## **React with TypeScript**

https://github.com/peerberger/TypeScriptReact

Demo project: https://github.com/weibenfalk/react-shopping-cart

Run the following commands.
 npm install
 npm start

## What is TypeScript?

- TypeScript is like an extension of JavaScript.
- It adds new features to JS that help building bigger apps. Features like:
  - Strong types so you can't change the type of variables, which means you can catch a lot of bugs without running the app, because the editor will tell you the type is wrong.
- Since TS is just an extension of JS, JS code is valid TS code (but not vice versa).

## Creating a React project with TypeScript

 You can create a React project that uses TS with the normal npm command, you simply need to add the --template typescript flag.
 Ex:

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

## Using TypeScript in React

Let's create a new component named "TextField".
 Right now it looks like a regular js react component.
 Ex:

Remember one of ts features is that it's strongly typed?
 Well, you can now define the component as a Function Component with a type from the react library.

Ex:

• One benefit of doing so, is that you can strongly define the properties the component is supposed to get.

To do that, pass an object with the property names and types in the arrow brackets, as follows.

Ex:

 When you start having more and more properties, you can write it simpler by declaring the object as an **interface**, and then passing it instead.
 Ex:

 What that does is that if you try to insert that component without the 'text' property, as follows, it will yell at you with a compilation error.
 Ex:

 Another perk is that (at least in vs code) you'll get intellisense for the components properties!
 Ex:

```
• App.tsx - TypeScriptReact - Visual Studi...
   File Edit Selection View Go Run Terminal
                                                                                                th 🗆 ...
                              TS App.tsx 1, M • TS TextField.tsx U
      ∨ TYPESCRIPTREACT
                               my-app > src > TS App.tsx > [❷] App
                                 1 ∨ import { FC } from "react";

✓ my-app

                                     import TextField from "./Components/TextField";
         > public
                                 4 \vee const App : FC = () => {
                               6
                                                <TextField />
                                            ₿ text
                                                                        (JSX attribute) Props.text: string
品
                                           ) ℘ children?
         TS index.tsx
                                            ₿ key?
         TS react-app-env.d.ts
                                            ☐ #endregion
         TS reportWebVitals.ts
                                      expor  #region
        .gitignore
                                            _____hoc
                                                                                            hocComponent
        {} package-lock.json
                                            hocComponentWithRedux
        {} package.json
                                            ☐_rafc
☐_rafce

 README.md

                                                                        reactArrowFunctionExportComponent
                                            _____rafcp
                                                                  _reactArrowFunctionComponentWithPropTypes
        stsconfig.json
                                            ___rfc
                                            ___rfce
```

The other types you can have are as follows.
 Ex:

```
interface Props {
    text: string;
    ok: boolean;
    i: number;
    foo: (name: string) => string; // the second 'string' is the
return type
    obj: {
        f1: string
    };
    personObj: Person;
}

interface Person {
    name: string;
    age: number;
}
```

You can also determine that a property is optional using ?.
 Ex:

```
interface Props {
    text: string;
    ok?: boolean;
    i?: number;
    foo?: (name: string) => string; // the second 'string' is the
return type
    obj?: {
        f1: string
    };
    personObj?: Person;
}
```

Now let's see what you can do with react hooks and ts.
 First of all, when you define a useState hook for example, the type is inferred from the initial value.

Ex:

```
TextField.tsx - TypeScriptReact - Visual Stud...
                                                                                          ti 🗆 ...
 EXPLORER ... TS App.tsx M
14 intertace Person {
15 name: string;
16 age: nu count
  > node_modules
                                     age: nu 'count' is declared but its value is never
  > public
                                            read. ts(6133)
   ✓ src

18

✓ Components

19

const TextF

const count: number

Quick Fix... (Ctrl+.)

TS App.tsx

M

21

const [count, setCount] = us
   TS App.tsx M
TS index.tsx M
                                    const [count, setCount] = useState(5);
  TS App.tsx
                                     return (
   TS reportWebVitals.ts 25
  gitignore
  {} package-lock.json
  {} package.json
  ③ README.md
                                export default TextField;
  stsconfig.json
```

 But if you want to make it so the value can be also null, you can explicitly define the type as follows.

In ts the logical OR operator is |.

Ex:

Ex:

```
const [count, setCount] = useState<number | null>(5);
setCount(null);
```

 It's worth mentioning that 'undefined' is a different type than null, so you can't set 'count' to 'undefined', unless of course you declare it can be that too.
 Ex:

```
const [count, setCount] = useState<number | null | undefined>(5);
setCount(undefined);
```

• You can also pass interfaces here.

```
const [person, setPerson] = useState<Person>({name: 'bob', age:
22});
```