

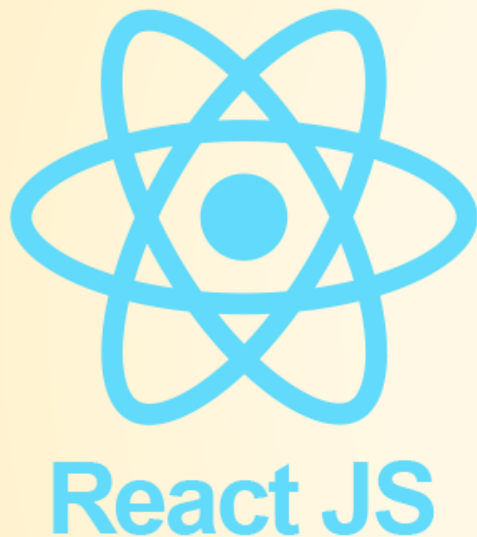


# THIẾT KẾ GIAO DIỆN FRONTEND VỚI NEXT.JS

Trình bày: Bùi Bá Nguyên

Ryomo Vietnam Solutions Co. Ltd

# React



- **React** là một thư viện JavaScript mã nguồn mở do Facebook phát triển, dùng để xây dựng giao diện người dùng (UI) cho các ứng dụng web.
- Với React, bạn có thể tạo ra các thành phần (component) nhỏ, dễ tái sử dụng giúp xây dựng giao diện web một cách hiệu quả và hiện đại.
- React chỉ tập trung vào phần “hiển thị” (View) trong ứng dụng.

# React

- **Đặc điểm nổi bật của React**

- Component-Based Architecture (Kiến trúc dựa trên Component):
  - Tất cả mọi thứ là Component: Trong React, giao diện người dùng (UI) được chia nhỏ thành các khối độc lập, có thể tái sử dụng gọi là Component. Mỗi Component có logic và giao diện riêng.
    - Ví dụ: Một trang web có thể được chia thành các Component như Header, Sidebar, ProductCard, Footer, v.v.
  - Tái sử dụng (Reusability): chúng ta có thể viết một Component một lần và sử dụng nó ở nhiều nơi khác nhau trong ứng dụng, hoặc thậm chí trong các dự án khác.
  - Quản lý dễ dàng: Việc chia nhỏ ứng dụng giúp dễ dàng quản lý, bảo trì và phát triển các phần khác nhau của UI.

# React

- **Đặc điểm nổi bật của React**

- Declarative Programming (Lập trình Khai báo)
  - Thay vì mô tả từng bước để thay đổi UI, React cho phép bạn mô tả trạng thái cuối cùng của UI mà bạn muốn. React sẽ tự động cập nhật UI để phù hợp với trạng thái đó.
  - Ví dụ: Thay vì nói "tìm phần tử này, xóa nó, rồi tạo một phần tử mới với dữ liệu này và thêm nó vào", ta chỉ cần nói "đây là dữ liệu mới, hãy hiển thị nó". Điều này làm cho code dễ đọc, dễ hiểu và ít lỗi hơn.
- Virtual DOM (DOM ảo)
  - Hiệu suất cao: Thay vì thao tác trực tiếp với DOM (Document Object Model) của trình duyệt (một quá trình tốn kém về hiệu suất), React sử dụng một bản sao nhẹ trong bộ nhớ gọi là Virtual DOM.
  - Cơ chế hoạt động:
    - Khi trạng thái của Component thay đổi, React sẽ xây dựng một Virtual DOM mới.
    - Nó so sánh Virtual DOM mới với Virtual DOM cũ để tìm ra sự khác biệt (diffing).
    - Chỉ những thay đổi nhỏ nhất được áp dụng vào DOM thực tế của trình duyệt.
  - Lợi ích: Giảm thiểu số lượng thao tác trực tiếp lên DOM, dẫn đến hiệu suất cao hơn và trải nghiệm người dùng mượt mà hơn.

# React

- **Đặc điểm nổi bật của React**

- Cú pháp mở rộng của JavaScript: JSX cho phép bạn viết mã giống HTML ngay trong file JavaScript. Mặc dù không bắt buộc, nhưng JSX được khuyến khích sử dụng vì nó giúp việc mô tả UI trở nên trực quan và dễ đọc hơn.
- Pre-compilation: Trình duyệt không thể hiểu trực tiếp JSX. Nó cần được biên dịch (transpile) thành JavaScript thuần túy bằng các công cụ như Babel trước khi chạy.
- Hiện dự án đang sử dụng TSX

⇒ Dạng file TSX là gì?

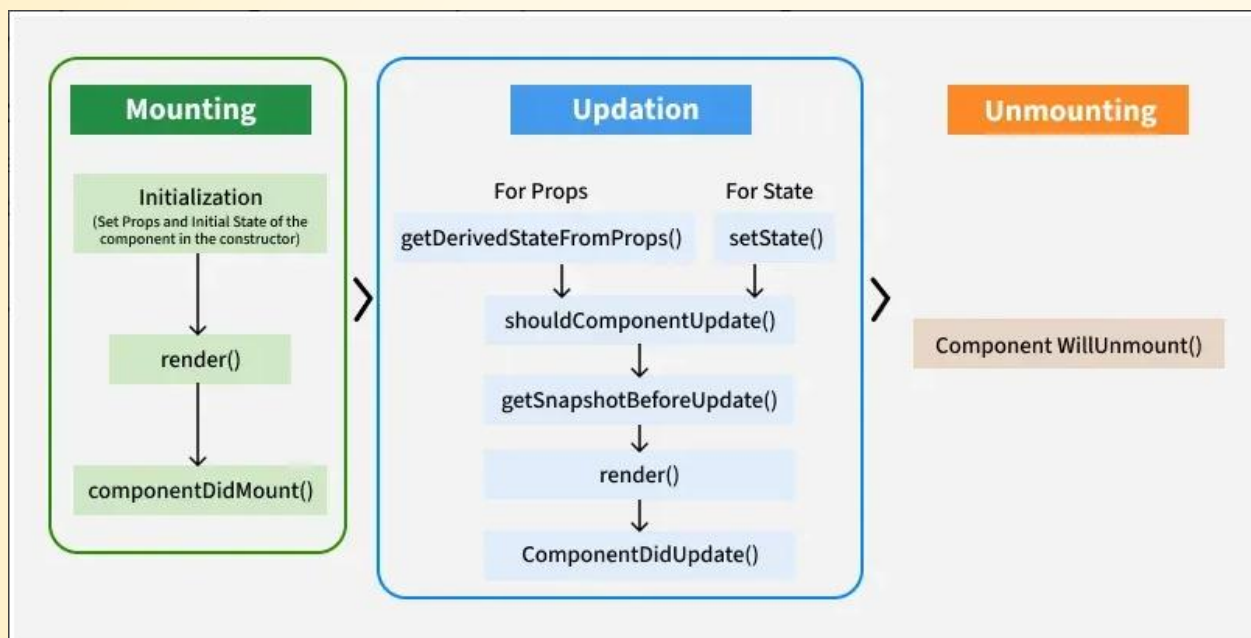
# React

- **Định dạng File .tsx**

- Dạng file TSX trong TypeScript là một phần mở rộng tập tin đặc biệt, được sử dụng khi bạn viết code TypeScript có chứa cú pháp JSX.
- Nói một cách đơn giản:
  - TypeScript (.ts): Là JavaScript nhưng có thêm hệ thống kiểu dữ liệu tĩnh. Điều này giúp bạn bắt lỗi sớm hơn trong quá trình phát triển và làm cho code của bạn dễ bảo trì hơn.
  - JSX: Là một cú pháp mở rộng của JavaScript, trông giống như HTML. Nó thường được dùng trong các thư viện giao diện người dùng (UI) như React để mô tả cấu trúc của UI một cách trực quan.
- Khi **kết hợp cả hai** – tức là chúng ta muốn viết code TypeScript nhưng lại cần dùng cú pháp JSX để xây dựng giao diện – ta sẽ sử dụng các file có đuôi .TSX.

# Vòng đời của React

- Được chia làm 3 phần chính



## • Mounting - Khởi tạo

- Component được tạo, khởi tạo state và các phương thức, sau đó được render và gắn vào DOM.

## • Updation: Cập nhật

- Khi props hoặc state thay đổi, component có thể được cập nhật và render lại, hoặc bỏ qua việc render.

## • Unmounting: Hủy bỏ

- Component bị gỡ bỏ khỏi DOM và các tài nguyên liên quan được dọn dẹp.

# Vòng đời của React

## ▪ Mounting (Khởi tạo)

- `constructor()`: được gọi đầu tiên khi một instance của component được tạo.
- `render()`: trả về JSX (hoặc null, false) để React biết cần hiển thị gì lên UI.
- `componentDidMount()` (class component) hoặc `useEffect(..., [])` (function component): gọi ngay sau khi component và tất cả các child component của nó đã được render và được gắn vào DOM.

## ▪ Updating

- `getDerivedStateFromProps(props, state)`: được gọi ngay trước khi `render()` được gọi, cả trong giai đoạn mounting và updating.
- `shouldComponentUpdate()`:
  - Được gọi trước khi re-rendering diễn ra, cho phép bạn kiểm soát liệu component có nên re-render hay không.
  - Nếu trả về false, quá trình re-render sẽ bị hủy bỏ (bao gồm cả `render()`, `getSnapshotBeforeUpdate()`, `componentDidUpdate()`).
- `componentDidUpdate()`:
  - Được gọi ngay sau khi component đã được cập nhật và re-render vào DOM.
  - Là nơi lý tưởng để thực hiện các side effects sau khi UI đã được cập nhật, ví dụ: kiểm tra thay đổi props và thực hiện hành động dựa trên đó, cập nhật DOM thủ công (rất hiếm khi cần).



# Vòng đời của React

- **Unmounting**

- `componentWillUnmount()`:

- Được gọi ngay trước khi component bị hủy bỏ và gỡ bỏ khỏi DOM.
    - Đây là nơi lý tưởng để thực hiện các tác vụ dọn dẹp (cleanup) để tránh rò rỉ bộ nhớ.
    - Ví dụ: Hủy bỏ event listeners, xóa timers, hủy bỏ các network requests đang chờ.

# Next.js

- Next.js là một framework xây dựng trên nền tảng React.
- Next.js giúp bạn phát triển các ứng dụng web React dễ dàng hơn, chuyên nghiệp hơn.
- Ngoài những gì React làm được, Next.js còn cung cấp thêm nhiều tính năng mạnh mẽ như:
  - **Tạo trang động và tĩnh dễ dàng**
  - **Hỗ trợ SEO tốt hơn** (giúp website dễ lên top Google)
  - **Tối ưu hiệu suất** (trang tải nhanh hơn)
  - **Routing (chuyển trang)** đơn giản

# Summary

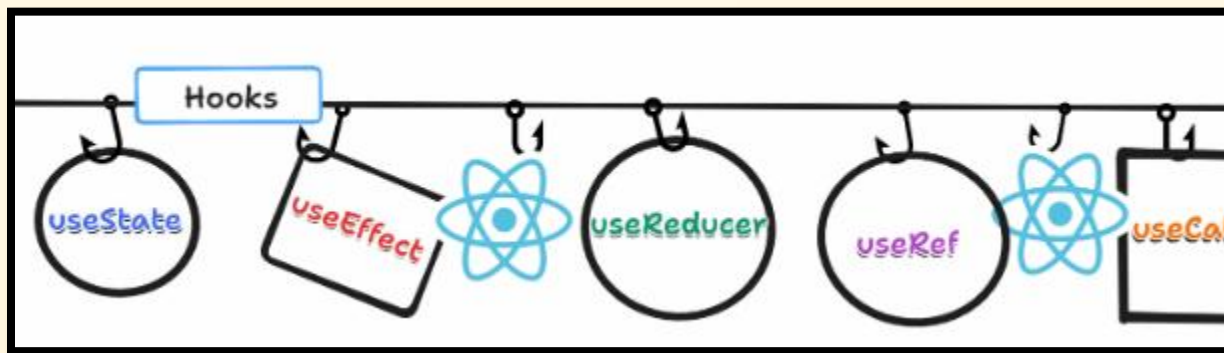
- **React:** Quản lý vòng đời cho từng component.
- **Next.js:** Quản lý vòng đời của trang (page) bao gồm lấy dữ liệu, render, sau đó chuyển sang quản lý vòng đời component như React.

## Next.js:

Next.js là một framework phát triển web dựa trên React, được xây dựng bởi Vercel. Nó giúp lập trình viên xây dựng các ứng dụng web hiện đại với nhiều tính năng mạnh mẽ đã được tích hợp sẵn.

# React Hooks

React Hook là một tính năng của React (từ phiên bản 16.8 trở lên) cho phép bạn sử dụng các chức năng như quản lý state (trạng thái), thực hiện các thao tác vòng đời (lifecycle) và nhiều tính năng khác ngay bên trong component dạng hàm (function component), mà trước đây chỉ có class component mới làm được.



# React Hooks – Ví dụ về Class Component

```
import React from 'react';

interface MySimpleComponentProps {
  message: string; // Component này sẽ nhận một prop tên là 'message' có kiểu string
  count?: number; // Một prop tùy chọn 'count' có kiểu number
}

interface MySimpleComponentState {
  clicks: number; // State của component này sẽ có 'clicks' là một số
}

class MySimpleComponent extends React.Component<MySimpleComponentProps, MySimpleComponentState> {
  constructor(props: MySimpleComponentProps) { // Constructor là nơi khởi tạo state và bind các phương thức
    super(props); // Luôn gọi super(props) trong constructor của Class Component

    // Khởi tạo state ban đầu
    this.state = {
      clicks: 0
    };
  }

  // Phương thức xử lý sự kiện
  handleClick = () => {
    this.setState(prevState => ({
      clicks: prevState.clicks + 1
    }));
  };

  render() {
    const { message, count } = this.props; // Truy cập props thông qua this.props
    const { clicks } = this.state; // Truy cập state thông qua this.state

    return (
      <div>
        <h1>{message}</h1> { /* Sử dụng prop message */ }
        {count && <p>Số đếm ban đầu: {count}</p>} { /* Hiển thị prop count nếu có */ }
        <p>Số lần click: {clicks}</p> { /* Hiển thị state clicks */ }
        <button onClick={this.handleClick}>Click me!</button> { /* Nút để thay đổi state */ }
      </div>
    );
  }
}

export default MySimpleComponent;
```

Phương thức render là bắt buộc

## ▪ Class Component:

- **MySimpleComponentProps** : định nghĩa kiểu cho Props (các thuộc tính mà component nhận vào)
- **MySimpleComponentState** : định nghĩa kiểu cho State (trạng thái) của component
- **MySimpleComponent** : định nghĩa Class Component với TypeScript

# React Hooks : useState

Quản lý **state** (trạng thái) bên trong function component.

Cho phép component lưu trữ và thay đổi dữ liệu.

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

function Counter() {
  const [count, setCount] = useState(0); // Khai báo state "count"

  return (
    <div>
      <p>Bạn đã bấm {count} lần</p>
      <button onClick={() => setCount(count + 1)}>
        Bấm tôi!
      </button>
    </div>
  );
}
```

## ▪ Giải thích:

- useState(0) tạo biến count (giá trị ban đầu là 0) và hàm setCount để cập nhật.
- Khi bấm nút, count tăng lên.

# React Hooks : useEffect

Thực hiện side effect (tác vụ phụ) như gọi API, cập nhật DOM, setTimeout, thao tác với localStorage(lưu bộ nhớ tạm)...

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

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

  useEffect(() => {
    document.title = `Bạn đã bấm ${count} lần`;
  }, [count]); // Chạy lại mỗi khi count thay đổi

  return (
    <div>
      <p>Bạn đã bấm {count} lần</p>
      <button onClick={() => setCount(count + 1)}>
        Bấm tôi!
      </button>
    </div>
  );
}
```

## ▪ Giải thích:

- Mỗi lần count đổi, useEffect sẽ đổi tiêu đề trình duyệt thành số lần bấm.
- Nếu [dependency] là mảng rỗng ([]): effect chỉ chạy 1 lần sau khi component mount.
- Nếu [dependency] chứa state/props: effect sẽ chạy mỗi khi giá trị trong mảng thay đổi.
- `useEffect(() => { // code thực thi khi component mount hoặc update return () => { // cleanup (dọn dẹp), chạy khi unmount hoặc trước khi chạy effect lần tiếp theo }; }, [dependency]);`

# React Hooks : useCallback

Memoize (“ghi nhớ”) một hàm giữa các lần render, chỉ tạo lại hàm khi dependency thay đổi.

Giúp tránh tạo ra hàm mới không cần thiết, tối ưu hiệu suất, đặc biệt khi truyền hàm xuống component con (props) hoặc dùng với useEffect/useMemo.

```
import React, { useState, useCallback } from "react";

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

  const increment = useCallback(() => {
    setCount(prev => prev + 1); // Sử dụng giá trị trước
  }, []); // Không phụ thuộc gì, hàm chỉ tạo 1 lần

  return (
    <div>
      <p>Bạn đã bấm {count} lần</p>
      <button onClick={increment}>
        Bấm tôi!
      </button>
    </div>
  );
}
```



# React Hooks : kết hợp cả ba hook

```
import React, { useState, useEffect, useCallback } from "react";

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

  useEffect(() => {
    document.title = `Bạn đã bấm ${count} lần`;
  }, [count]);

  const increment = useCallback(() => {
    setCount(prev => prev + 1);
  }, []);

  return (
    <div>
      <p>Bạn đã bấm {count} lần</p>
      <button onClick={increment}>
        Bấm tôi!
      </button>
    </div>
  );
}
```

**useState** để tạo state lưu trữ giá trị của count.

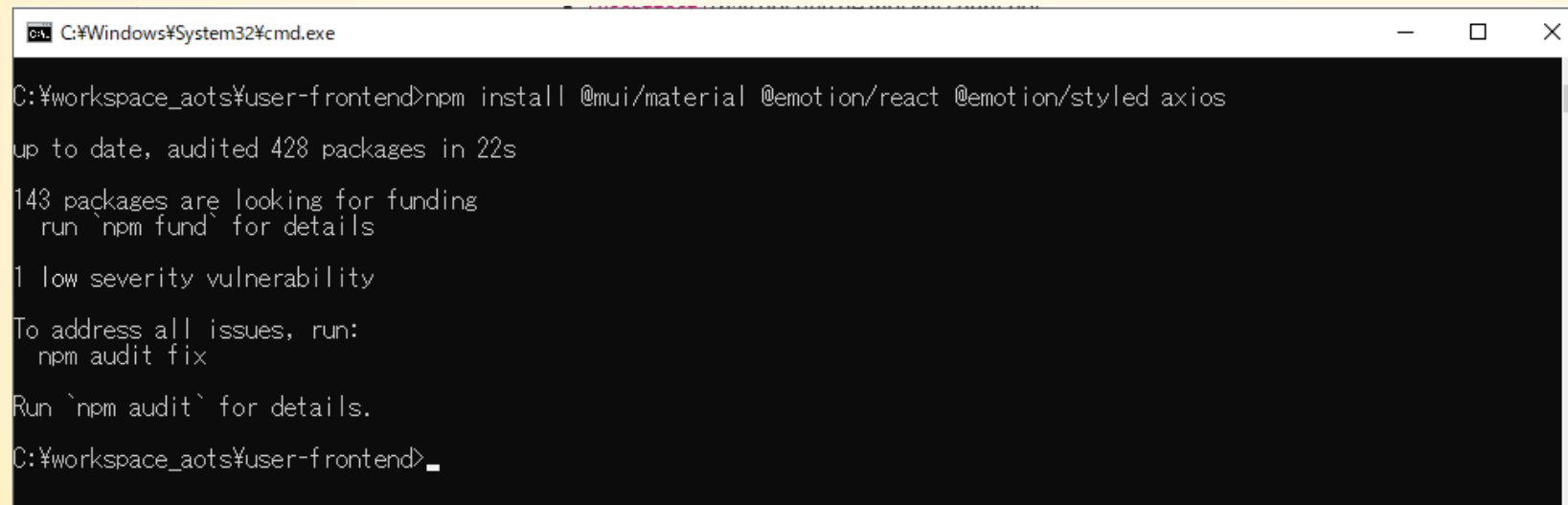
**useEffect** thay đổi tiêu đề mỗi khi count đổi.

**useCallback** tối ưu hàm tăng count.

# Cài MUI và Axios

Axios là một thư viện JavaScript nhằm thực hiện các yêu cầu HTTP từ trình duyệt hoặc Node.js. Nó được sử dụng rộng rãi vì sự đơn giản, linh hoạt và nhiều tính năng hữu ích.

```
npm install @mui/material @emotion/react @emotion/styled axios
```



```
C:\Windows\System32\cmd.exe

C:\workspace_aots\user-frontend>npm install @mui/material @emotion/react @emotion/styled axios

up to date, audited 428 packages in 22s

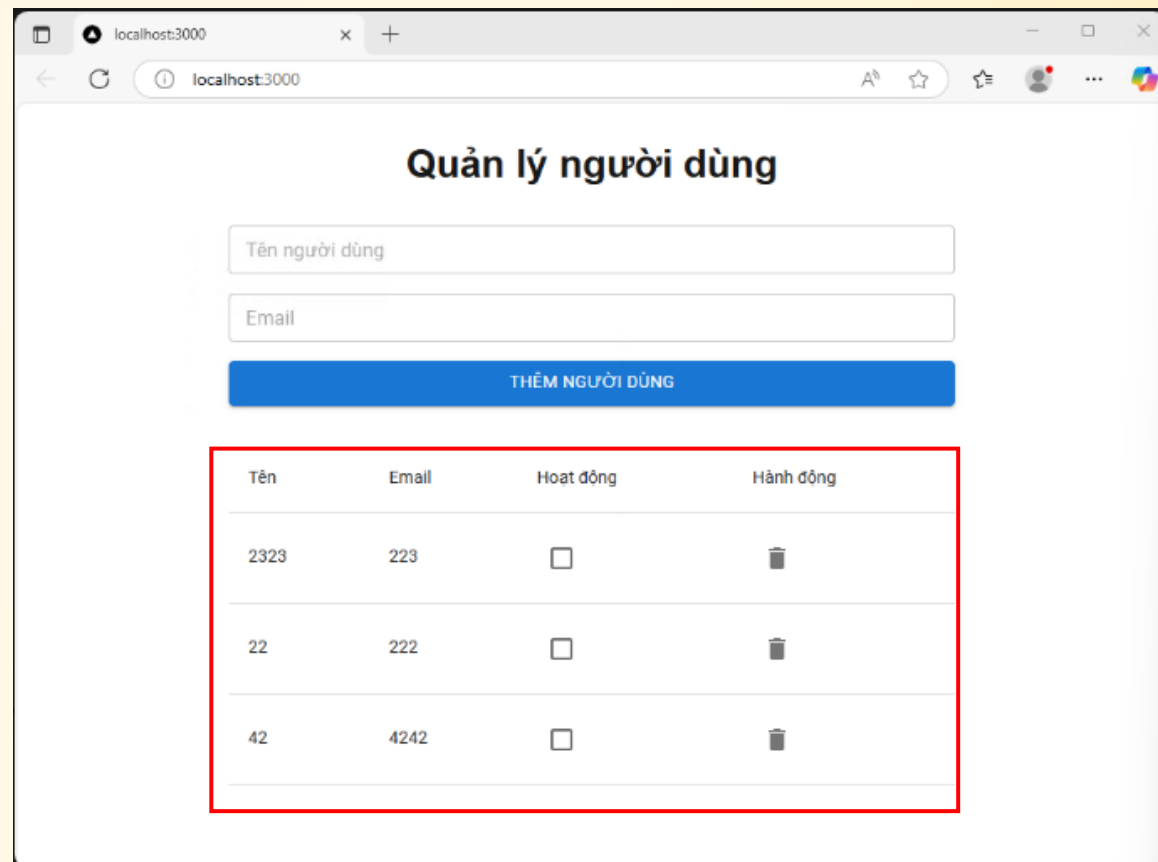
143 packages are looking for funding
  run `npm fund` for details

1 low severity vulnerability
To address all issues, run:
  npm audit fix
Run `npm audit` for details.

C:\workspace_aots\user-frontend>
```

# UserList

- Tạo components/UserList.tsx:
  - Chúng ta sẽ thiết kế component để hiển thị (vẽ) ra grid như hình bên



# UserList

- Tạo components/UserList.tsx:

Định nghĩa Kiểu dữ liệu cho Props (User.tsx)

```
export interface User {
  id?: number;
  name?: string;
  email?: string;
  active?: boolean;
}
```

- React.FC (hoặc React.FunctionComponent) là một kiểu dữ liệu tích hợp sẵn của React cho Functional Components.
- <UserProps> trong dấu ngoặc nhọn là generic type, cho TypeScript biết props của Component này sẽ tuân theo interface User.

## Định nghĩa Functional Component

```
import { Box, Table, TableBody, TableCell, TableContainer, TableHead, TableRow, Checkbox, IconButton } from '@mui/material';
import DeleteIcon from '@mui/icons-material/Delete';

interface UserListProps {
  users: User[];
  onDelete: (id: number | undefined) => void;
  onToggle: (id: number | undefined) => void;
}

const UserList: React.FC<UserListProps> = ({ users, onDelete, onToggle }) => {
  return (
    <Box sx={{ maxWidth: 600, mx: 'auto', mt: 4 }}>
      <TableContainer>
        <Table>
          <TableHead>
            <TableRow>
              <TableCell>Tên</TableCell>
              <TableCell>Email</TableCell>
              <TableCell>Hoạt động</TableCell>
              <TableCell>Hành động</TableCell>
            </TableRow>
          </TableHead>
          <TableBody>
            {users.map((user) => (
              <TableRow key={user.id}>
                <TableCell>{user.name}</TableCell>
                <TableCell>{user.email}</TableCell>
                <TableCell>
                  <Checkbox checked={user.active} onChange={() => onToggle(user.id)} />
                </TableCell>
                <TableCell>
                  <IconButton onClick={() => onDelete(user.id)}>
                    <DeleteIcon />
                  </IconButton>
                </TableCell>
              </TableRow>
            ))}
          </TableBody>
        </Table>
      </TableContainer>
    </Box>
  );
};

export default UserList;
```

# UserForm

- Tạo components/UserForm.tsx:

## Định nghĩa Functional Component

- React.FC (hoặc React.FunctionComponent) là một kiểu dữ liệu tích hợp sẵn của React cho Functional Components.
- <UserFormProps> trong dấu ngoặc nhọn là generic type, cho TypeScript biết props của Component này sẽ tuân theo interface UserFormProps.

```
import { useState } from 'react';
import { Box, TextField, Button } from '@mui/material';
interface UserFormProps {
  onAdd: (name: string, email: string) => void;
}
const UserForm: React.FC<UserFormProps> = ({ onAdd }) => {
  const [name, setName] = useState('');
  const [email, setEmail] = useState('');
  const handleSubmit = (e: React.FormEvent) => {
    e.preventDefault();
    if (name.trim() && email.trim()) {
      onAdd(name, email);
      setName('');
      setEmail('');
    }
  };
  return (
    <Box sx={{ maxWidth: 600, mx: 'auto', mt: 4 }}>
      <form onSubmit={handleSubmit}>
        <TextField
          fullWidth
          value={name}
          onChange={(e) => setName(e.target.value)}
          placeholder="Tên người dùng"
          variant="outlined"
          size="small"
          sx={{ mb: 2 }}
        />
        <TextField
          fullWidth
          value={email}
          onChange={(e) => setEmail(e.target.value)}
          placeholder="Email"
          variant="outlined"
          size="small"
          sx={{ mb: 2 }}
        />
        <Button type="submit" variant="contained" fullWidth>
          Thêm người dùng
        </Button>
      </form>
    </Box>
  );
};
export default UserForm;
```

Định nghĩa Kiểu dữ liệu cho Props

# Demo với dữ liệu tĩnh

- Cập nhật pages/index.tsx (dữ liệu tĩnh):

Thêm vào dữ liệu tĩnh

Cách sử dụng các Component TypeScript đã tạo

```
import Head from "next/head";
import Image from "next/image";
import { Geist, Geist_Mono } from "next/font/google";
import styles from "@styles/Home.module.css";
import { useState } from "react";
import UserForm from "@components/UserForm";
import UserList from "@components/UserList";

const geistSans = Geist({
  variable: "--font-geist-sans",
  subsets: ["latin"],
});

const geistMono = Geist_Mono({
  variable: "--font-geist-mono",
  subsets: ["latin"],
});

interface User {
  id: number;
  name: string;
  email: string;
  active: boolean;
}

export default function Home() {
  const [users, setUsers] = useState<User[]>([
    { id: 1, name: 'Nguyễn Văn A', email: 'a@example.com', active: true },
    { id: 2, name: 'Trần Thị B', email: 'b@example.com', active: false },
  ]);

  const addUser = (name: string, email: string) => {
    const newUser: User = { id: users.length + 1, name, email, active: true };
    setUsers([...users, newUser]);
  };

  return (
    <div>
      <h1 style={{ textAlign: 'center', marginTop: '2rem' }}>Quản lý người dùng</h1>
      <UserForm onAdd={addUser} />
      <UserList users={users} />
    </div>
  );
}
```

# Thực hành

- **Nhiệm vụ** (40 phút):
  - Chạy npm run dev, kiểm tra giao diện.
  - Thêm người dùng tĩnh, tùy chỉnh MUI (đổi màu button).
- **Demo vòng đời React:**
  - Thêm console.log trong useEffect để kiểm tra mount/update.

# Tổng kết

- **Tổng kết:**

- Hiểu React Hooks (useState, useEffect, useCallback).
- Xây dựng giao diện tĩnh với MUI (UserList, UserForm).
- Đặc điểm React (vòng đời) và Next.js (SSR, routing).

- **Bài tập:**

- Thêm combobox Active (Yes/No) vào UserForm.tsx:
- Sử dụng MUI Select để chọn active: true/false.
- Tùy chỉnh giao diện (đổi màu form).



# Q&A

