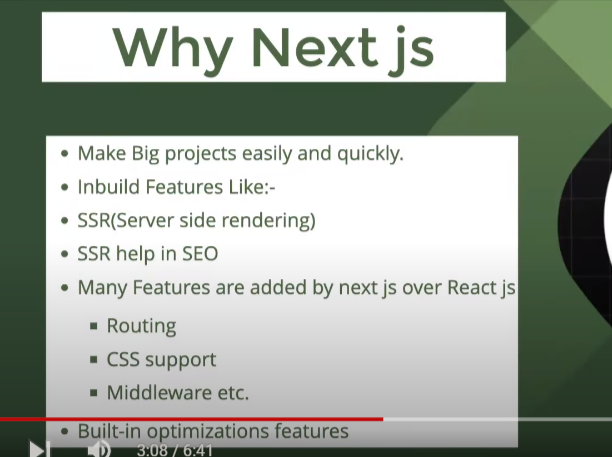
NextJS





What is component ?

Ans – component is pice of code that can be used again and again .

What is diffrence between function and componenet ?

i)componnet name should be start with the capital letter but that is not applicable with the function .

ii)a component is always return and jsx but it it not applicable with the function .

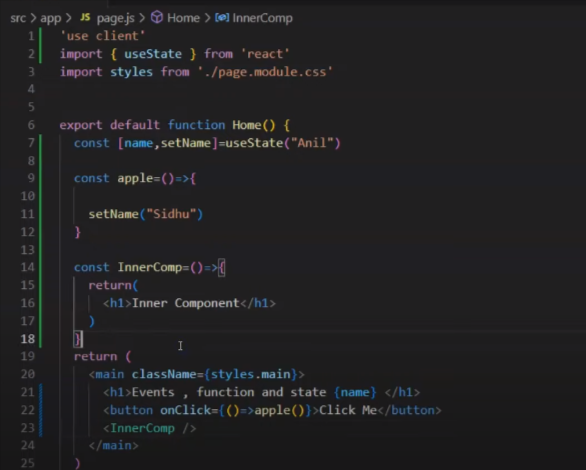
🡺What is use client in nextjs ?

Ans : use component is use to make a complent as a client site rendering .

ii)to use use client we need to put “use client “ at the top of the compnent .

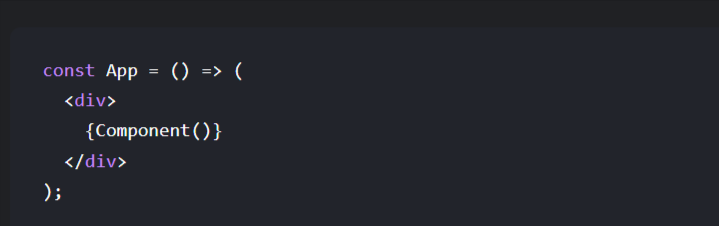
iii)we do not need to put “use client” in the child component of compoent that we want to have client site rendering .

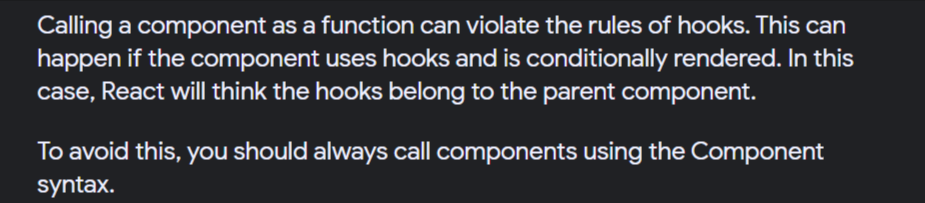
🡺Make a component inside a component



Calling a component as a function and what will be its effect

Ans : use can use comonent as a function





i)React will going to teat as normal javascript function there will be no creation of component instance because react works on the compoenent hrechy

Calling a React component directly as a function instead of using JSX or `React.createElement` can lead to unexpected behavior and issues. Here are the key effects and reasons why it's generally discouraged:

1. \*\*Loss of React's Context\*\*: When you call a component directly as a function, it bypasses React's context system. This means any context provided above this component in the component tree won't be accessible via `useContext` or other context-consuming mechanisms.

2. \*\*Hooks Don't Work\*\*: React relies on the component hierarchy to properly manage hooks' state. If you call a component as a function, hooks like `useState` or `useEffect` inside that component will not work correctly. This is because hooks rely on React's internal mechanisms to keep track of component instances and their states, which are bypassed when directly invoking the component function.

3. \*\*No Lifecycle Methods\*\*: For class components, lifecycle methods (such as `componentDidMount`, `componentDidUpdate`, etc.) will not be called if the component is invoked as a function. This is because lifecycle methods are part of React's class component model, which relies on React creating and managing the component instance.

4. \*\*No Refs\*\*: Refs will not work as expected because React uses refs to reference actual DOM nodes or class component instances. When a component is called as a function, it doesn't create a component instance or DOM node in the same way, so refs cannot be attached.

5. \*\*Performance Implications\*\*: React's reconciliation process relies on being able to track components across renders. Calling a component as a function bypasses this mechanism, potentially leading to inefficient re-rendering and performance issues.

6. \*\*Type Checking and PropTypes\*\*: If you're using `PropTypes` for type checking in development, calling a component as a function will bypass the checks that React performs on props, potentially leading to harder-to-debug issues.

In summary, while technically possible in some cases, calling a React component as a function directly circumvents many of the features and safeguards that React provides, leading to potential bugs, performance issues, and non-idiomatic code. It's recommended to use JSX or `React.createElement` to render components within React applications.