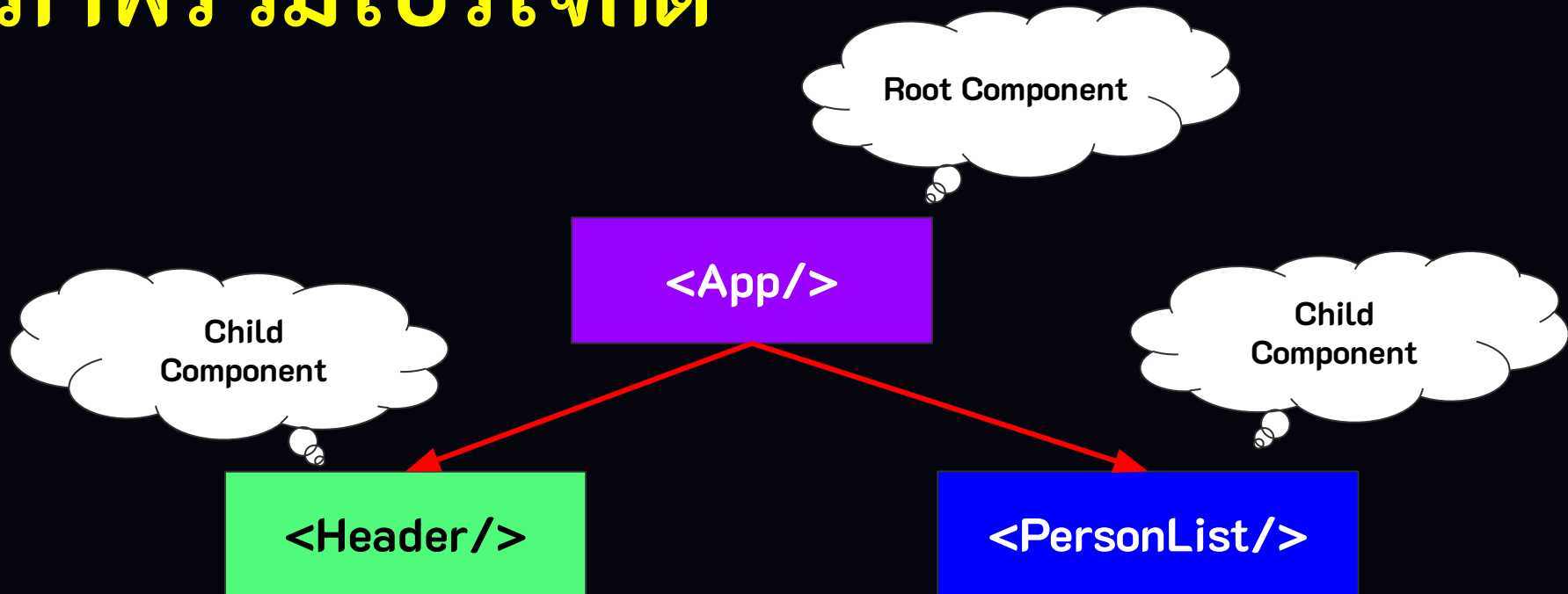
ภาพรวมโปรเจกต์



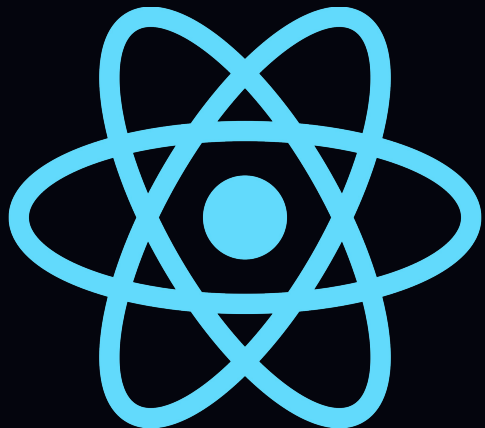
ภาพรวมโปรเจกต์



ภาพรวมโปรเจกต์



ขั้นตอนการสร้าง Component



- สร้างไฟล์ (.jsx) และจัดเก็บลงในโฟลเดอร์ Components เพื่อให้ง่ายต่อการจัดการ
- ออกแบบโครงสร้างการทำงานของ Component และ export เพื่อให้สามารถนำไปใช้งานได้
- เรียกใช้งาน Component ผ่านคำสั่ง import

แสดงรูปภาพใน React

แสดงรูปภาพใน React

ในโปรเจกต์จะมีโฟลเดอร์ **public** และ **src/assets** สำหรับ
ใช้จัดเก็บไฟล์แบบ static เช่น รูปภาพ ไอคอน แต่มีวัตถุประสงค์
ในการใช้งานที่แตกต่างกัน

โฟลเดอร์	การทำงาน
public	<ul style="list-style-type: none">• ไฟล์ในโฟลเดอร์นี้จะไม่ถูกประมวลผล หรือ ไม่ถูก Compile/Bundle ด้วย vite• สามารถเข้าถึงตำแหน่งไฟล์ผ่าน URL โดยตรง เช่น <code></code>• โฟลเดอร์นี้จะเหมาะสำหรับจัดเก็บไฟล์ที่มี URL คงที่ เช่น favicon, รูปโลโก้ (เหมาะสำหรับทำงานกับไฟล์ขนาดใหญ่)

Bundle : การรวมโค้ดและ Asset ต่างๆเข้าด้วยกันเป็นชุดไฟล์ขนาดเล็กเพื่อให้ Browser โหลดเร็วขึ้น

โฟลเดอร์	การทำงาน
src/assets	<ul style="list-style-type: none">• ไฟล์ในโฟลเดอร์นี้จะถูกประมวลผลหรือมีการ Compile/Bundle ด้วย vite• สามารถเข้าถึงตำแหน่งไฟล์ผ่านคำสั่ง import เช่น <code>import logo from '../assets/logo.png'</code>• โฟลเดอร์นี้จะเหมาะสำหรับจัดเก็บไฟล์ขนาดเล็กถึงปานกลาง

Bundle : การรวมโค้ดและ Asset ต่างๆเข้าด้วยกันเป็นชุดไฟล์ขนาดเล็กเพื่อให้ Browser โหลดเร็วขึ้น

สรุปการใช้งาน

- **public** เหมาะสำหรับทำงานไฟล์ static ที่ไม่มีการเปลี่ยนแปลง เช่น favicon , รูปโลโก้ , robots.txt, manifest.json สามารถเรียกใช้งานผ่าน URL โดยตรง
- **src/assets** เหมาะสำหรับทำงานกับไฟล์ static ที่นำมาใช้งานในโค้ดผ่านคำสั่ง **import** และสามารถ Optimize ได้ในตอน Build เช่น โค้ด CSS , ไฟล์ภาพต่างๆที่จะนำมาแสดงผลบนหน้าเว็บ

กำหนด Style ใน React

รูปแบบการกำหนด Style

- **Global Style** คือการกำหนดให้ทุกๆคอมโพเนนต์มีการใช้งาน Style แบบเดียวกันโดยเขียนคำสั่ง CSS ในไฟล์ที่ชื่อว่า index.css (ถูกเรียกใช้งานในไฟล์ main.jsx)
- **Component Style** คือ การกำหนดให้ Style นั้นมีผลเฉพาะ คอมโพเนนต์ที่เราสนใจ

รูปแบบการสร้าง Style

- **Inline Style** คือ การกำหนด Style ลงใน JSX โดยตรงผ่าน Object ของ JavaScript โดยให้มีผลเฉพาะ Element ที่เรียกใช้งาน *(เขียนครอบด้วยเครื่องหมายปีกกาเปิด-ปิดและคั่นด้วยเครื่องหมายโคลอน :)*
- **External Style** คือ การกำหนด Style แยกเป็นไฟล์ด้านนอกที่มีนามสกุล **.css** แล้วนำเข้ามาใช้งานภายในคอมโพเนนต์ที่ต้องการ เหมาะสำหรับการกำหนดให้หลายๆคอมโพเนนต์ใช้รูปแบบ Style เดียวกันผ่านการเขียน Style แค่ครั้งเดียว

ตัวอย่าง Inline Style

```
const myStyle={  
  color:"white",  
  background:"red"  
}
```

Property	value
color	white
background	red

```
<div style = {myStyle}>My React</div>
```


ตัวอย่าง External Style

style.css

```
div{  
    color:"white";  
    background:"red";  
}
```

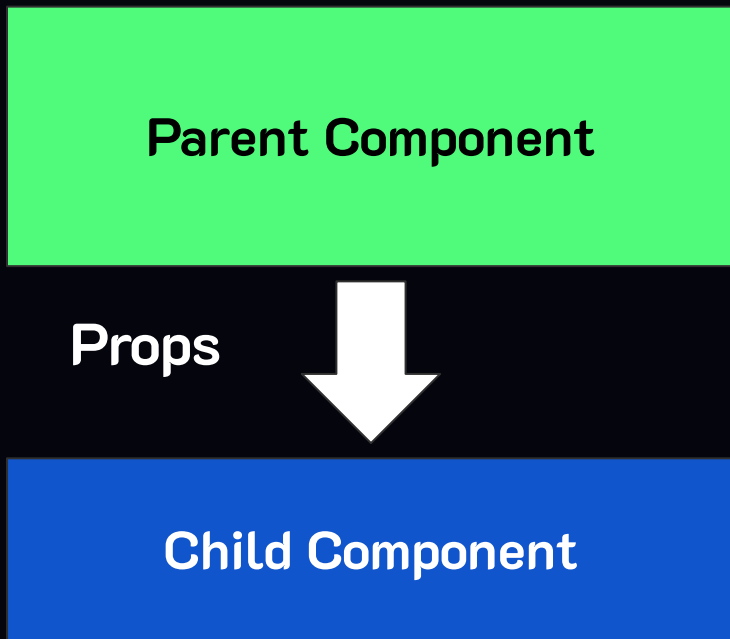
import "./style.css"

```
function Header(){  
    return (  
        <>  
        <div>สวัสดี React </div>  
        </>  
    );  
}
```

รู้จักกับ Props (Properties)

Props คือ ข้อมูลที่เราสามารถส่งเข้าไปทำงานในคอมโพเนนต์ได้
โดยข้อมูลดังกล่าวจะถูกส่งจากคอมโพเนนต์ที่อยู่สูงกว่า (*Parent Component*) ไปยังคอมโพเนนต์ที่อยู่ต่ำกว่า (*Child Component*)

รู้จักกับ Props



การทำงานของ Props

การส่งค่า Props จาก Parent Component ไปยัง Child Component

มีวัตถุประสงค์ดังนี้

- กำหนดข้อมูลสำหรับแสดงเนื้อหาใน Child Component
- กำหนดความสามารถบางอย่างให้กับ Child Component

ในการติดต่อหรือสื่อสารข้อมูลหากันระหว่างคอมโพเนนต์

การสร้าง Props

Props แบบค่าเดียว

<ชื่อคอมโพเนนต์ ชื่อพร็อพ =ค่าที่กำหนดให้พร็อพ/>

Props แบบหลายค่า

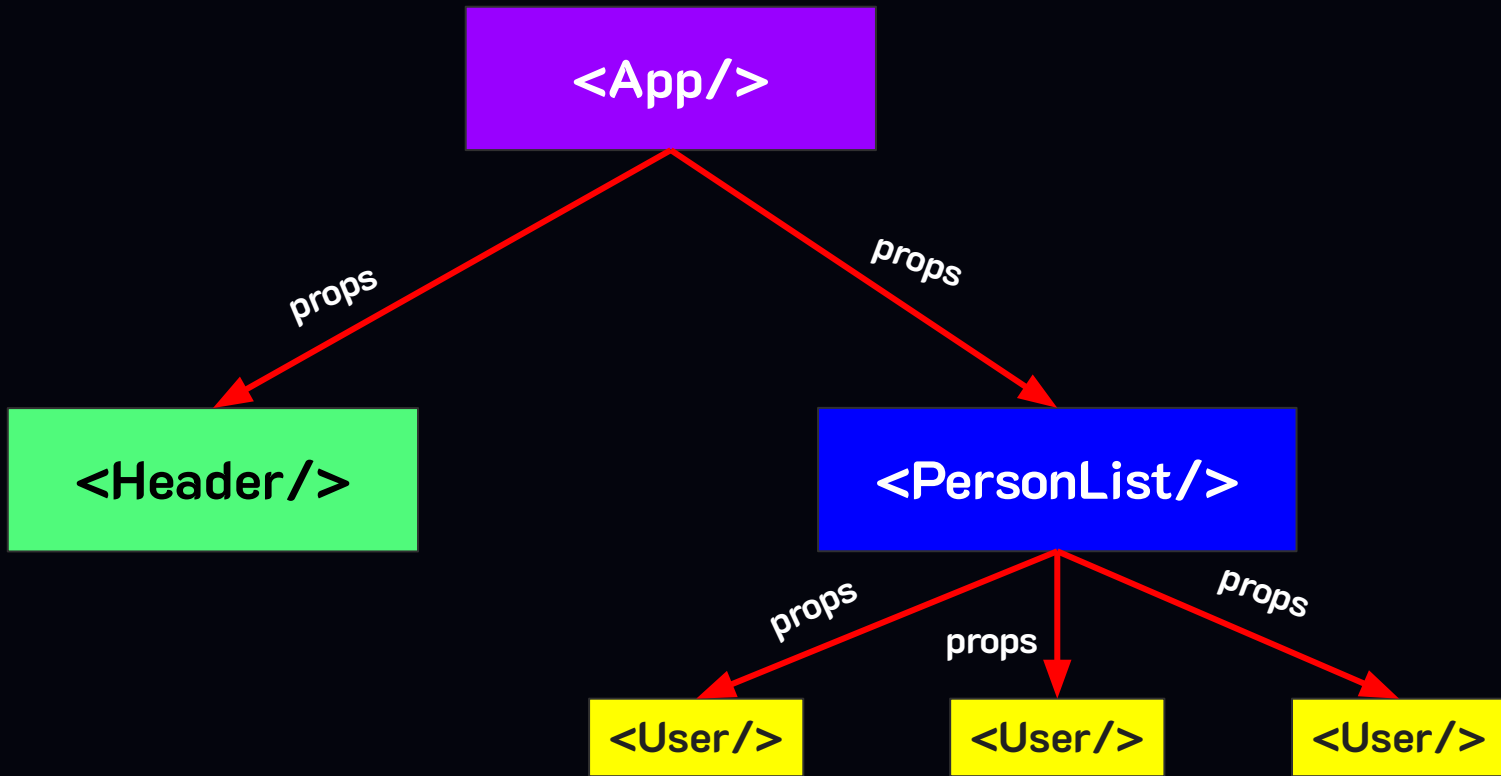
<ชื่อคอมโพเนนต์ ชื่อพร็อพ =ค่าที่1 ชื่อพร็อพ =ค่าที่2 />

Props สามารถกำหนดค่าได้หลากหลายชนิด

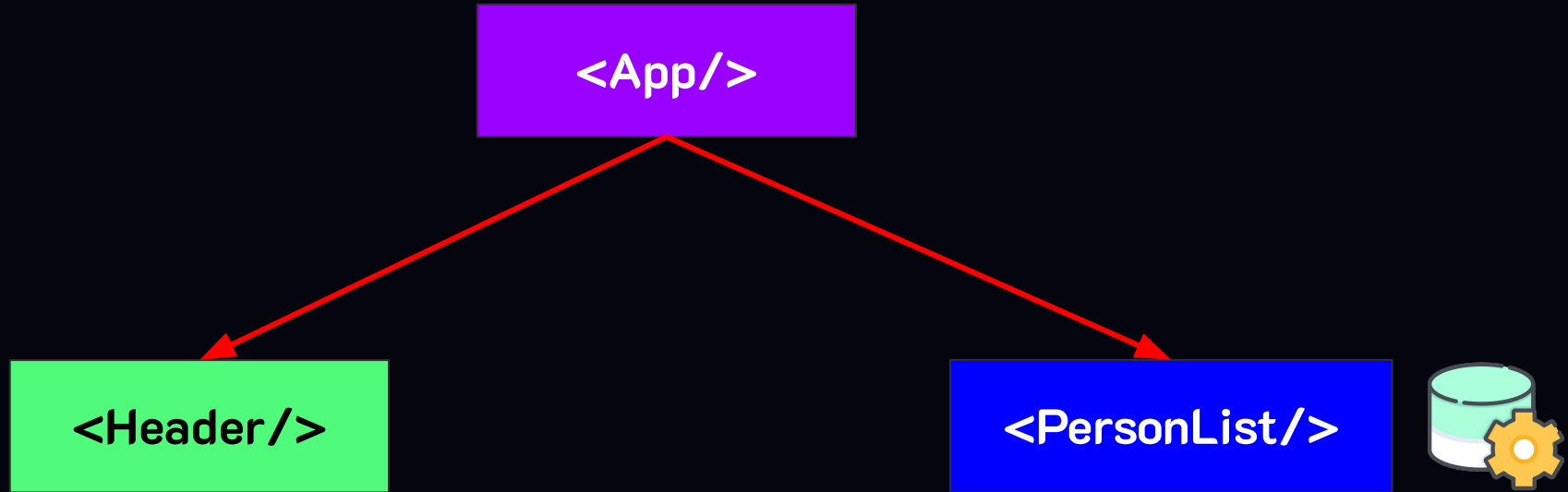
- **string** : `<Component props="หน้าหลัก" />`
- **number** : `<Component props={5} />`
- **boolean** : `<Component props={true} />`
- **array** : `<Component props={['item1', 'item2']} />`
- **object** : `<Component props={{name: 'Kong', age: 25}} />`
- **function** : `<Component props={() => console.log('คลิก')} />`

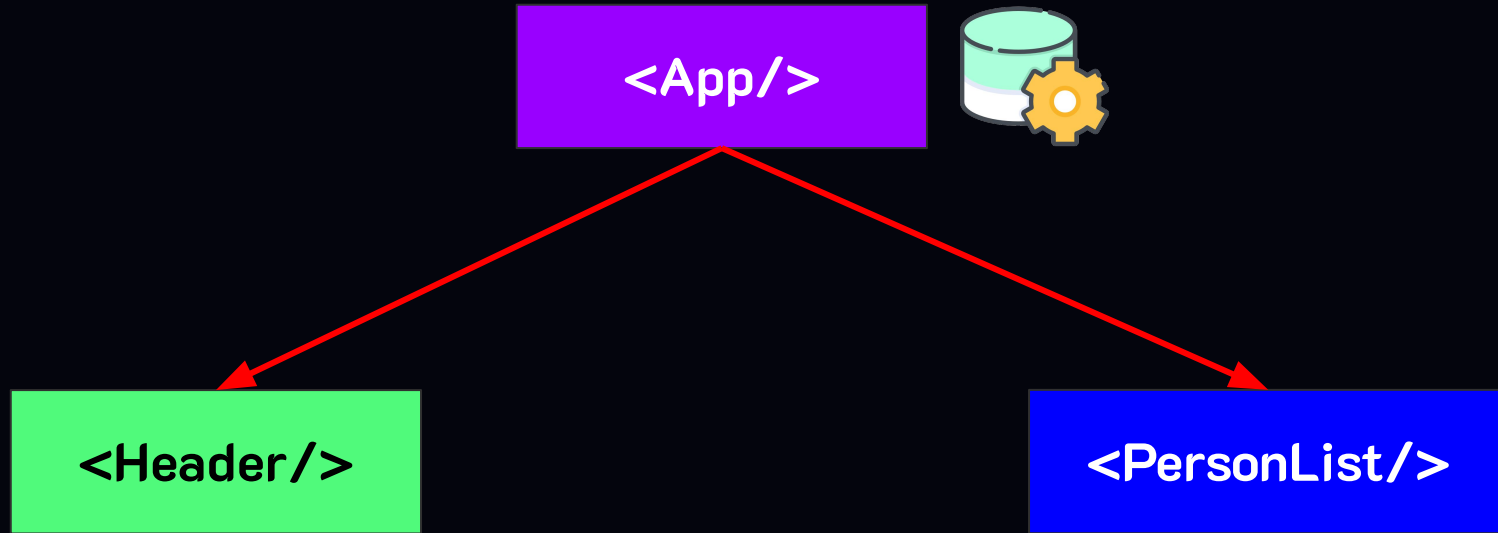
Props สามารถกำหนดค่าได้หลากหลายชนิด

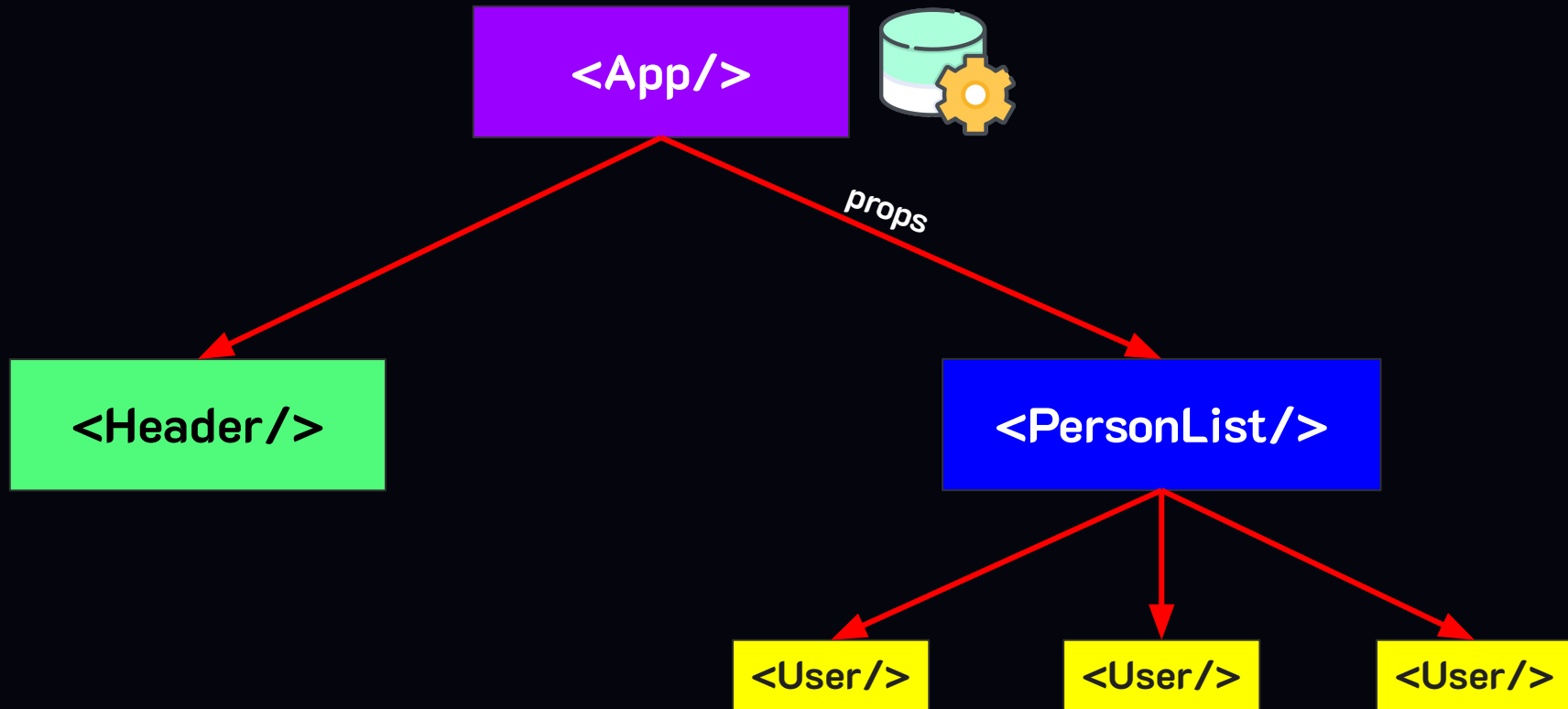
- **string** : `<Component title="หน้าหลัก" />`
- **number** : `<Component count={5} />`
- **boolean** : `<Component isAdmin={true} />`
- **array** : `<Component items={['item1', 'item2']} />`
- **object** : `<Component user={{name: 'Kong', age: 25}} />`
- **function** : `<Component onClick={() => console.log('คลิก')} />`

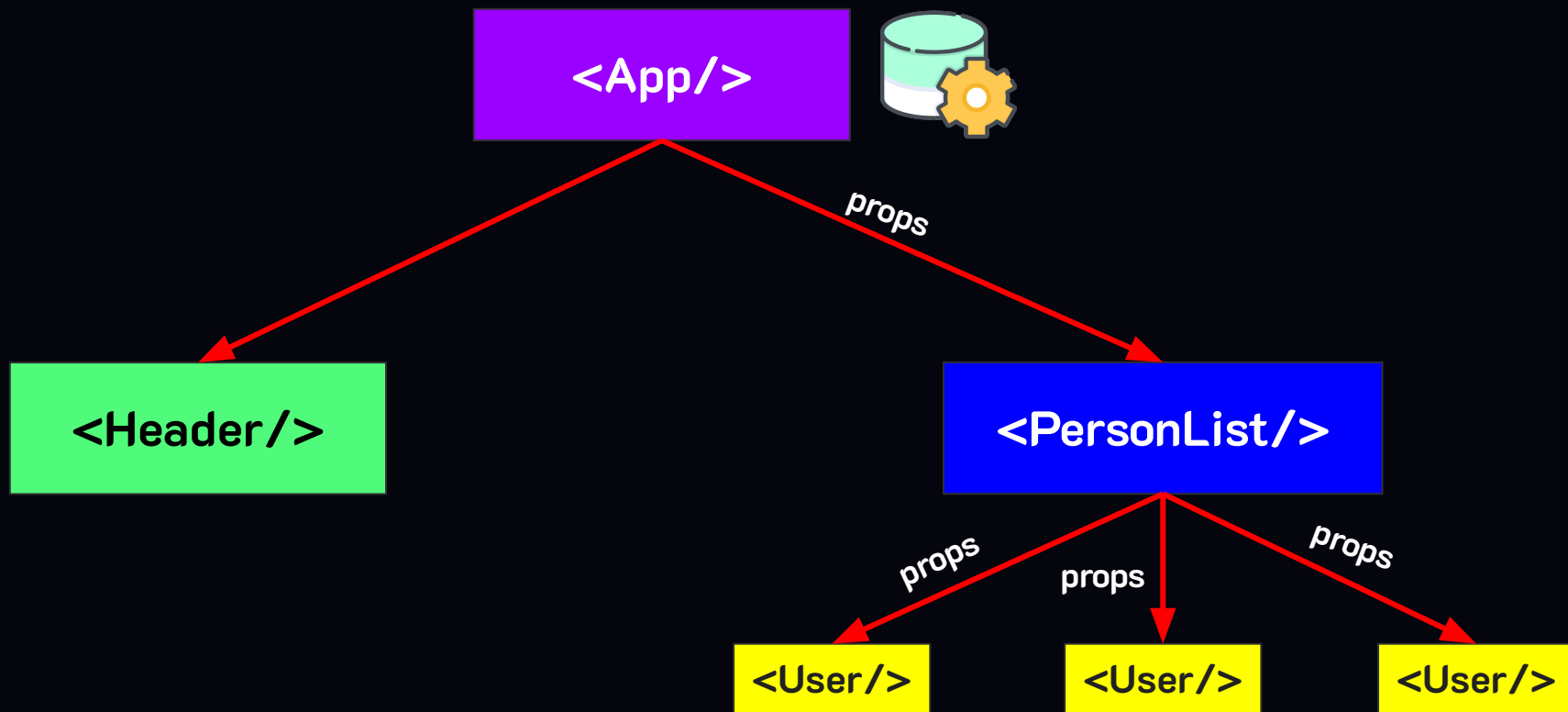


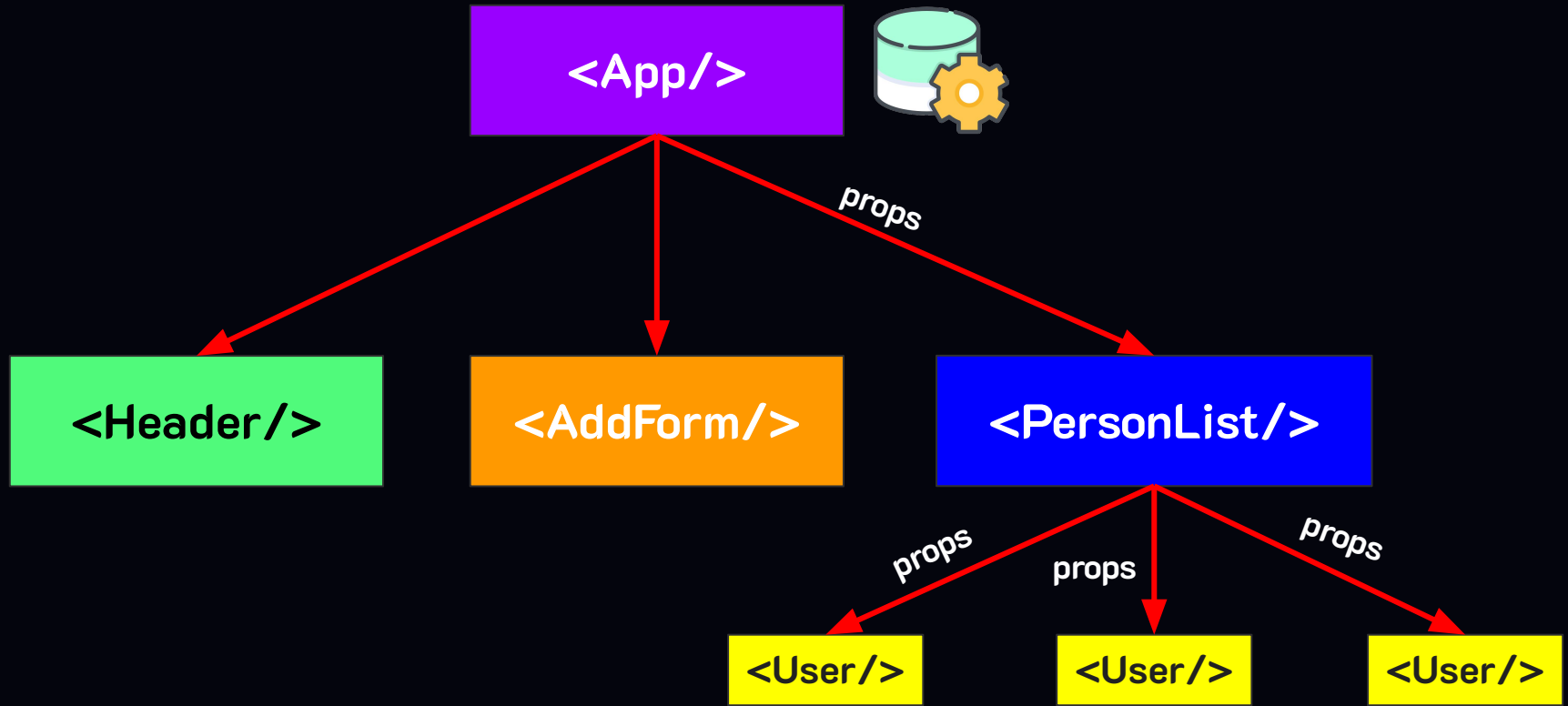
เจาะลึกการใช้งาน Props











useEffect

- **Effect** คือ ผลกระทบหรือผลข้างเคียง (Side Effect)
- **useEffect** คือ การจัดการกับผลกระทบที่เกิดขึ้นภายในคอมโพเนนต์ เช่น การดึงข้อมูลจาก API , การติดตาม Event หรือการเปลี่ยนแปลง DOM ถูกนำมาใช้งานเพื่อต้องการทราบว่าเกิดการอัปเดตหรือเปลี่ยนแปลงอะไรขึ้นบ้างภายใน คอมโพเนนต์ *จนส่งผลให้เกิดการ **Render UI ใหม่*** โดยสาเหตุหลักๆที่คอมโพเนนต์ Render ใหม่จะมาจากการเปลี่ยนแปลงค่าที่อยู่ภายใน Props และ State นั้นเอง

โครงสร้างคำสั่ง (1)

```
import {useEffect} from "react"
```

```
useEffect(()=>{
```

```
    //คำสั่งต่างๆ
```

```
})
```

* ทำงานคำสั่งที่ระบุใน `useEffect` ทุกครั้งเมื่อคอมโพเนนต์มีการเรนเดอร์

* เรนเดอร์ คือ การโหลดคอมโพเนนต์หรือการแสดงผลบนหน้าเว็บ

โครงสร้างคำสั่ง (2)

```
import {useEffect} from "react"
```

```
useEffect(()=>{
```

```
    //คำสั่งต่างๆ
```

```
},[])
```

* ทำงานคำสั่งใน `useEffect` เพียงครั้งเดียวเมื่อคอมโพเนนต์ถูกเรนเดอร์ครั้งแรก (เหมาะสำหรับการดึงข้อมูลมาใช้งานในตอนเริ่มต้น)

โครงสร้างคำสั่ง (3)

```
import {useEffect} from "react"
```

```
useEffect(()=>{
```

```
  //คำสั่งต่างๆ
```

```
},[state])
```

**เมื่อมีการเปลี่ยนแปลงข้อมูลใน State ที่กำหนด จะเรนเดอร์คอมโพ*

เนนต์และทำงานคำสั่งที่อยู่ใน useEffect

Local Storage

คือ Web Storage API สำหรับใช้เก็บข้อมูลในฝั่งของผู้ใช้งาน (Client-Side) โดยข้อมูลที่ถูกเก็บไว้ใน Local Storage นั้นจะเก็บแบบถาวรไม่มีวันหมดอายุจนกว่าจะถูกสั่งให้ลบ ถึงแม้ผู้ใช้งานจะปิดเบราว์เซอร์หรือปิดเครื่องไปแล้วก็ตาม ข้อมูลก็จะยังคงอยู่เสมอ (นอกจากจะทำการล้างข้อมูล)

คุณสมบัติของ Local Storage

- เก็บข้อมูลเป็นคู่ key-value (เหมือน Object)
- ไม่สามารถเก็บ Object ได้โดยตรง (ต้องแปลงเป็น JSON ก่อน)
- ข้อมูลที่เก็บไว้จะคงอยู่ถาวรจนกว่าจะถูกลบ
- ขนาดของพื้นที่จัดเก็บข้อมูลประมาณ 5-10 MB (ขึ้นอยู่กับเบราว์เซอร์)
- ใช้ได้เฉพาะภายในโดเมนเดียวกันเท่านั้น (Same-origin policy) โดยแต่ละโดเมนจะมีพื้นที่ Local Storage แยกกัน

ข้อควรทราบ

- ไม่ควรเก็บข้อมูลสำคัญ เช่น ข้อมูลส่วนตัว หรือ รหัสผ่านที่ใช้เข้าสู่ระบบ
- สามารถดูหรือเปลี่ยนแปลงข้อมูลได้ผ่าน *Developer Tools*
- ข้อมูลที่มีขนาดใหญ่เกินไปอาจถูกตัดทิ้งและส่งผลให้บันทึกข้อมูลไม่สำเร็จ
- รองรับเบราว์เซอร์ที่ทันสมัย (ควรตรวจสอบว่าเบราว์เซอร์รองรับหรือไม่ ก่อนใช้งาน)

โครงสร้างคำสั่ง

- การบันทึกข้อมูล

```
localStorage.setItem('key', 'value');
```

- การดึงข้อมูล

```
localStorage.getItem('key');
```

โครงสร้างคำสั่ง

- การลบข้อมูล (เฉพาะคีย์ที่สนใจ)

```
localStorage.removeItem('key');
```

- การลบข้อมูล Local Storage ทั้งหมดของโดเมนนั้น

```
localStorage.clear();
```



React Icons

คือ ไลบรารีที่รวบรวมไอคอนจากชุดไอคอนยอดนิยมต่าง ๆ เช่น Font Awesome, Material Design, Bootstrap Icons และอื่น ๆ โดยจะแปลงให้อยู่ในรูปแบบ **React Components** ซึ่งทำให้สามารถใช้งานไอคอนในโปรเจกต์ React ได้ โดยขั้นตอนการติดตั้งและการใช้งาน สามารถศึกษาเพิ่มเติมได้ที่ : <https://react-icons.github.io/react-icons/>

รูปแบบการเรียกใช้งาน React Icons

`react-icons/xx` โดย xx คือ ตัวย่อของชุดไอคอน ตัวอย่าง เช่น

- bs = Bootstrap Icons
- fa = Font Awesome
- md = Material Design
- ai = Ant Design Icons

การปรับแต่ง Icons (Props)

- **size** กำหนดขนาดของไอคอน
- **color** กำหนดสีของไอคอน
- **className** กำหนด CSS class ให้กับไอคอน
- **style** กำหนด Inline Style ให้กับไอคอน

