TypeScript React Essentials

(Don't Worry! this is optional)

Why TypeScript + React?

Catch Errors Early

Compile-time error detection before running your app

Better IntelliSense

Enhanced code completion and documentation

Safer Refactoring

Confidently change code with type checking

Team Collaboration

Self-documenting code improves team workflow

Setting Up Your Environment

Create a new project

```
npx create-react-app my-app --template typescript
```

Install dependencies

```
npm install @types/react @types/react-dom
```

Configure VS Code (Optional)

Install ESLint and TypeScript extensions

Start development server

```
npm start
```

Project Structure

File Types

- `.tsx` - React components with JSX
- `.ts` - Pure TypeScript files
- `.d.ts` - Type declarations

Folder Organization (recommended but not necessary)

- `/components` - Reusable UI components
- `/hooks` - Custom React hooks
- `/types` - Shared type definitions
- `/utils` - Helper functions

Hooks?

What Are Custom Hooks?

Custom hooks let you create your own hooks by combining built-in React hooks (useState, useEffect, etc.) to encapsulate complex logic that can be shared across multiple components.

Hooks?

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

// Custom hook for managing counter logic
function useCounter(initialValue: number = 0) {
  const [count, setCount] = useState(initialValue);

  const increment = () => setCount(prev => prev + 1);
  const decrement = () => setCount(prev => prev - 1);
  const reset = () => setCount(initialValue);

  return { count, increment, decrement, reset };
}

// Using the custom hook in components
function Counter() {
  const { count, increment, decrement, reset } = useCounter(10);

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={increment}>+</button>
      <button onClick={decrement}>-</button>
      <button onClick={reset}>Reset</button>
    </div>
  );
}
```

Your First TypeScript React Component

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

type GreetingProps = {
  name: string;
};

const Greeting: React.FC<GreetingProps> = ({ name }) => {
  return <h1>Hello, {name}!</h1>;
};

export default Greeting;
```

Your First TypeScript (Modern) React Component

```
// Greeting.tsx
type GreetingProps = {
  name: string;
};

const Greeting = ({ name }: GreetingProps) => {
  return <h1>Hello, {name}!</h1>;
};

export default Greeting;
```

Typing Props - The Foundation

```
interface UserProps {
  name: string;          // required
  age?: number;           // optional
  isActive: boolean;
  role: 'admin' | 'user';  // union type
}
```

Best Practices

- Use `interface` for component props (component input)
- Mark optional props with `?`
- Use specific types (avoid `any`)

Event Handling with TypeScript

```
// Button click event
const handleClick = (e: React.MouseEvent) => {
  console.log('Button clicked', e.currentTarget.name);
};

// Input change event
const handleChange = (e: React.ChangeEvent) => {
  setName(e.target.value);
};
```

React provides typed events for all standard DOM events

useState Hook with TypeScript

Explicit Type

```
const [name, setName] =
useState<string>('');
```

Type Inference

```
const [count, setCount] =
useState(0); // inferred as
number
```

Complex Types

```
const [user, setUser] =
useState<User | null>(null);
```

Component Libraries & Third-Party Types

Install library and its types

`npm install material-ui @types/material-ui`

Use with type checking

TypeScript validates prop usage automatically

Handle missing types

Create custom type declarations in `*.d.ts` files

Popular typed libraries

Material-UI, Ant Design, Chakra UI, Tailwind CSS