

## Why typescript+react:-

With static type checking you get to learn about potential bugs as you are typing code, then heading to the browser and figuring out at runtime

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

```
npm install --save typescript @types/node @types/react  
@types/react-dom @types/jest
```

You might need to configure TypeScript according to your project's needs. You can do this by creating a `tsconfig.json` file in the root of your project and adding necessary configurations.

```
{  
  "compilerOptions": {  
    "target": "es5",  
    "lib": ["dom", "dom.iterable", "esnext"],  
    "allowJs": true,  
    "skipLibCheck": true,  
    "esModuleInterop": true,  
    "allowSyntheticDefaultImports": true,  
    "strict": true,  
    "forceConsistentCasingInFileNames": true,  
    "module": "esnext",  
    "moduleResolution": "node",  
    "resolveJsonModule": true,  
    "isolatedModules": true,  
    "noEmit": true,  
    "jsx": "react"
```

```
},  
  "include": ["src"]  
}
```

components are defined in .tsx file extension

### **basic prop types:-**

```
type GreetProps = {  
  name: 'string',  
  messageCount: number,  
  isLoggedIn: boolean  
}  
  
Export const Greet = (props: GreetProps) => [  
]
```

Use types when building applications and interfaces when building libraries

```
Const personname = {  
  First: 'bruce',  
  Last: 'wayne'  
}
```

```
type PersonProps = {  
  Name: {  
    First: string,  
    Last: string,
```

```
}
```

```
Const namelist=[
```

```
{first:'bruce,
```

```
Last:'wayne'}
```

```
]
```

```
Type personListsProps={
```

```
Names:{
```

```
First:string,last:string}
```

```
}]
```

Advanced types:-

```
Type statusProps={
```

```
status:"loading"|"success"|"error"
```

```
}
```

---

```
Type oscarprops={
```

```
Children:React.ReactNode
```

```
}
```

---

Optional props

```
Type greetprops={
```

```
Name:string,
```

```
Messagecount?:number,
```

```
Isloggeinin?:Boolean
```

```
}
```

```
Const {messagecount=0}=props
```

Event props:-

Type buttonpros={

Handleclick : ()=>void

}

---

Type buttonpros={

Handleclick :

(event:React.MouseEvent<HTMLButtonElement>,id:number)=>v

oid

}

---

type inputprops={

Value:string,

handleChange :

event:React.ChangeEvent<HTMLInputElement>)=>void

}

Const

handleInputChange=(event:React.ChangeEvent<HTMLInputElem  
ent>)=>{

Console.log(event)

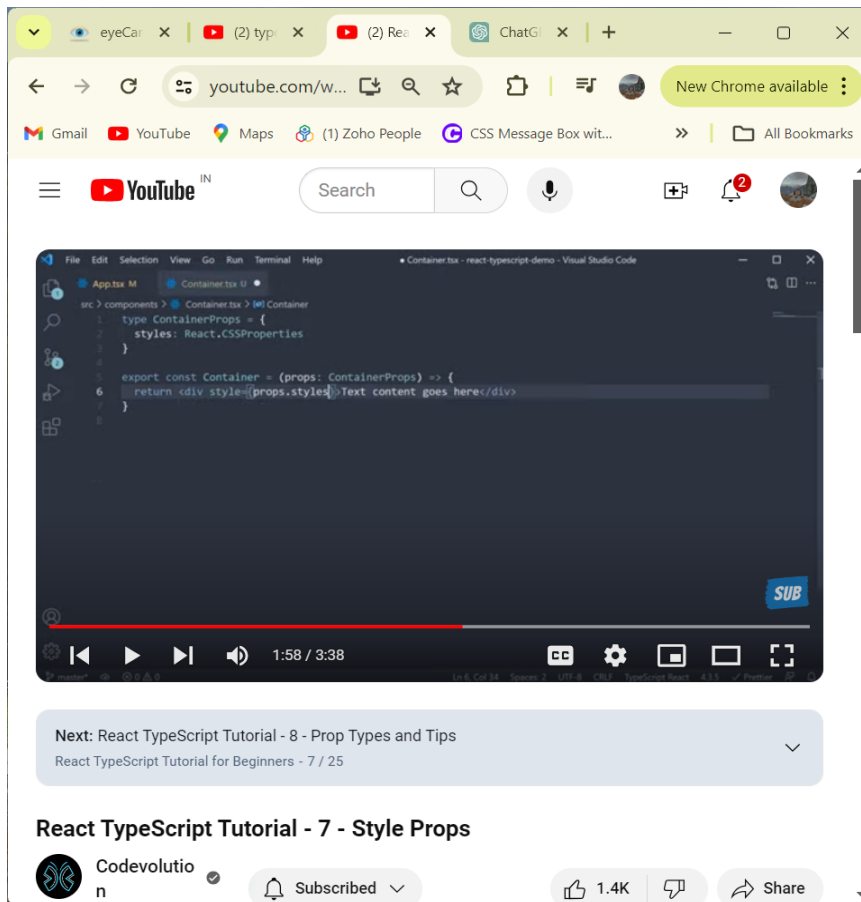
}

## Style Props:-

Type containerprops={

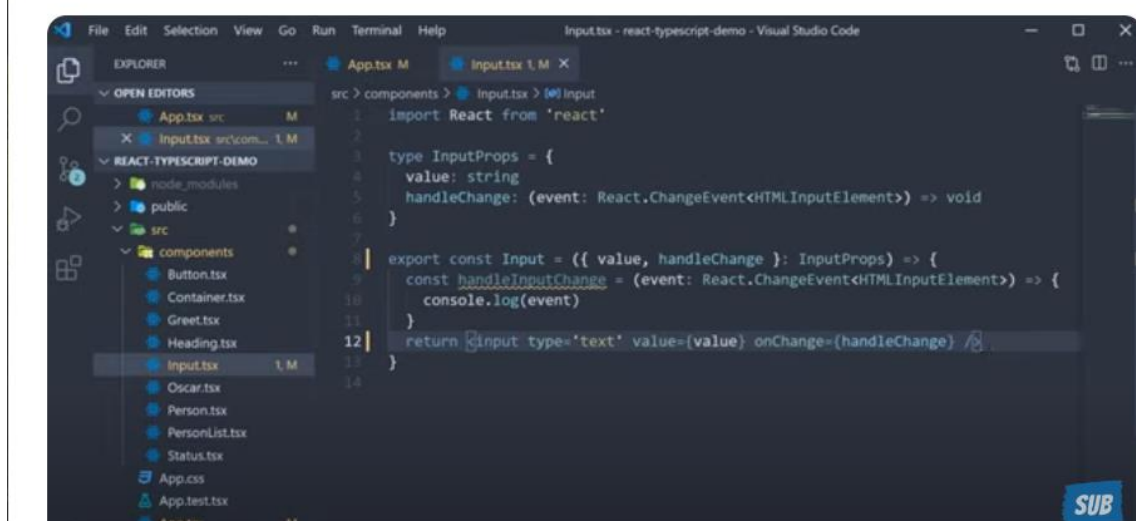
Styles:React.CSSProperties

}

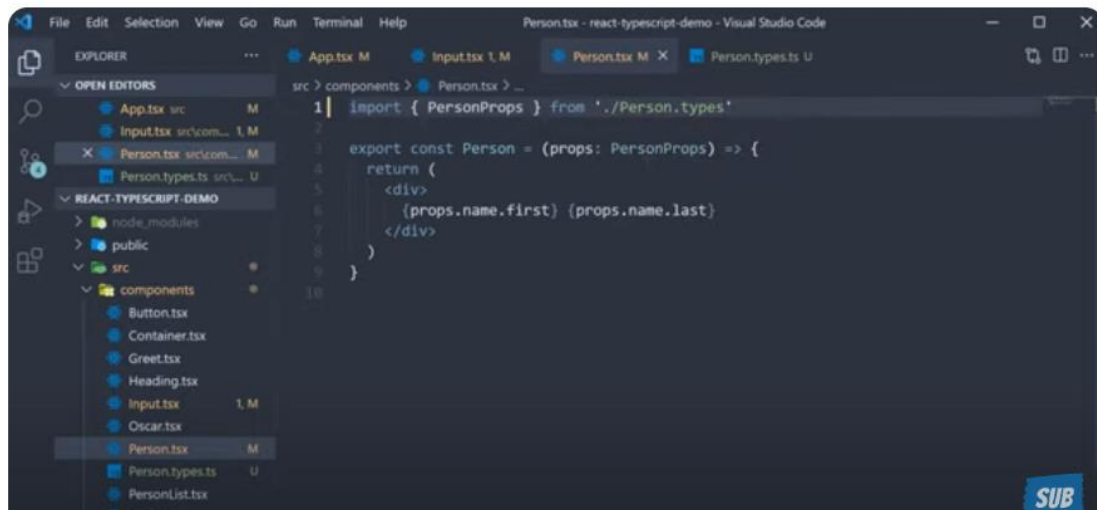
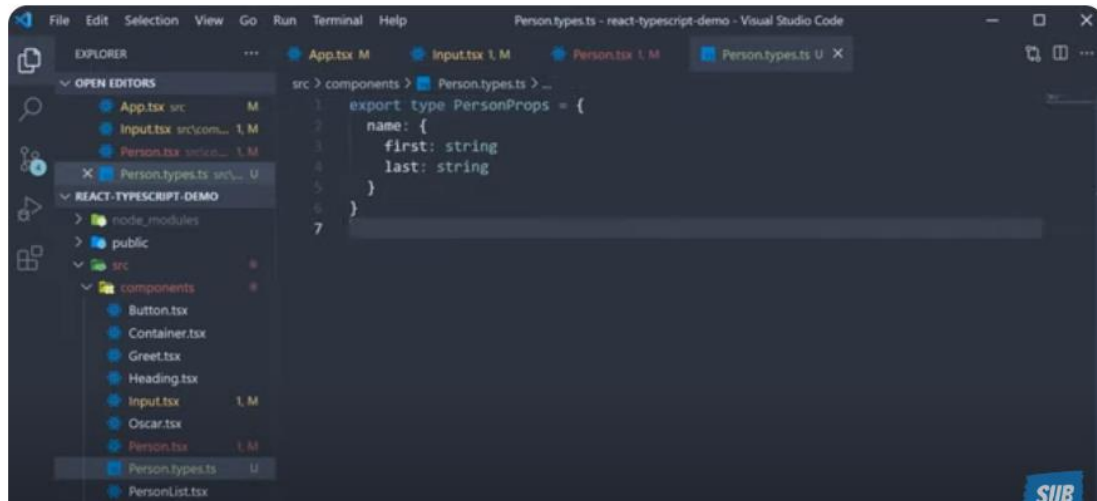


## Prop Types and Tips:-

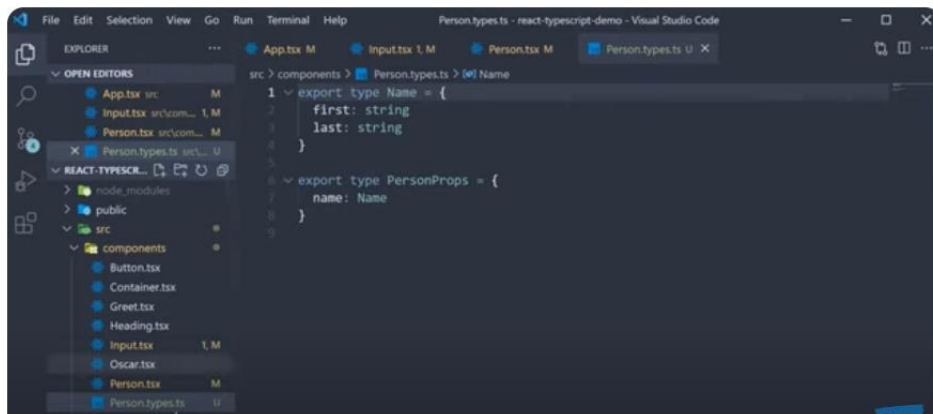
1)destructuring props:-

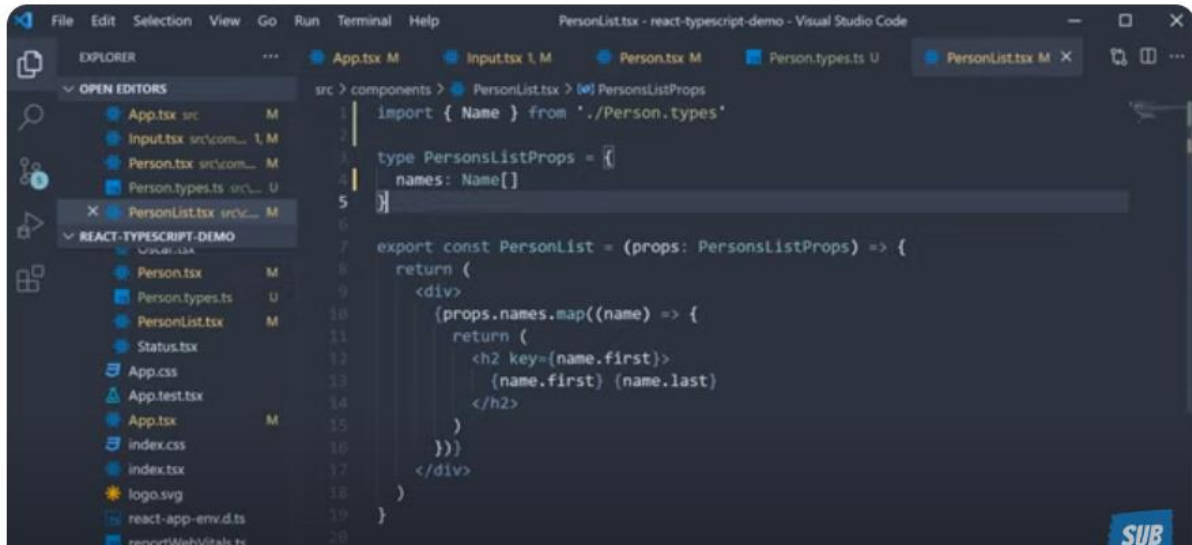


2)can create types to another file



3) it is possible to use a type in multiple places

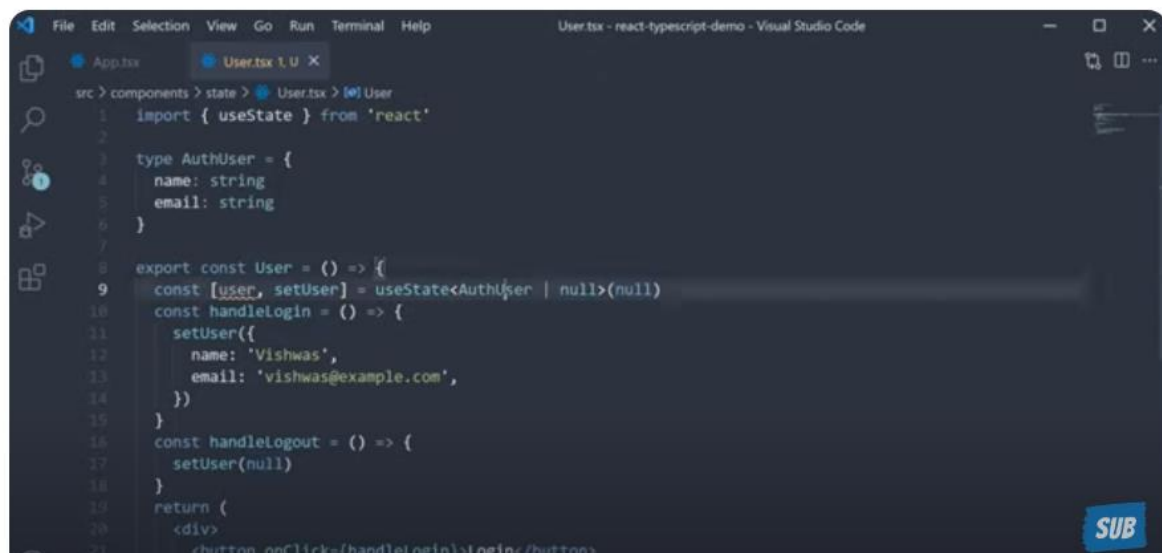




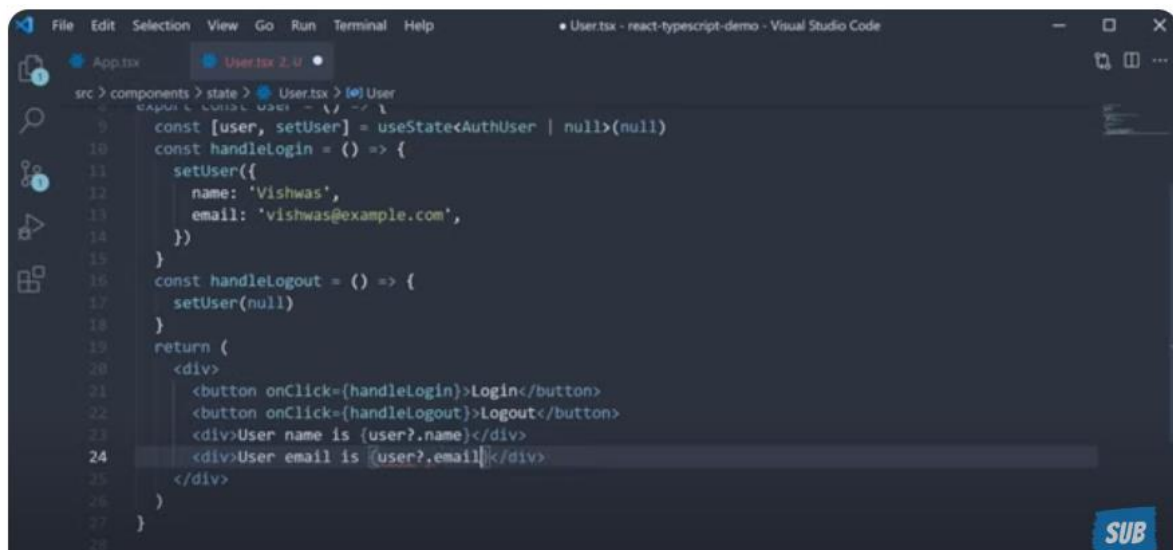
```
1 import { Name } from './Person.types'
2
3 type PersonsListProps = {
4   names: Name[]
5 }
6
7 export const PersonList = (props: PersonsListProps) => {
8   return (
9     <div>
10       {props.names.map((name) => {
11         return (
12           <h2 key={name.first}>
13             {name.first} {name.last}
14           </h2>
15         )
16       })}
17     </div>
18   )
19 }
```

**useState Hook:-**

**useState Future Value:-**



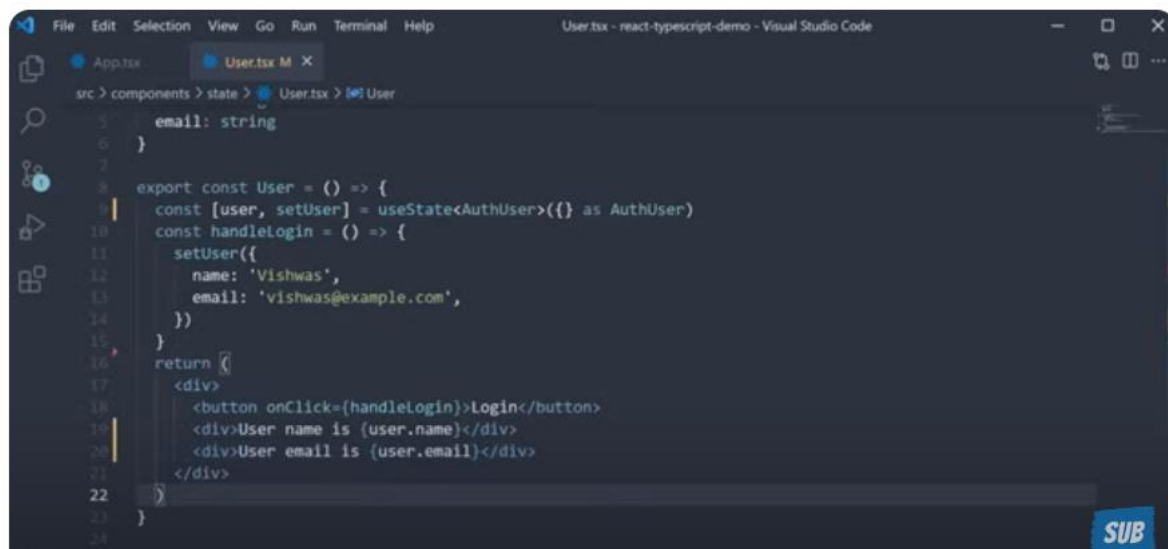
```
1 import { useState } from 'react'
2
3 type AuthUser = {
4   name: string
5   email: string
6 }
7
8 export const User = () => {
9   const [user, setUser] = useState<AuthUser | null>(null)
10   const handleLogin = () => {
11     setUser({
12       name: 'Vishwas',
13       email: 'vishwas@example.com',
14     })
15   }
16   const handleLogout = () => {
17     setUser(null)
18   }
19   return (
20     <div>
21       <button onClick={handleLogin}>Login</button>
```



```
File Edit Selection View Go Run Terminal Help
User.tsx - react-typescript-demo - Visual Studio Code

src > components > state > User.tsx > User
1  import { useState } from 'react'
2
3  const [user, setUser] = useState<AuthUser | null>(null)
4
5  const handleLogin = () => {
6    setUser({
7      name: 'Vishwas',
8      email: 'vishwas@example.com',
9    })
10  }
11
12  const handleLogout = () => {
13    setUser(null)
14  }
15
16  return (
17    <div>
18      <button onClick={handleLogin}>Login</button>
19      <button onClick={handleLogout}>Logout</button>
20      <div>User name is {user?.name}</div>
21      <div>User email is {user?.email}</div>
22    </div>
23  )
24 }
```

## useState Type Assertion:-



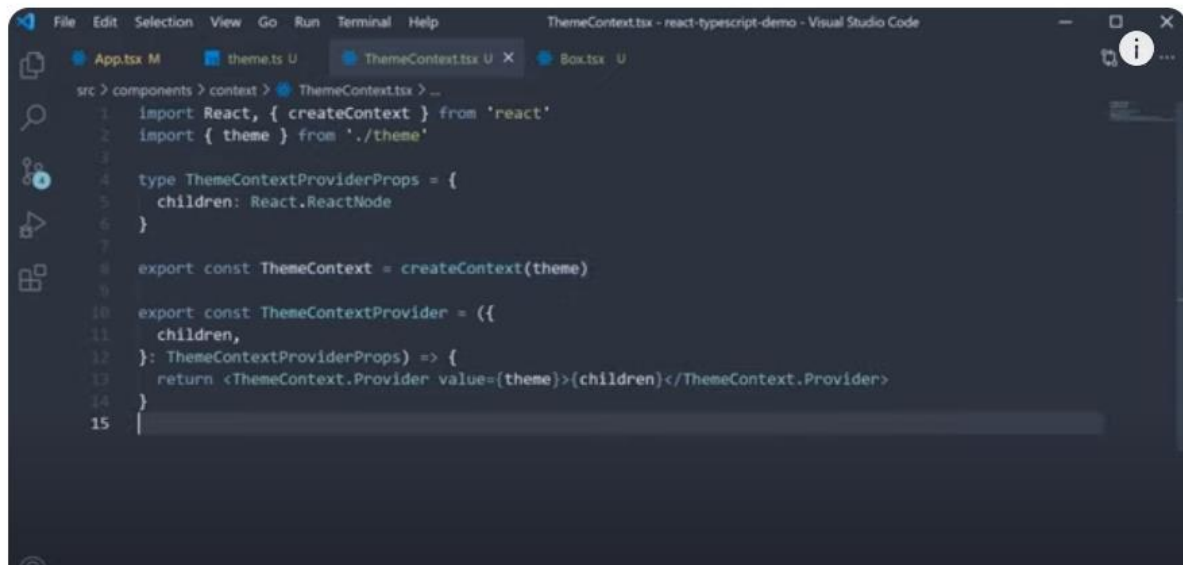
```
File Edit Selection View Go Run Terminal Help
User.tsx - react-typescript-demo - Visual Studio Code

src > components > state > User.tsx > User
1  email: string
2  }
3
4  export const User = () => {
5    const [user, setUser] = useState<AuthUser>({} as AuthUser)
6    const handleLogin = () => {
7      setUser({
8        name: 'Vishwas',
9        email: 'vishwas@example.com',
10      })
11    }
12    return (
13      <div>
14        <button onClick={handleLogin}>Login</button>
15        <div>User name is {user.name}</div>
16        <div>User email is {user.email}</div>
17      </div>
18    )
19  }
20 }
```

## Use Reducer Hook:-

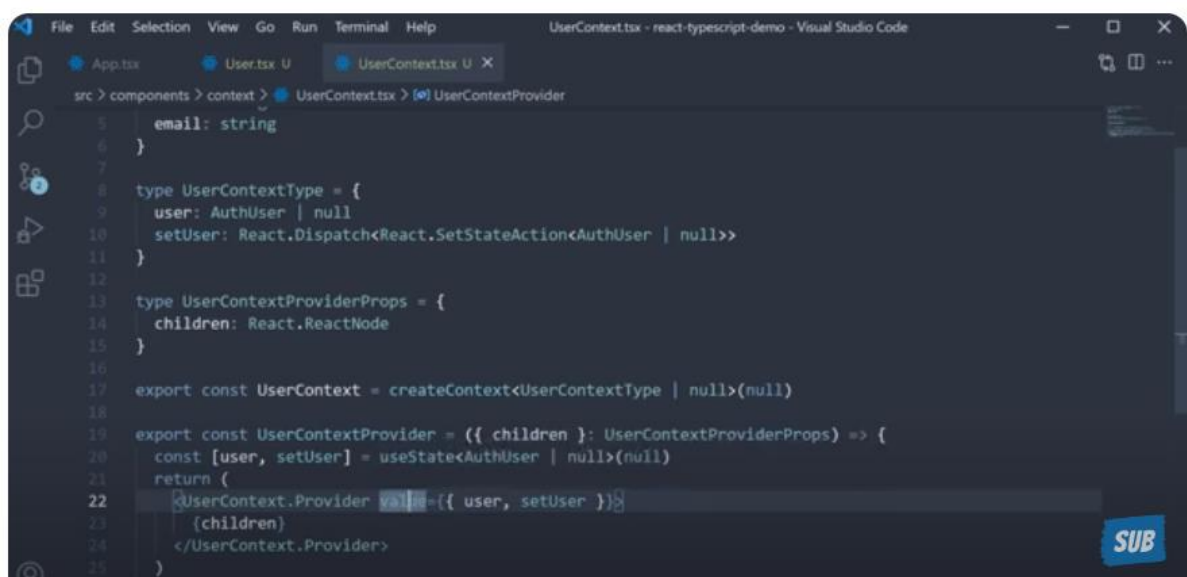
## useContext Hook:-



A screenshot of the Visual Studio Code editor showing a file named ThemeContext.tsx. The file is part of a project structure where the current path is src > components > context > ThemeContext.tsx. The code defines a React context for themes. It imports React and createContext from 'react', and a theme object from './theme'. It defines a ThemeContextProviderProps interface with a children property of type React.ReactNode. Then, it creates a ThemeContext using createContext(theme). Finally, it defines a ThemeContextProvider component that takes children and ThemeContextProviderProps as props and returns a ThemeContext.Provider with the theme value and the children.

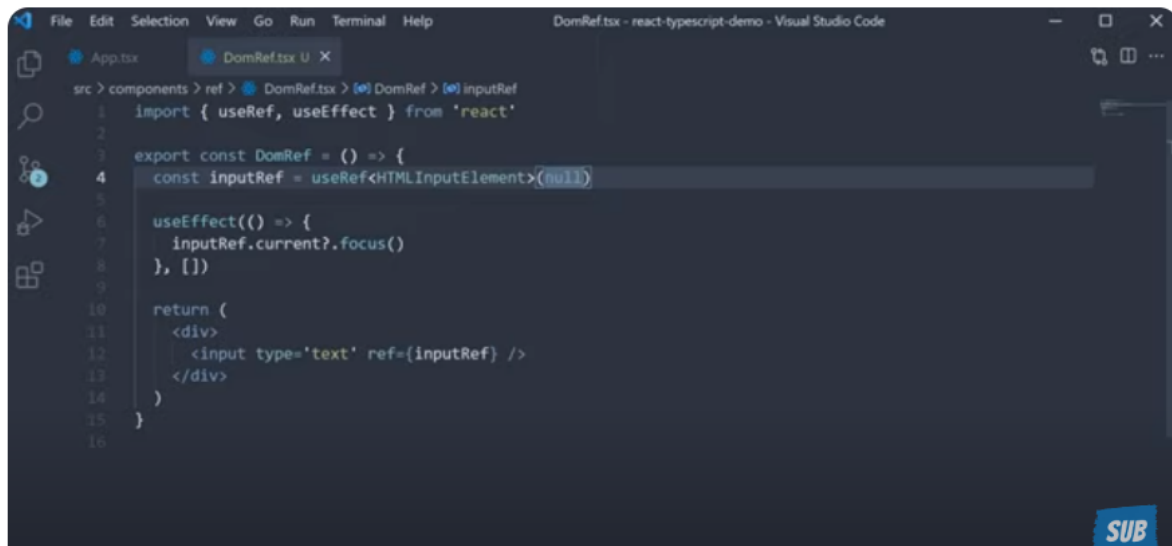
```
1 import React, { createContext } from 'react'
2 import { theme } from './theme'
3
4 type ThemeContextProviderProps = {
5   children: React.ReactNode
6 }
7
8 export const ThemeContext = createContext(theme)
9
10 export const ThemeContextProvider = ({
11   children,
12 }: ThemeContextProviderProps) => {
13   return <ThemeContext.Provider value={theme}>{children}</ThemeContext.Provider>
14 }
15
```

## useContext Future Value:-

A screenshot of the Visual Studio Code editor showing a file named UserContext.tsx. The file is part of a project structure where the current path is src > components > context > UserContext.tsx. The code defines a React context for user authentication. It defines a UserContextType interface with user (AuthUser | null) and setUser (React.Dispatch<React.SetStateAction<AuthUser | null>>) properties. Then, it creates a UserContext using createContext<UserContextType | null>(null). Finally, it defines a UserContextProvider component that takes children and UserContextProviderProps as props. Inside the component, it uses useState to initialize user and setUser. It then returns a UserContext.Provider with the user and setUser values and the children.

```
5   email: string
6 }
7
8 type UserContextType = {
9   user: AuthUser | null
10   setUser: React.Dispatch<React.SetStateAction<AuthUser | null>>
11 }
12
13 type UserContextProviderProps = {
14   children: React.ReactNode
15 }
16
17 export const UserContext = createContext<UserContextType | null>(null)
18
19 export const UserContextProvider = ({ children }: UserContextProviderProps) => {
20   const [user, setUser] = useState<AuthUser | null>(null)
21   return (
22     <UserContext.Provider value={{ user, setUser }}>
23       {children}
24     </UserContext.Provider>
25   )
26 }
```

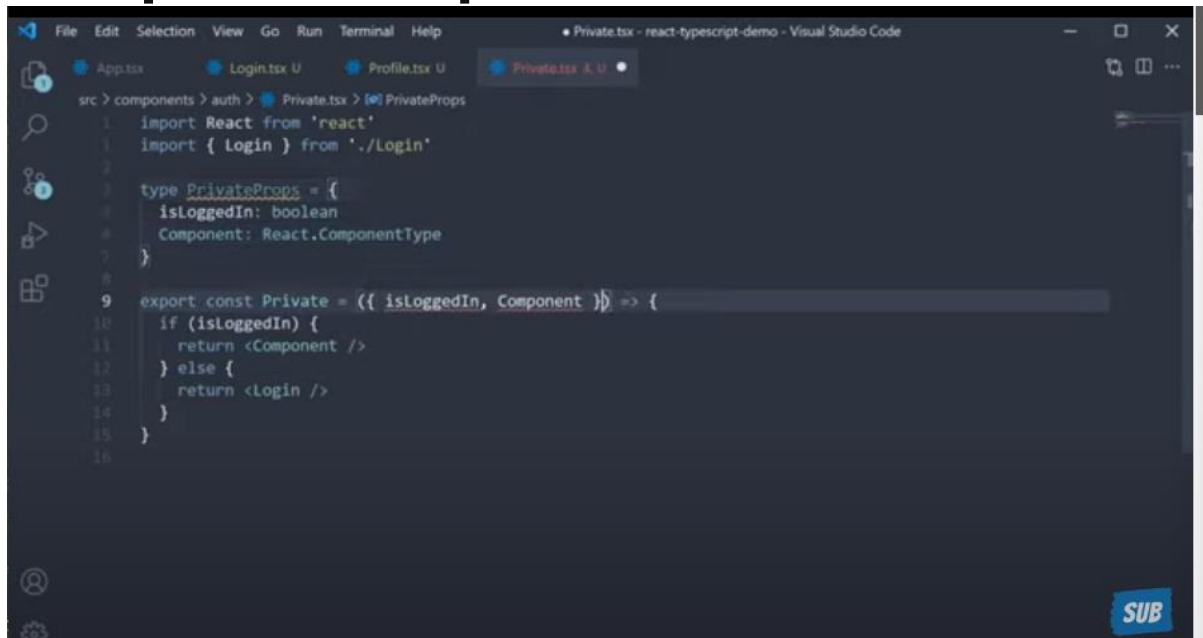
## useRef Hook:-



```
File Edit Selection View Go Run Terminal Help
DomRef.tsx - react-typescript-demo - Visual Studio Code

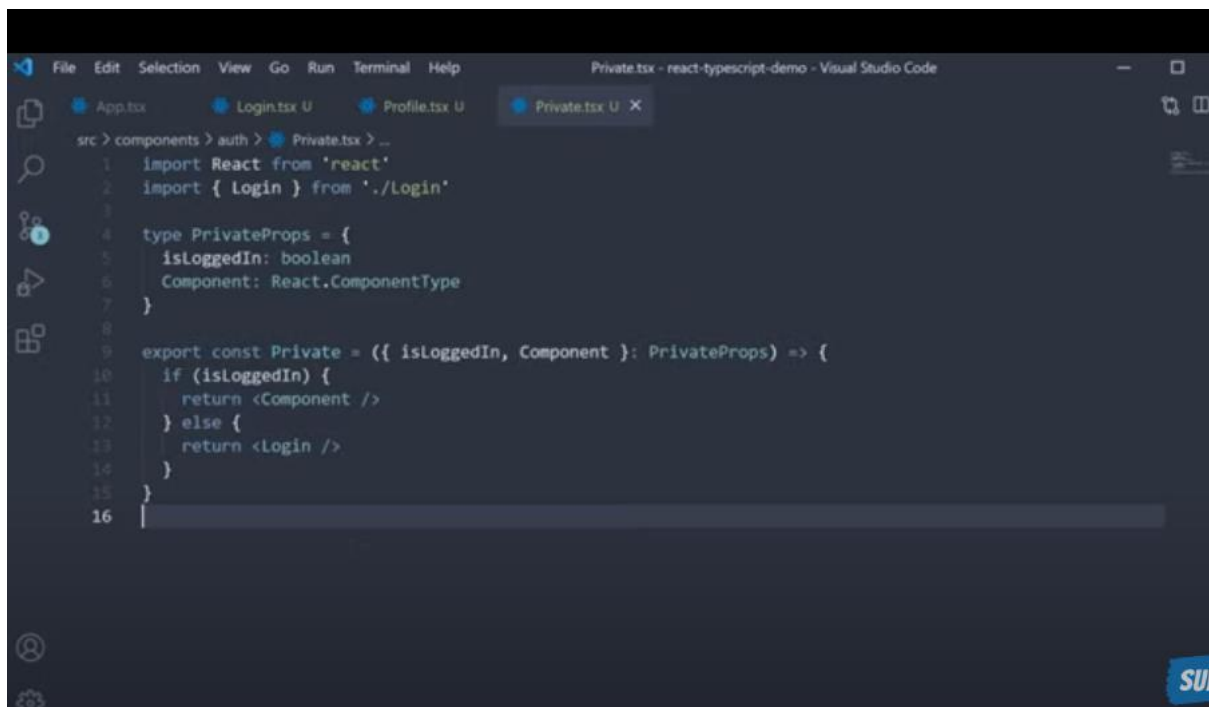
src > components > ref > DomRef.tsx > DomRef > inputRef
1 import { useRef, useEffect } from 'react'
2
3 export const DomRef = () => {
4   const inputRef = useRef<HTMLInputElement>(null)
5
6   useEffect(() => {
7     inputRef.current?.focus()
8   }, [])
9
10  return (
11    <div>
12      <input type='text' ref={inputRef} />
13    </div>
14  )
15 }
16
```

## Component Prop:-

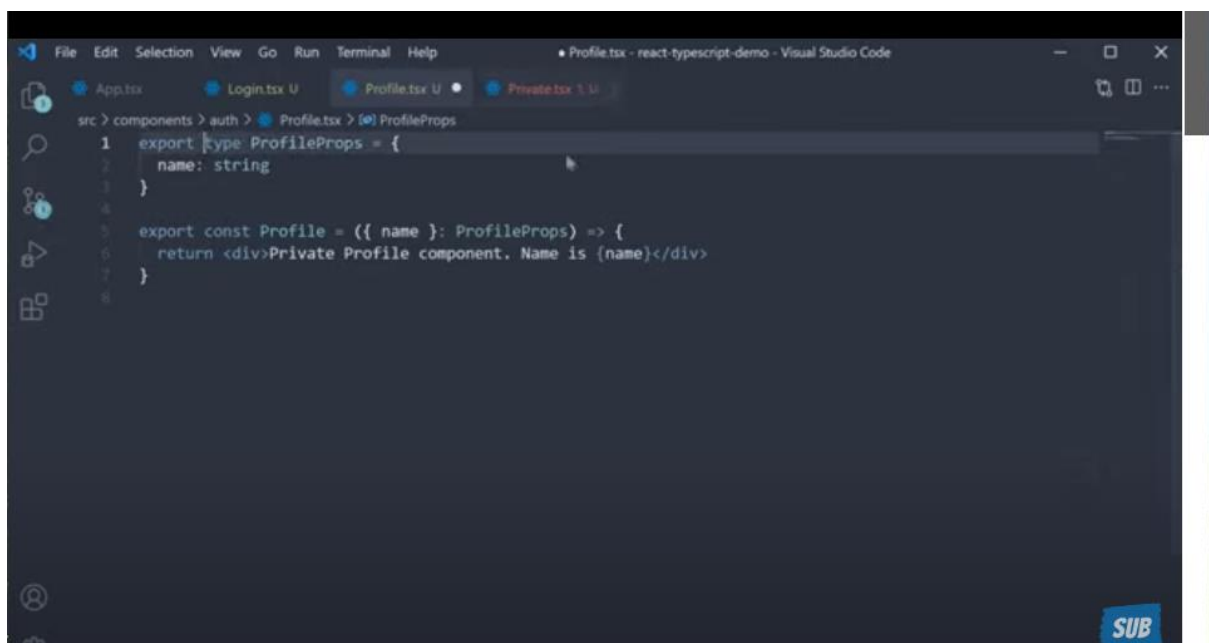


```
File Edit Selection View Go Run Terminal Help
Private.tsx - react-typescript-demo - Visual Studio Code

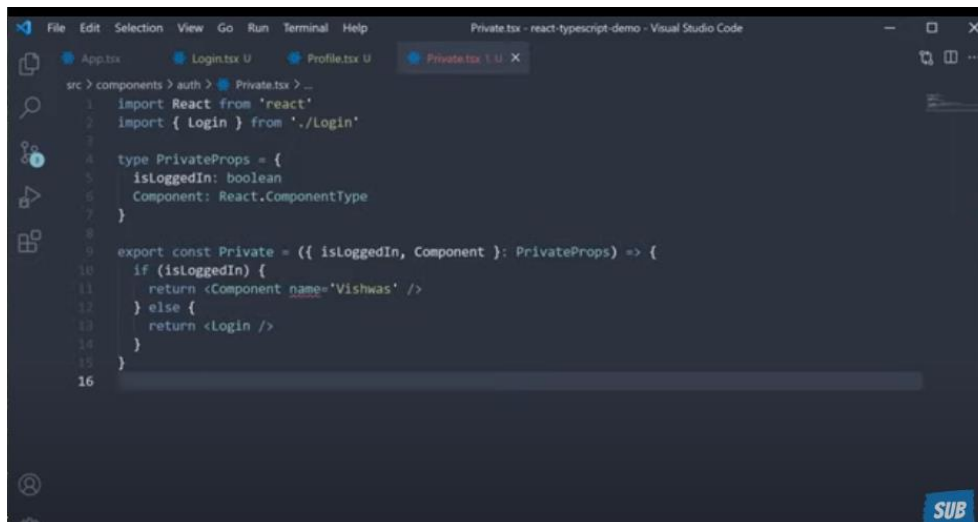
src > components > auth > Private.tsx > PrivateProps
1 import React from 'react'
2 import { Login } from '../Login'
3
4 type PrivateProps = {
5   isLoggedin: boolean
6   Component: React.ComponentType
7 }
8
9 export const Private = ({ isLoggedin, Component }) => {
10   if (isLoggedin) {
11     return <Component />
12   } else {
13     return <Login />
14   }
15 }
16
```



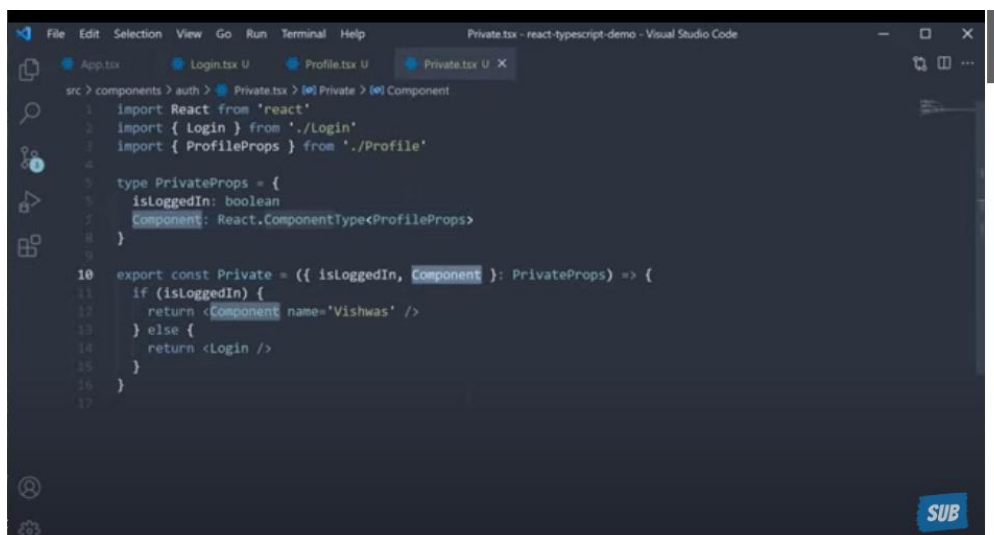
```
File Edit Selection View Go Run Terminal Help Private.tsx - react-typescript-demo - Visual Studio Code
App.tsx Login.tsx U Profile.tsx U Private.tsx U X
src > components > auth > Private.tsx > ...
1 import React from 'react'
2 import { Login } from '../Login'
3
4 type PrivateProps = {
5   isLoggedIn: boolean
6   Component: React.ComponentType
7 }
8
9 export const Private = ({ isLoggedIn, Component }: PrivateProps) => {
10   if (isLoggedIn) {
11     return <Component />
12   } else {
13     return <Login />
14   }
15 }
16
```



```
File Edit Selection View Go Run Terminal Help Profile.tsx - react-typescript-demo - Visual Studio Code
App.tsx Login.tsx U Profile.tsx U Private.tsx U
src > components > auth > Profile.tsx > ProfileProps
1 export type ProfileProps = {
2   name: string
3 }
4
5 export const Profile = ({ name }: ProfileProps) => {
6   return <div>Private Profile component. Name is {name}</div>
7 }
8
```

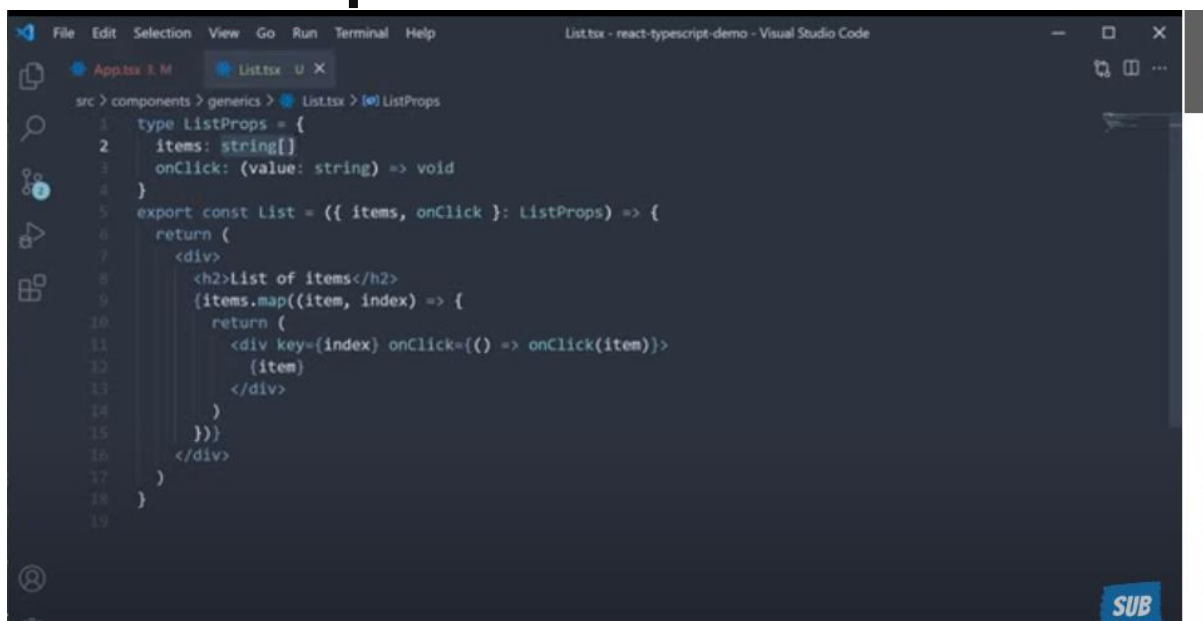


```
File Edit Selection View Go Run Terminal Help Private.tsx - react-typescript-demo - Visual Studio Code
src > components > auth > Private.tsx > ...
1 import React from 'react'
2 import { Login } from './Login'
3
4 type PrivateProps = {
5   isLoggedIn: boolean
6   Component: React.ComponentType
7 }
8
9 export const Private = ({ isLoggedIn, Component }: PrivateProps) => {
10   if (isLoggedIn) {
11     return <Component name='Vishwas' />
12   } else {
13     return <Login />
14   }
15 }
16
```



```
File Edit Selection View Go Run Terminal Help Private.tsx - react-typescript-demo - Visual Studio Code
src > components > auth > Private.tsx > [0] Private > [0] Component
1 import React from 'react'
2 import { Login } from './Login'
3 import { ProfileProps } from './Profile'
4
5 type PrivateProps = {
6   isLoggedIn: boolean
7   Component: React.ComponentType<ProfileProps>
8 }
9
10 export const Private = ({ isLoggedIn, Component }: PrivateProps) => {
11   if (isLoggedIn) {
12     return <Component name='Vishwas' />
13   } else {
14     return <Login />
15   }
16 }
17
```

## Generic Props:-



```
File Edit Selection View Go Run Terminal Help List.tsx - react-typescript-demo - Visual Studio Code
src > components > generics > List.tsx > [0] ListProps
1 type ListProps = {
2   items: string[]
3   onClick: (value: string) => void
4 }
5 export const List = ({ items, onClick }: ListProps) => {
6   return (
7     <div>
8       <h2>List of items</h2>
9       {items.map((item, index) => {
10         return (
11           <div key={index} onClick={() => onClick(item)}>
12             {item}
13           </div>
14         )
15       })}
16     </div>
17   )
18 }
19
```

```
File Edit Selection View Go Run Terminal Help
• List.tsx - react-typescript-demo - Visual Studio Code

App.tsx M List.tsx U
src > components > generics > List.tsx > ListProps
1 type ListProps = {
2   items: string[] | number[]
3   onClick: (value: string | number) => void
4 }
5 export const List = ({ items, onClick }: ListProps) => {
6   return (
7     <div>
8       <h2>List of items</h2>
9       {items.map((item, index) => {
10         return (
11           <div key={index} onClick={() => onClick(item)}>
12             {item}
13           </div>
14         )
15       })}
16     </div>
17   )
18 }
19
```

```
File Edit Selection View Go Run Terminal Help
App.tsx - react-typescript-demo - Visual Studio Code

App.tsx 1 M X List.tsx U X
src > App.tsx > App
9   onClick={() => console.log(item)}
10  </>
11  <List items={[1, 2, 3]} onClick={() => console.log(item)} />
12  <list
13    items={[
14      {
15        first: 'Bruce',
16        last: 'Wayne',
17      },
18      {
19        first: 'Clark',
20        last: 'Kent',
21      },
22      {
23        first: 'Princess',
24        last: 'Diana',
25      },
26    ]}
27    onClick={() => console.log(item)}
28  </list>
29 </div>
30 )
31 }
32
```

The screenshot shows a Visual Studio Code editor window with the file 'List.tsx' open. The code defines a generic component 'List' that takes 'items' and 'onClick' as props. The 'onClick' prop is typed as '(value: T) => void'. The component returns a JSX element with a heading and a list of items, each with a click handler that calls 'onClick(item)'.

```
1 type ListProps<T> = {
2   items: T[]
3   onClick: (value: T) => void
4 }
5 export const List = <T extends {}>({ items, onClick }: ListProps<T>) => {
6   return (
7     <div>
8       <h2>List of items</h2>
9       {items.map((item, index) => {
10         return (
11           <div key={index} onClick={() => onClick(item)}>
12             {item}
13           </div>
14         )
15       })}
16     </div>
17   )
18 }
19
```

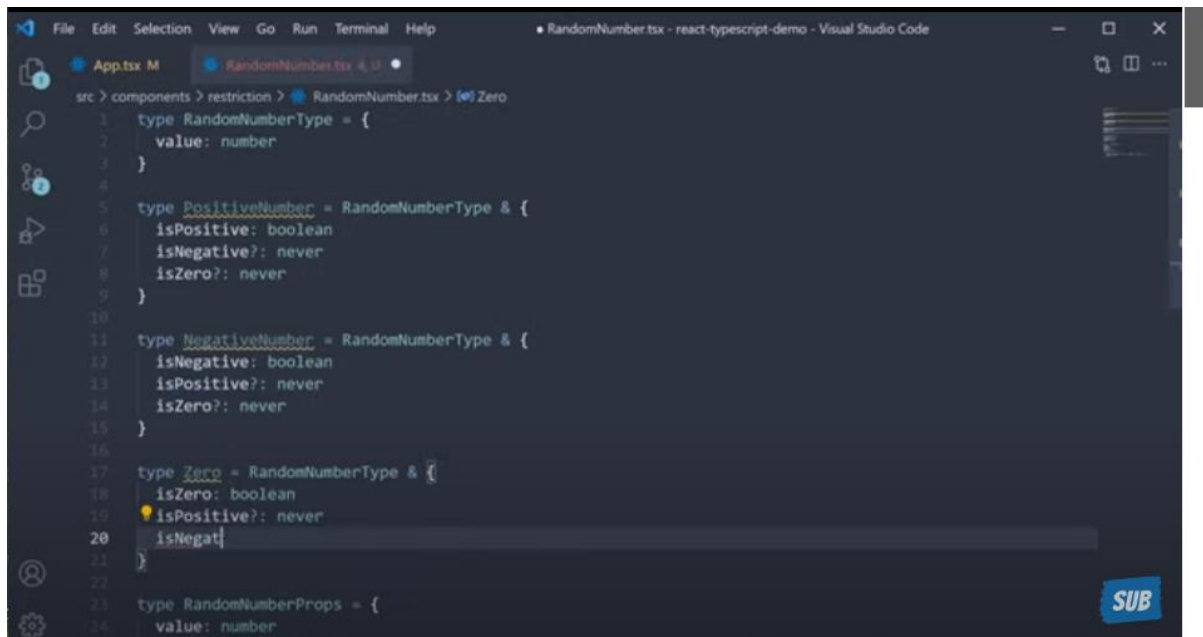
The screenshot shows the same Visual Studio Code editor window, but the 'List' component is now restricted to accept only 'string' or 'number' types. A hover tooltip is visible over the 'number' type, showing a list of related TypeScript types.

```
5 export const List = <T extends string | number>({ items, onClick }: ListProps<T>) => {
6   return (
7     <div>
8       <h2>List of items</h2>
9       {items.map((item, index) => {
10         return (
11           <div key={index} onClick={() => onClick(item)}>
12             {item}
13           </div>
14         )
15       })}
16     </div>
17   )
18 }
19
```

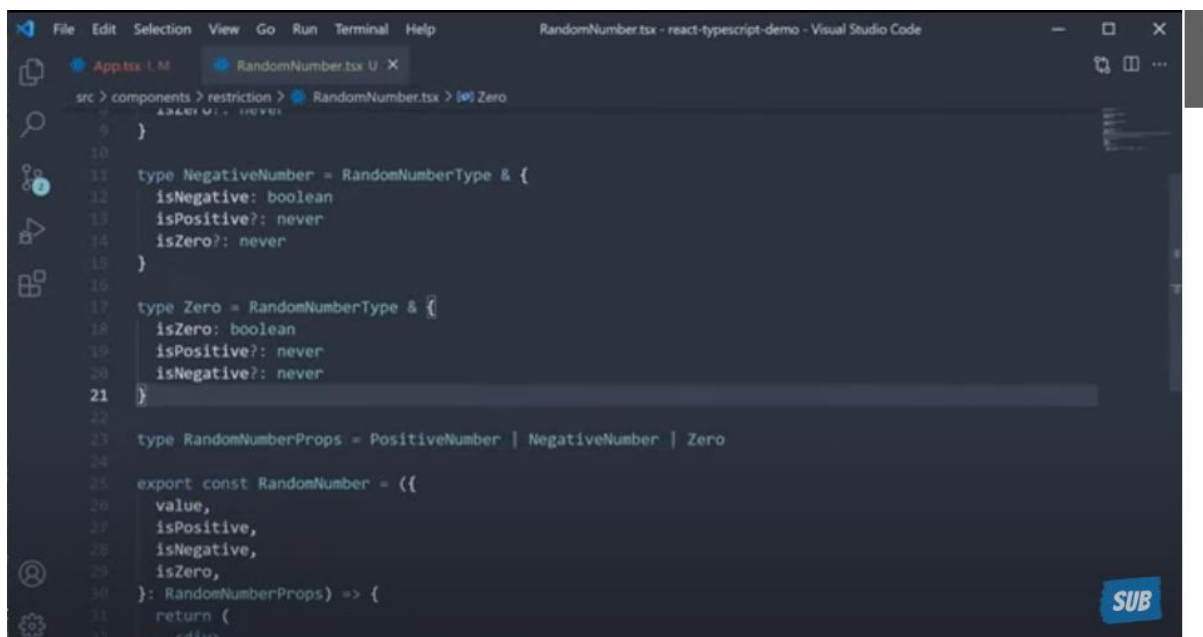
Hover tooltip for 'number':

- number
- Number
- NumberConstructor
- NumberLiteralType
- SVGAnimatedNumber
- SVGAnimatedNumberList
- NamedTupleMember
- NoSubstitutionTemplateLiteral

**Restricting Props:-**

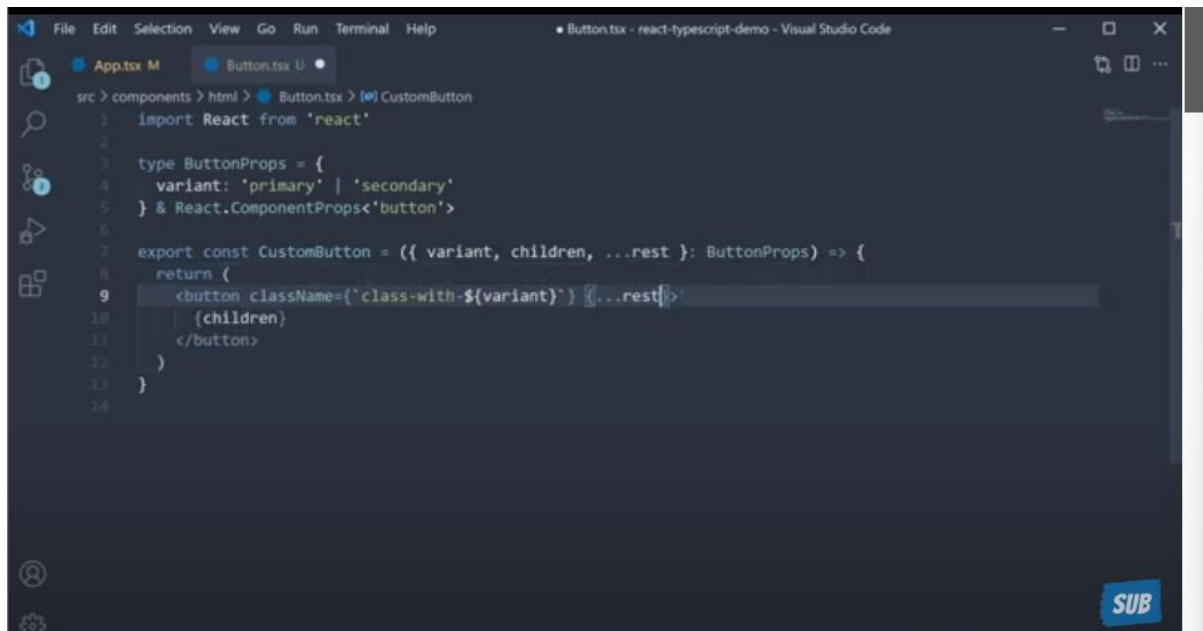


```
1 type RandomNumberType = {
2   value: number
3 }
4
5 type PositiveNumber = RandomNumberType & {
6   isPositive: boolean
7   isNegative?: never
8   isZero?: never
9 }
10
11 type NegativeNumber = RandomNumberType & {
12   isNegative: boolean
13   isPositive?: never
14   isZero?: never
15 }
16
17 type Zero = RandomNumberType & {
18   isZero: boolean
19   isPositive?: never
20   isNegative?: never
21 }
22
23 type RandomNumberProps = {
24   value: number
```

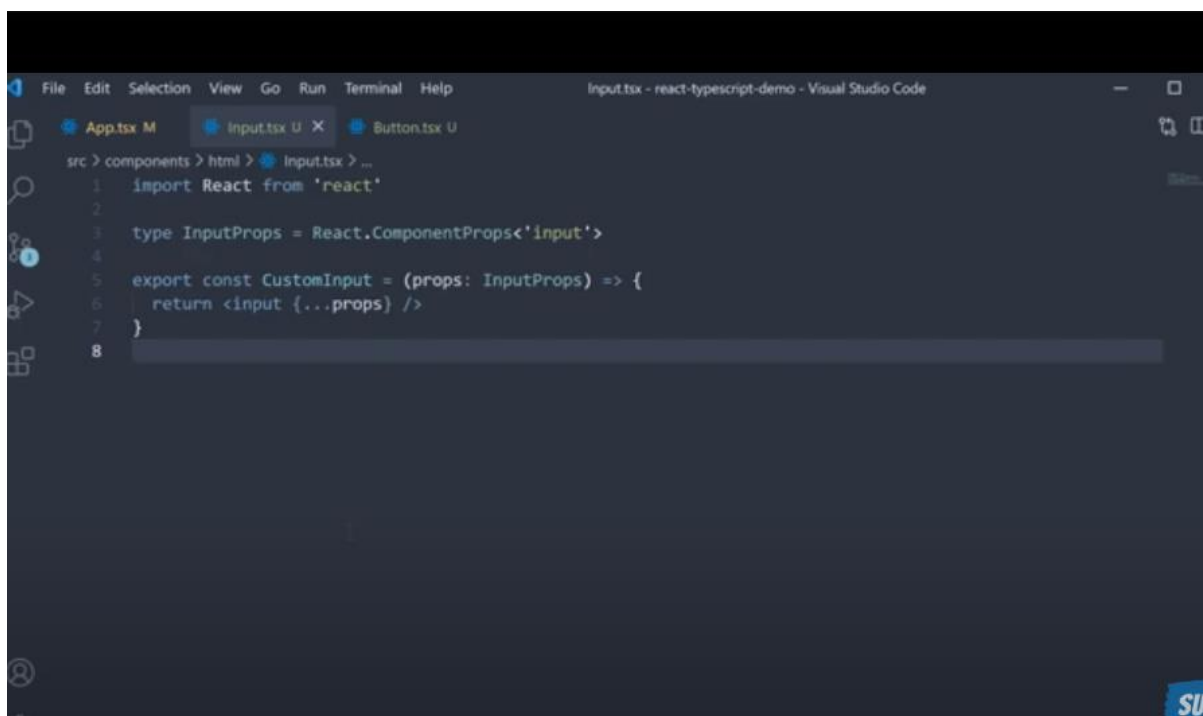


```
1 type RandomNumberType = {
2   value: number
3 }
4
5 type PositiveNumber = RandomNumberType & {
6   isPositive: boolean
7   isNegative?: never
8   isZero?: never
9 }
10
11 type NegativeNumber = RandomNumberType & {
12   isNegative: boolean
13   isPositive?: never
14   isZero?: never
15 }
16
17 type Zero = RandomNumberType & {
18   isZero: boolean
19   isPositive?: never
20   isNegative?: never
21 }
22
23 type RandomNumberProps = PositiveNumber | NegativeNumber | Zero
24
25 export const RandomNumber = ({
26   value,
27   isPositive,
28   isNegative,
29   isZero,
30 }: RandomNumberProps) => {
31   return (
32     <div>
```

## Wrapping HTML Elements:-

A screenshot of the Visual Studio Code editor showing a file named Button.tsx. The file is located in the path src > components > html > Button.tsx. The code defines a ButtonProps type with a variant property of type 'primary' | 'secondary'. It then defines a CustomButton component that takes variant, children, and ...rest as props. The component returns a button element with a className that includes the variant and the rest of the props.

```
1 import React from 'react'
2
3 type ButtonProps = {
4   variant: 'primary' | 'secondary'
5 } & React.ComponentProps<'button'>
6
7 export const CustomButton = ({ variant, children, ...rest }: ButtonProps) => {
8   return (
9     <button className={`class-with-${variant}`} {...rest}>
10       {children}
11     </button>
12   )
13 }
```

A screenshot of the Visual Studio Code editor showing a file named Input.tsx. The file is located in the path src > components > html > Input.tsx. The code defines an InputProps type as React.ComponentProps<'input'>. It then defines a CustomInput component that takes props of type InputProps and returns an input element with the props.

```
1 import React from 'react'
2
3 type InputProps = React.ComponentProps<'input'>
4
5 export const CustomInput = (props: InputProps) => {
6   return <input {...props} />
7 }
8
```

**Extracting a Components Prop Types:-**



