

## Typescript in React

Watch video to get basic understanding: <https://www.youtube.com/watch?v=ydkQlJhodio>

My takeaways:

Can use typescript template when running `npx create-react-app`

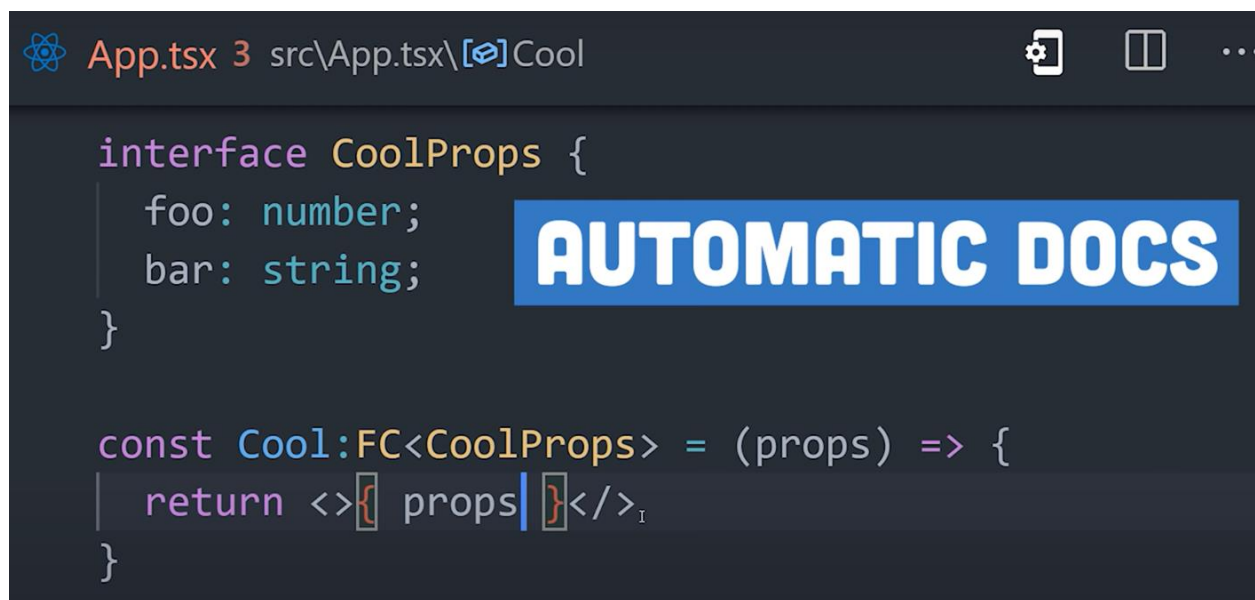
Can detect bad code, but can't detect bad logic! (so still need tests)

In react:

Use FC for Function Component (it later turned out this isn't needed anymore)

Give props an expected structure

Add `?` before `:` to `foo` and `bar` to make them optional



```
App.tsx 3 src\App.tsx\Cool

interface CoolProps {
  foo: number;
  bar: string;
}

const Cool: FC<CoolProps> = (props) => {
  return <>{ props }</>
}
```

Researching `useState` and props syntax using <https://github.com/typescript-cheatsheets/react>

`useState`:

Type inference works well when setting simple state, providing you set it to the correct type in the first place!

If initialising with `null`, then you need to be explicit, use a union type and use angular brackets

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

## useState

Type inference works very well for simple values:

```
const [state, setState] = useState(false);  
// `state` is inferred to be a boolean  
// `setState` only takes booleans
```

See also the [Using Inferred Types](#) section if you need to use a complex type that you've relied on inference for.

However, many hooks are initialized with null-ish default values, and you may wonder how to provide types. Explicitly declare the type, and use a union type:

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

You can also use type assertions if a state is initialized soon after setup and always has a value after:

```
const [user, setUser] = useState<User>({} as User);  
  
// later...  
setUser(newUser);
```

This temporarily "lies" to the TypeScript compiler that `{}` is of type `User`. You should follow up by setting the `user` state — if you don't, the rest of your code may rely on the fact that `user` is of type `User` and that may lead to runtime errors.

## Props

Define them at the top of the file for the component receiving the props – but this works cross file and will work wherever that component is called to ensure you are passing the correct props too!

### Arshi's takeaway


Google for generics in TS

Typescript doesn't just ensure the state is of the correct type, it checks that whatever you pass to the Set function to update state is the correct type too!

Can de-structure props and still use type annotation:

TS App.tsx M X

src > TS App.tsx > Heading

```
3
4  function App() {
5      const [state, setState] = useState(false);
6
7      return <Heading title="ABC" />;
8  }
9
10 type HeadingProps = {
11     title: string;
12 };
13  (parameter) title: string
14 function Heading({ title }: HeadingProps) {
15     return <h2>{title}</h2>;
16 }
17
18 export default App;
```