Best practices & advanced TypeScript tips for React developers

Maurice de Beijer
@mauricedb

- Maurice de Beijer
- The Problem Solver
- Microsoft MVP
- Freelance developer/instructor
- Currently at https://someday.com/
- Twitter: @mauricedb
- Web: http://www.TheProblemSolver.nl
- E-mail: maurice.de.beijer@gmail.com

Topics

- Writing **React components** using TypeScript
  - Mutually exclusive component props
  - Generic React component prop types
  - Deriving React component prop types
  - Inferring TypeScript types
- TypeScript **stricter** features
  - Beyond strict using noUncheckedIndexedAccess
- **Validating data at the boundary** using Zod
  - Inferring TypeScript types from Zod schemas
- **Type mapping**
  - With Omit<>, Pick<> and Readonly<>
  - Custom type mapping
- **Runtime and compile type safety**
  - Type predicate and assertion functions
  - Exhaustiveness checking

Type it out by hand?

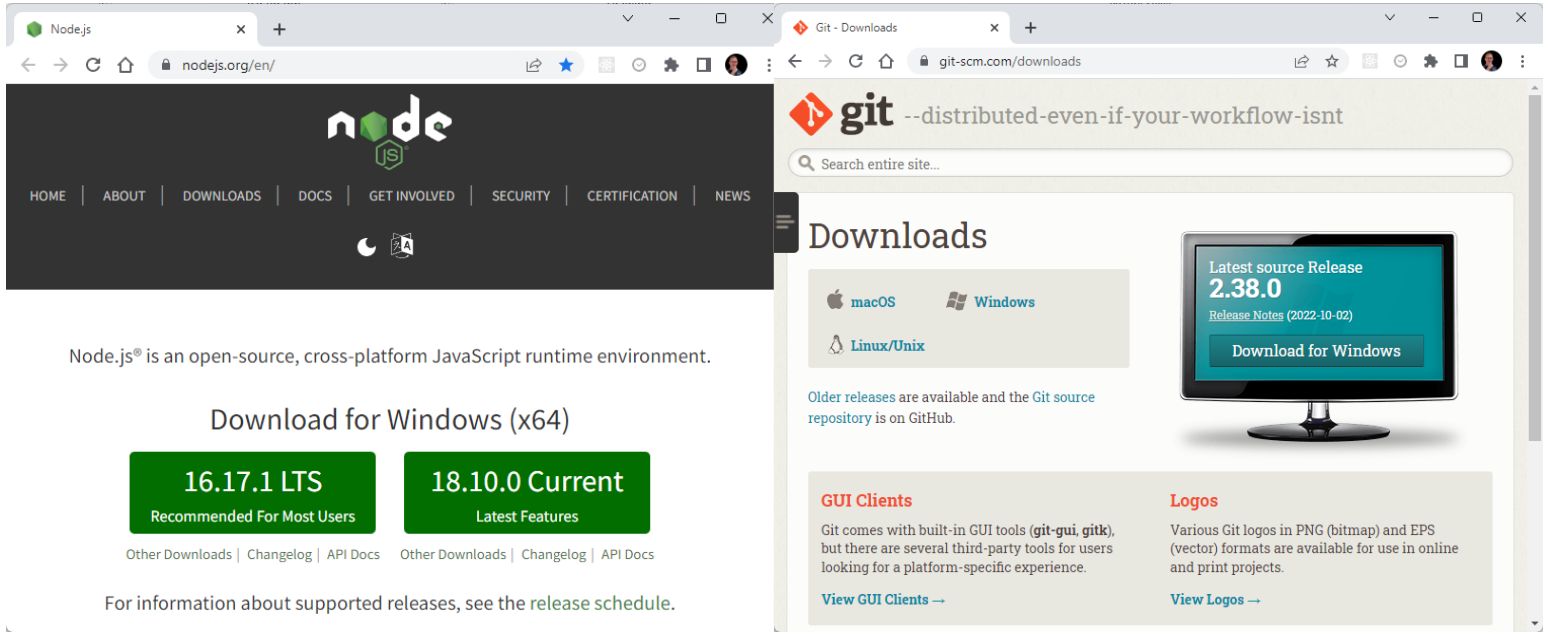
"Typing it drills it into your brain much better than simply copying and pasting it. You're forming new neuron pathways. Those pathways are going to help you in the future. Help them out now!"

Prerequisites

Install Node & NPM

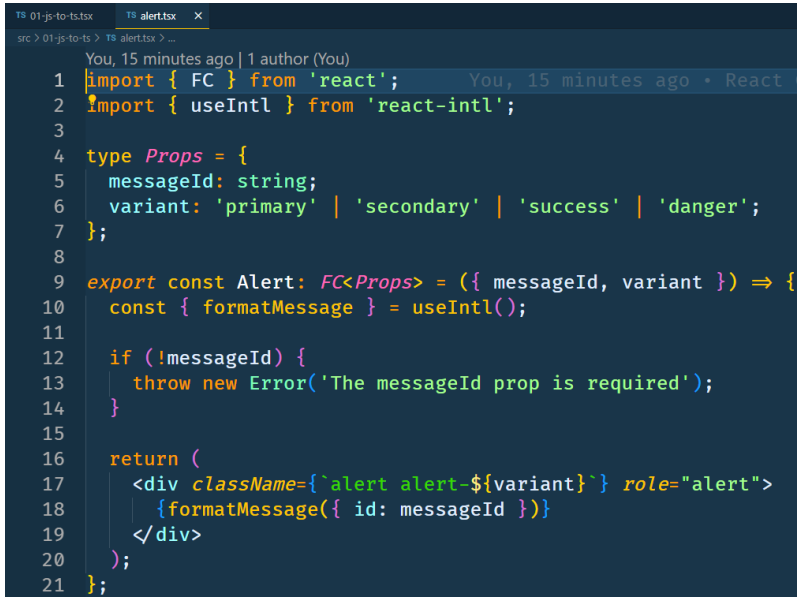
Install the GitHub repository

Install Node.js & NPM




```
PS C:\Repos> node --version
v16.17.1
PS C:\Repos> git --version
git version 2.38.0.windows.1
PS C:\Repos> npm --version
8.19.2
```

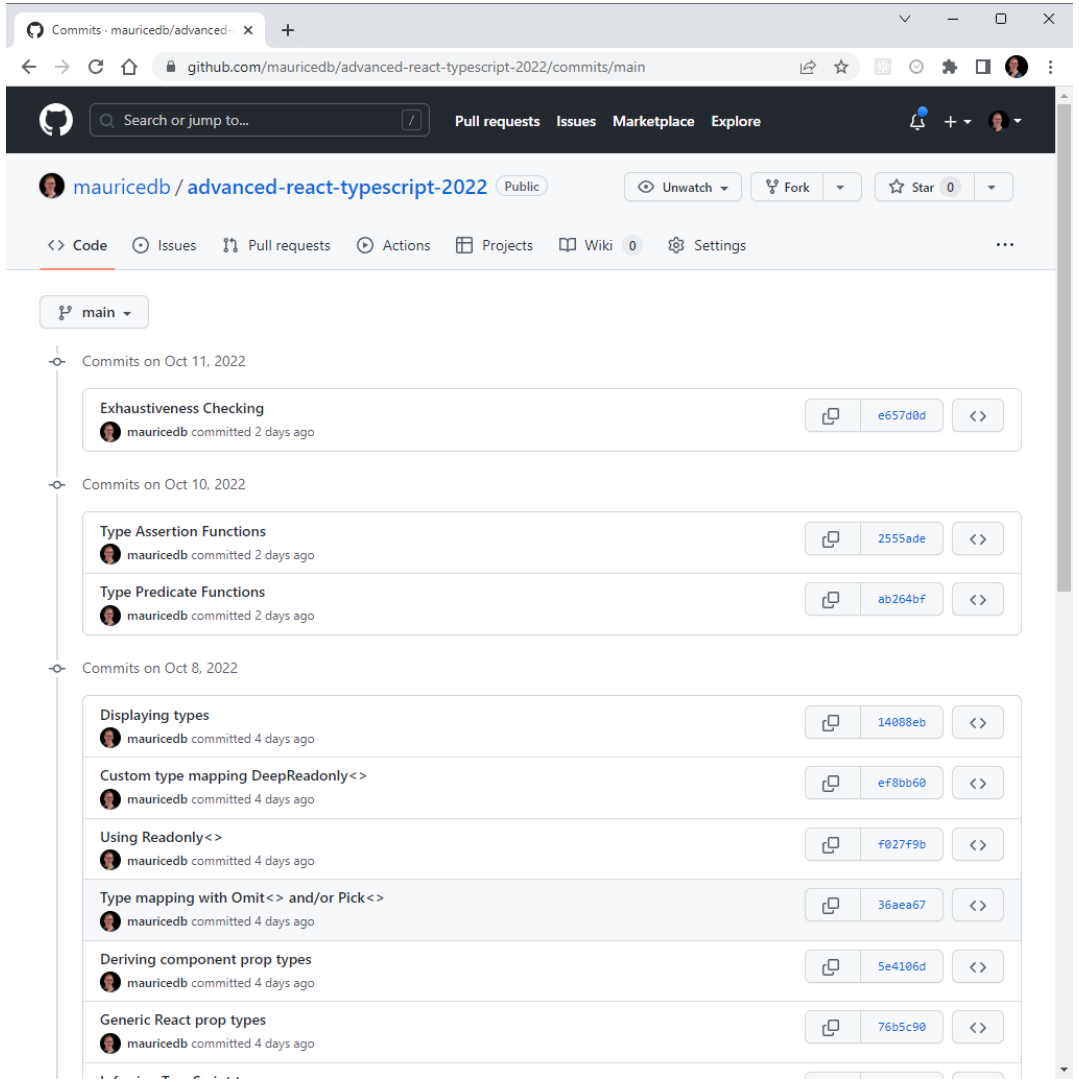
Following Along



- Repo: https://github.com/mauricedb/advanced-react-typescript-2022
- Slides: https://bit.ly/react-ts-2022

The changes


MAKE IT SO
memegenerator.net


Commits · mauricedb/advanced-
github.com/mauricedb/advanced-react-typescript-2022/commits/main
Search or jump to...
Pull requests
Issues
Marketplace
Explore
mauricedb / advanced-react-typescript-2022
Public
Unwatch
Fork
Star 0
Code
Issues
Pull requests
Actions
Projects
Wiki 0
Settings
main
Commits on Oct 11, 2022
Exhaustiveness Checking
mauricedb committed 2 days ago
e657d0d
Commits on Oct 10, 2022
Type Assertion Functions
mauricedb committed 2 days ago
2555ade
Type Predicate Functions
mauricedb committed 2 days ago
ab264bf
Commits on Oct 8, 2022
Displaying types
mauricedb committed 4 days ago
14088eb
Custom type mapping DeepReadonly<>
mauricedb committed 4 days ago
ef8bb60
Using Readonly<>
mauricedb committed 4 days ago
f027f9b
Type mapping with Omit<> and/or Pick<>
mauricedb committed 4 days ago
36aea67
Deriving component prop types
mauricedb committed 4 days ago
5e4106d
Generic React prop types
mauricedb committed 4 days ago
76b5c90

Clone the GitHub Repository

```
PS C:\Temp> git clone git@github.com:mauricedb/advanced-react-typescript-2022.git
Cloning into 'advanced-react-typescript-2022'...
remote: Enumerating objects: 145, done.
remote: Counting objects: 100% (145/145), done.
remote: Compressing objects: 100% (92/92), done.
Receiving objects:  76% (111/145), 9.28 MiB | 4.60 MiB/sremote: Total 145 (delta 53), reused 138 (delta 46), pack-reused 0
Receiving objects: 100% (145/145), 9.82 MiB | 4.65 MiB/s, done.
Resolving deltas: 100% (53/53), done.
PS C:\Temp>
```

Install NPM Packages

```
PS C:\Temp> cd .\advanced-react-typescript-2022\
PS C:\Temp\advanced-react-typescript-2022> npm install

added 251 packages, and audited 252 packages in 5s

81 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS C:\Temp\advanced-react-typescript-2022>
```

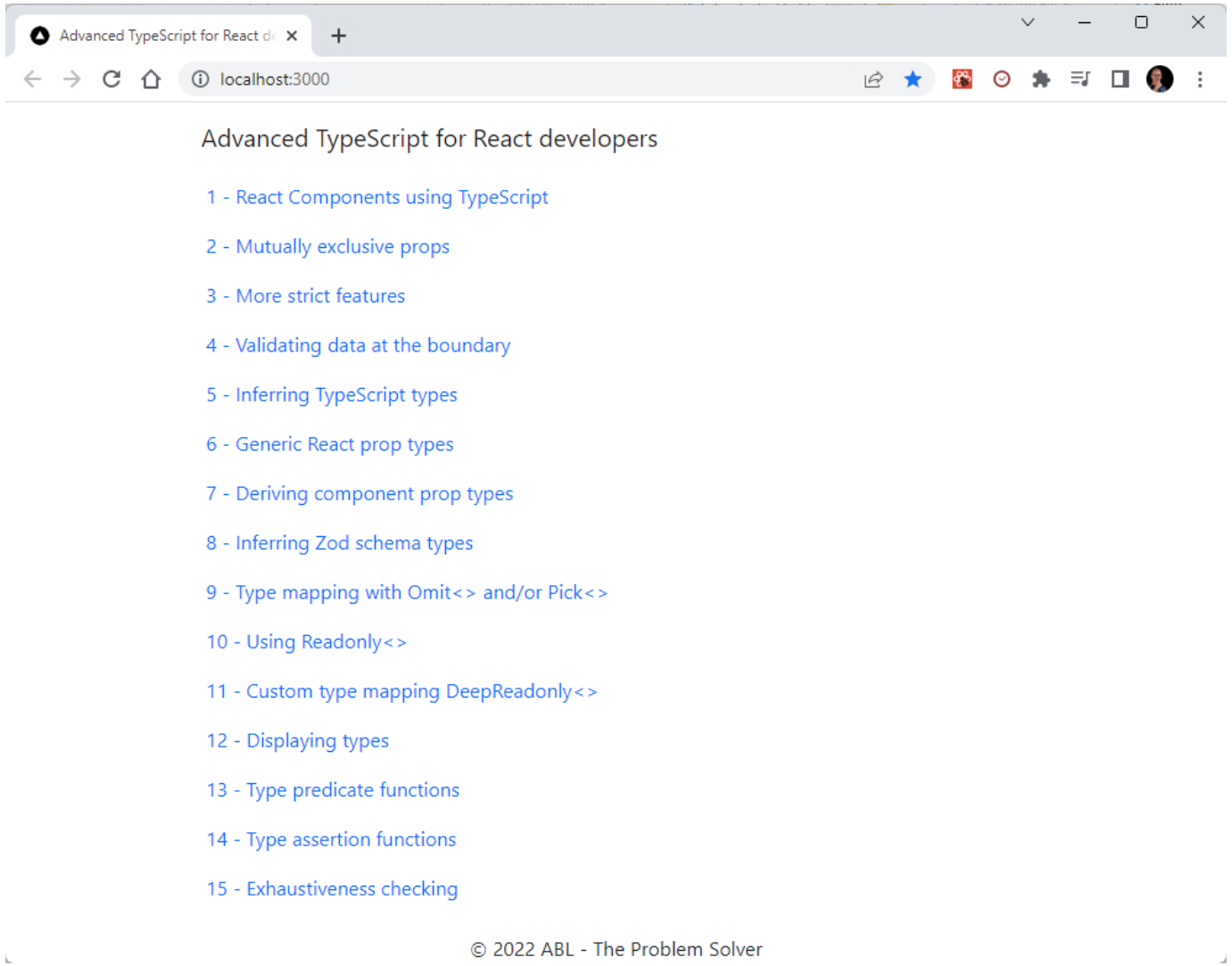
Start branch

- Start with the **00-start** branch
  - `git checkout --track origin/00-start`

Start the application

```
PS C:\Temp\advanced-react-typescript-2022> npm run dev

> advanced-react-typescript-2022@0.1.0 dev
> next dev

ready - started server on 0.0.0.0:3000, url: http://localhost:3000
event - compiled client and server successfully in 543 ms (335 modules)
```

The application



Compiling the code

Compiling the code

- Quite often TypeScript code is not type checked during development
  - Create React App use Babel
  - Next.js uses SWC

package.json

npm run compile


```
TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE  AZURE  GITLENS

[14:12:26] File change detected. Starting incremental compilation...

pages/index.tsx:84:7 - error TS2322: Type 'number' is not assignable to type 'string'.

84 const notString: string = 1;
         ~~~~~~~~~~

[14:12:26] Found 1 error. Watching for file changes.
```

React Components using TypeScript

Components and TypeScript

- React components can be **written in different ways**
  - Named functions or arrow functions
  - Just like with ECMAScript
- Create a **type to describe** the component **Props**
  - Either an interface or a type alias
- Annotate the result as a **valid React type**
  - Or let TypeScript infer the resulting type
- Typing with an arrow function is often easier with **React.FC<TProp>**
  - But doesn't work well with generic components

alert.tsx

TS 01-js-to-ts.tsx | TS alert.tsx ×

src > 01-js-to-ts > TS alert.tsx > ...

You, 15 minutes ago | 1 author (You)

```tsx
import { FC } from 'react';      You, 15 minutes ago • React
import { useIntl } from 'react-intl';

type Props = {
  messageId: string;
  variant: 'primary' | 'secondary' | 'success' | 'danger';
};

export const Alert: FC<Props> = ({ messageId, variant }) => {
  const { formatMessage } = useIntl();

  if (!messageId) {
    throw new Error('The messageId prop is required');
  }

  return (
    <div className={`alert alert-${variant}`} role="alert">
      {formatMessage({ id: messageId })}
    </div>
  );
};
```

Mutually exclusive props

Mutually exclusive props

- Sometimes **not all combinations of props** are allowed
  - Two props might be mutually exclusive
  - You must pass one of them but not both
- Use an ***or*** between **multiple prop types**
  - With an optional "*never*" to prevent illegal combinations

dual-alert.tsx


```
TS 02-mutually-exclusive.tsx 1   TS dual-alert.tsx M ×
src > 02-mutually-exclusive > TS dual-alert.tsx > ...
14  type Props = (
15    | {
16        message: string;
17        messageId?: never;
18      }
19    | {
20        message?: never;
21        messageId: string;
22      }
23  ) & {
24    variant?: Variant;
25  };
```

```
TERMINAL  PROBLEMS 1  OUTPUT  DEBUG CONSOLE  AZURE  GITLENS

[14:42:13] File change detected. Starting incremental compilation...

pages/02-mutually-exclusive.tsx:10:8 - error TS2322: Type '{ message: string; messageId: string; }' is not assignable to type 'IntrinsicAttributes & Props'.
  Type '{ message: string; messageId: string; }' is not assignable to type '{ message?: undefined; messageId: string; }'.
    Types of property 'message' are incompatible.
      Type 'string' is not assignable to type 'undefined'.

10        <DualAlert
          ~~~~~~~~~
```

More strict features

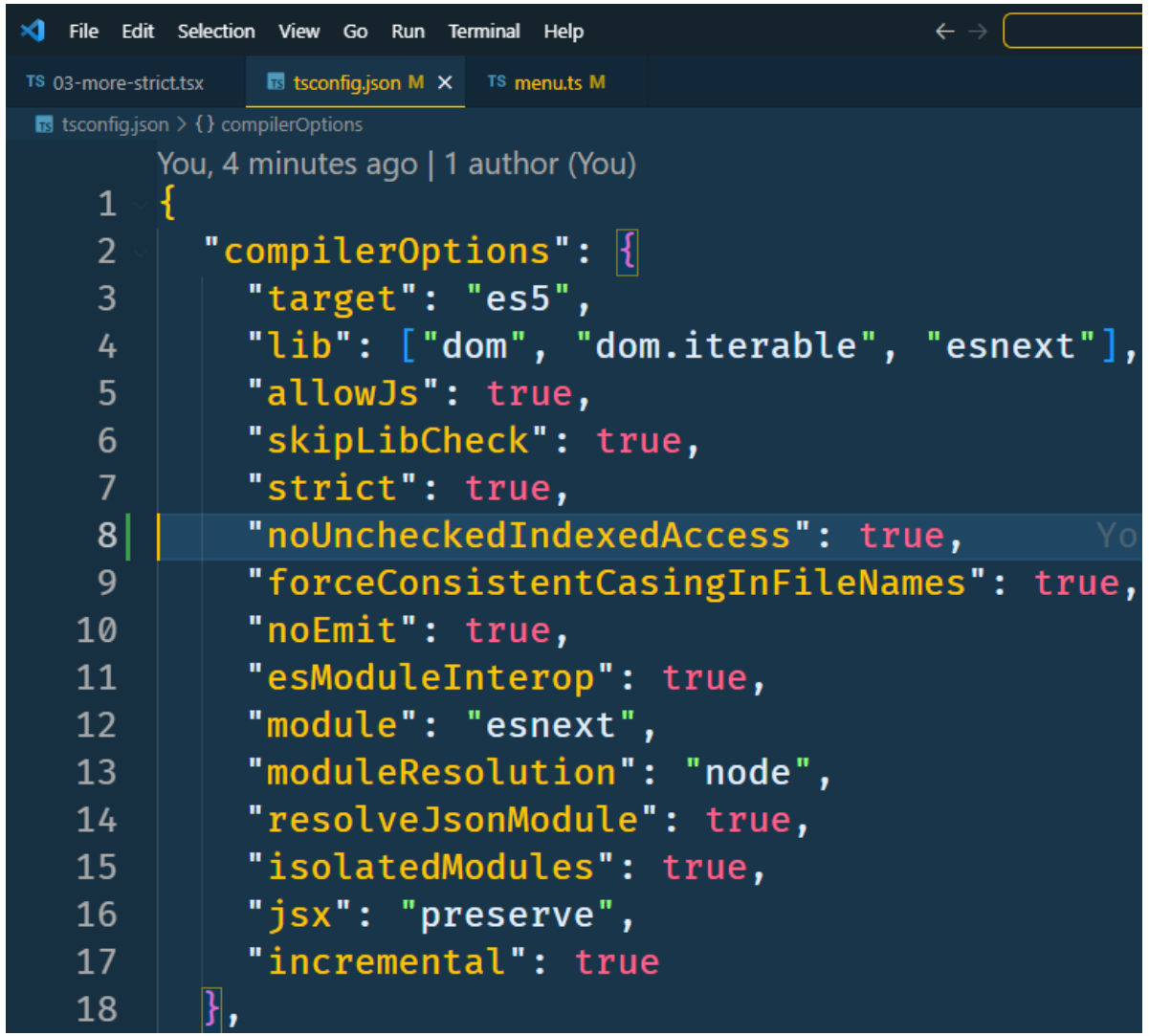
More Strict Features

- There are **many more strict settings** not enabled by "strict"
  - allowUnreachableCode
  - allowUnusedLabels
  - exactOptionalPropertyTypes
  - noFallthroughCasesInSwitch
  - noImplicitOverride
  - noImplicitReturns
  - noPropertyAccessFromIndexSignature
  - noUncheckedIndexedAccess
  - noUnusedLocals
  - noUnusedParameters

noUnchecked IndexedAccess

- By default **every** index from an array is seen as the **array element type**
  - Even if it exceeds the items available and will result in *undefined*
- Enabling "***noUncheckedIndexedAccess"*** requires you to check the element before using
  - Or the element is its type or *undefined*
- Try adding Mushrooms to the Pizza Ai Funghi and observe a runtime error ☹

tsconfig.json



menu.ts

File Edit Selection View Go Run Terminal Help — advanced-react-typescript-2022

03-more-strict.tsx | tsconfig.json M | menu.ts M

src > 03-more-strict > menu.ts > ...

```ts
export const getExtraIngredient = (name: string): ExtraIngredient => {
  const extraIngredient = extraIngredients[name] ?? { name, price: 0 };

  return extraIngredient;
};
```

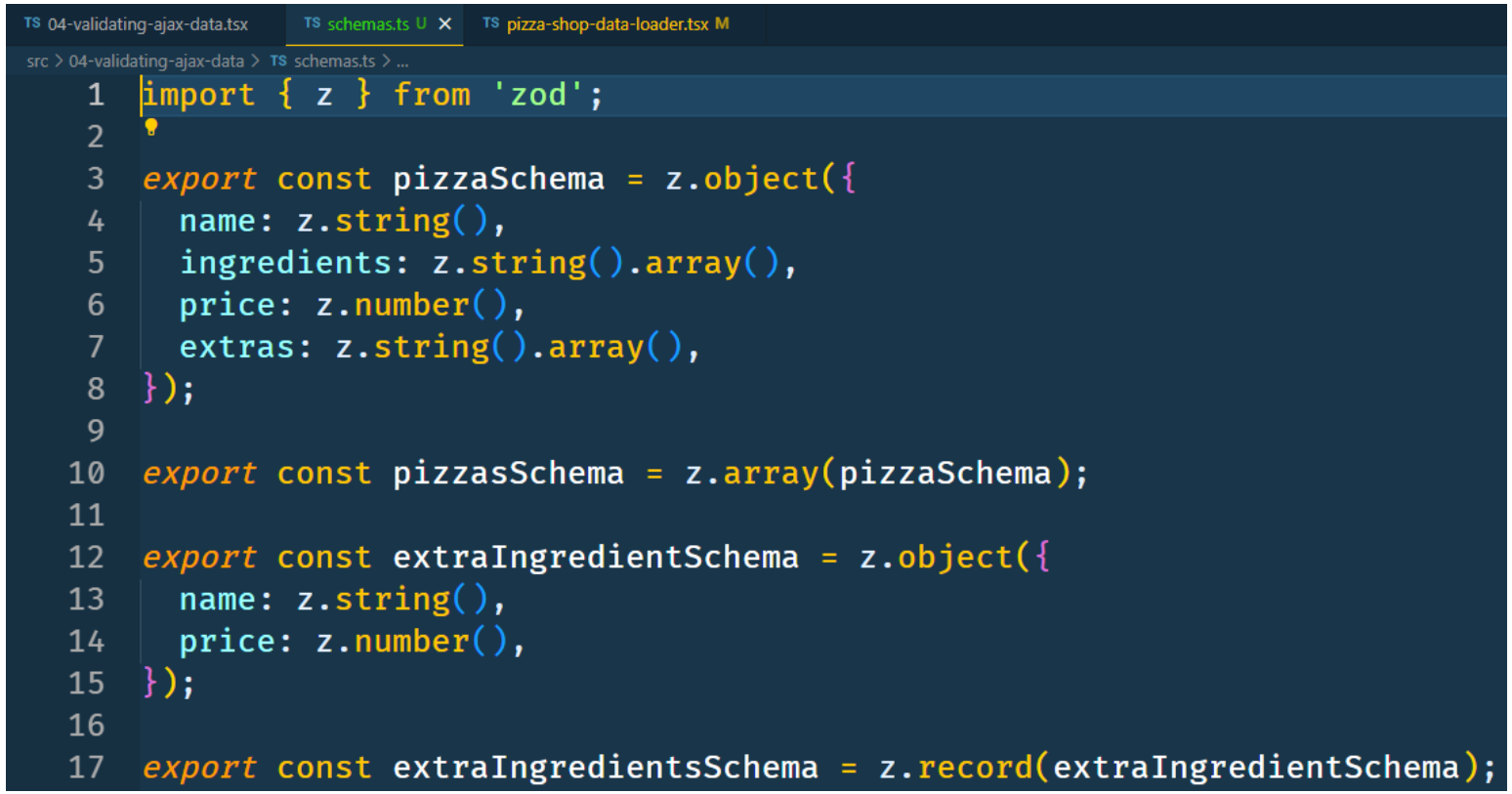
Validating data at the boundary

Validating Data

- The type definitions are **used at compile time**
- They **might not match the runtime** behavior
  - Specially when doing AJAX requests or reading JSON files

- Try adding Mushrooms to the Pizza Ai Funghi and observe another runtime error ☹

schemas.ts



pizza-shop-data-loader.tsx


```tsx
import { PizzaShop } from './pizza-shop';
import { extraIngredientsSchema, pizzasSchema } from './schemas';

const server = 'http://localhost:3000';

export const PizzaShopDataLoader: FC = () => {
  const { data: pizzas, error: pizzasError } = useSWR<Pizza[]>(
    '/api/pizzas.json',
    (resource, init) =>
      fetch(`${server}${resource}`, init)
        .then((res) => res.json())
        .then(pizzasSchema.parse)
  );

  const { data: extraIngredients, error: extraIngredientsError } =
    useSWR<ExtraIngredients>(
      '/api/good-extra-ingredients.json',
      (resource, init) =>
        fetch(`${server}${resource}`, init)
          .then((res) => res.json())
          .then(extraIngredientsSchema.parse)
    );
```

Inferring TypeScript types

Inferring TypeScript types

- In many cases **TypeScript can infer types** from existing objects
  - Not just the object shape but also valid keys
- Use the "***extends"* keyword** to limit a generic type argument
  - One generic argument can be used to infer a second etc.

configuration.tsx

File Edit Selection View Go Run Terminal Help — advanced-react-typescript-2022

05-inferring-types.tsx | configuration.tsx M

src > 05-inferring-types > configuration.tsx > ...

```tsx
function getConfigItem<
  TSection extends keyof typeof config,
  TItem extends keyof typeof config[TSection]
>(section: TSection, item: TItem) {
  const config = {
    user: {
      firstName: 'John',
      lastName: 'Doe',
      birthDate: new Date(1990, 6, 10),
    },
    address: {
      street: 'Main St',
      houseNumber: 123,
      city: 'New York',
    },
  };

  return config[section][item];
}

export const Configuration: FC = () => {
  const firstName = getConfigItem('user', 'firstName');
  const lastName = getConfigItem('user', 'lastName');
  const birthDate = getConfigItem('user', 'birthDate').toLocaleDateString();

  // const employer = getConfigItem('employer', 'name');
```

Generic React prop types

Generic React prop types

- React component **prop types** can also be **generic**
  - To ensure that various props have matching type definitions
- The **generic type can be specified** when the component is rendered
  - Or will automatically inferred if not
- Very powerful to create reusable, flexible but fully typed components

two-forms.tsx

generic-form.tsx

File Edit Selection View Go Run Terminal Help — advanced-react-typescript-2022

06-generic-prop-types.tsx | two-forms.tsx M | generic-form.tsx M

src > 06-generic-prop-types > generic-form.tsx > ...

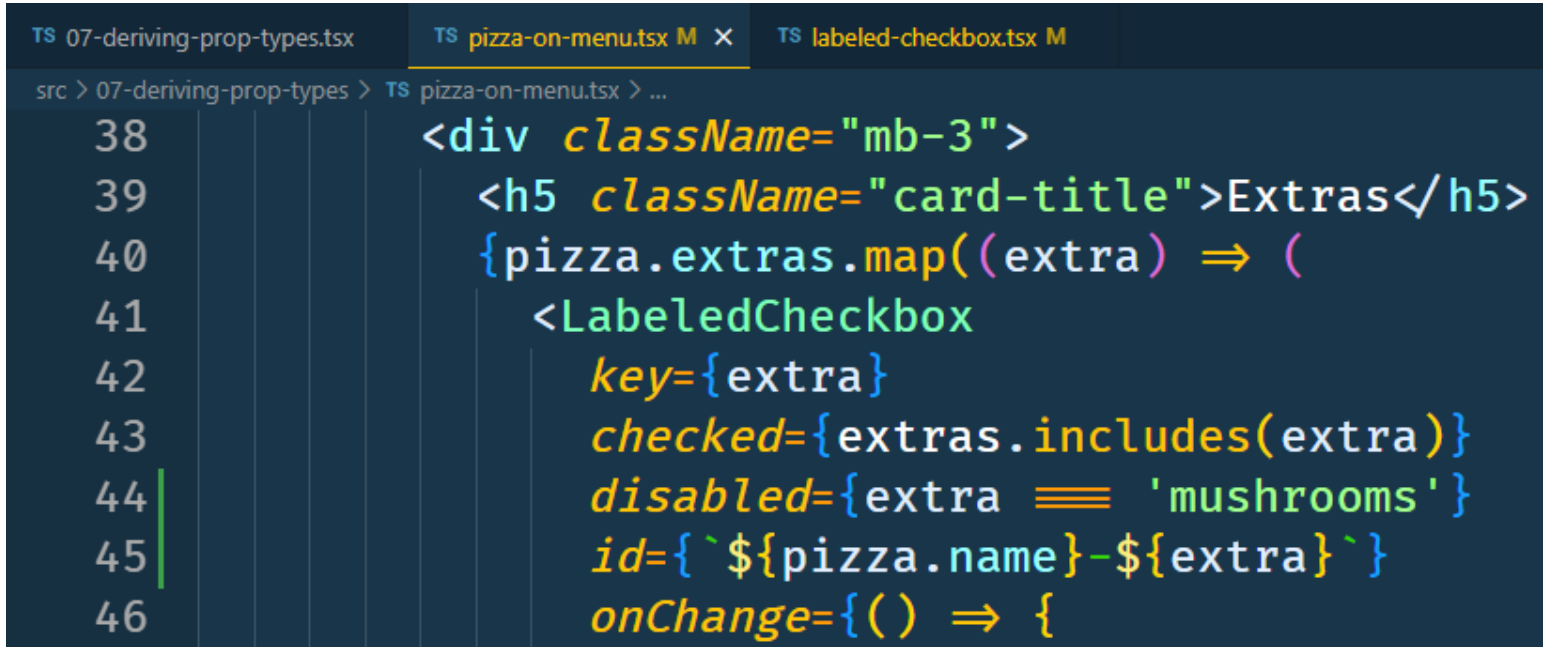
```tsx
type Props<TValues> = {
  header: string;
  initialValues: TValues;
  onSubmit: (values: TValues) => void;
};

export function GenericForm<TValues extends Record<string, any>>({
  header,
  initialValues,
  onSubmit,
}: Props<TValues>) {
  const [values, setValues] = useState(initialValues);

  return (
    <form className="mb-5" onSubmit={() => onSubmit(values)}>
      <h3 className="mb-3">{header}</h3>

      {Object.keys(values).map((key) => (
        <LabeledInput
          key={key}
          value={values[key]}
          onChange={(e) => setValues({ ...values, [key]: e.target.value })}
        >
```

Deriving component prop types

Deriving component prop types

- **Infer a component Prop type**
  - Using *React.ComponentProps<typeof Component>*
- **No need to publicly export** all those prop definitions
  - Just in case they are needed
- Very useful when you want to export the **nested component props**
  - Use type mappings to modify the type as needed

pizza-on-menu.tsx



labeled-checkbox.tsx

TS 07-deriving-prop-types.tsx | TS pizza-on-menu.tsx M | TS labeled-checkbox.tsx M

src > components > TS labeled-checkbox.tsx > ...

You, 2 minutes ago | 1 author (You)

```tsx
import { ComponentProps, FC, useId } from 'react';      You, 2 minutes ago
import { Checkbox } from './checkbox';
import { Label } from './label';

type Props = ComponentProps<typeof Checkbox>;

export const LabeledCheckbox: FC<Props> = ({ children, id, ...props }) => {
  const internalId = useId();

  return (
    <div className="form-check">
      <Checkbox id={id ?? internalId} {...props} />
      <Label htmlFor={id ?? internalId}>{children}</Label>
    </div>
  );
};
```

Inferring Zod schema types

Inferring Types

- Maintaining **a Zod schema and a TypeScript interface** is tedious
  - Both have to be kept in sync
- The TypeScript types can be **inferred** from the Zod schema
  - Using "z.infer<typeof schema>"

types.ts

TS 08-inferring-schema-types.tsx | TS schemas.ts | TS types.ts M

src > 08-inferring-schema-types > TS types.ts > ...

You, 1 minute ago | 1 author (You)

```ts
import { z } from 'zod';
import {
  extraIngredientSchema,
  extraIngredientsSchema,
  pizzaSchema,
} from './schemas';

export type Pizza = z.infer<typeof pizzaSchema>;

export type ExtraIngredients = z.infer<typeof extraIngredientsSchema>;

export type ExtraIngredient = z.infer<typeof extraIngredientSchema>;

export type PizzaOnOrder = {
  name: string;
  price: number;
  extraIngredients: ExtraIngredient[];
};
```

You, 1 minute ago • Uncommitted changes

Type mapping with Omit<> and/or Pick<>

Type mapping with Omit<> and/or Pick<>

- **Type mapping** lets you **create a new type** based on an existing type
  - With one or more modifications
- There are many **build in type mappings**
  - "*Omit<T>*": Create a new type by removing one or more pros
  - "*Pick<T>*": Create a new type with just the specified props
- A type mapping can contain **conditional logic** to alter a part of the type

types.ts

TS 09-type-mapping.tsx | TS types.ts M ×

src > 09-type-mapping > TS types.ts > ...

```ts
export type Pizza = z.infer<typeof pizzaSchema>;
export type ExtraIngredients = z.infer<typeof extraIngredientsSchema>;
export type ExtraIngredient = z.infer<typeof extraIngredientSchema>;

export type PizzaOnOrder = Pick<Pizza, 'name' | 'price'> & {
  extraIngredients: ExtraIngredient[];
};
```

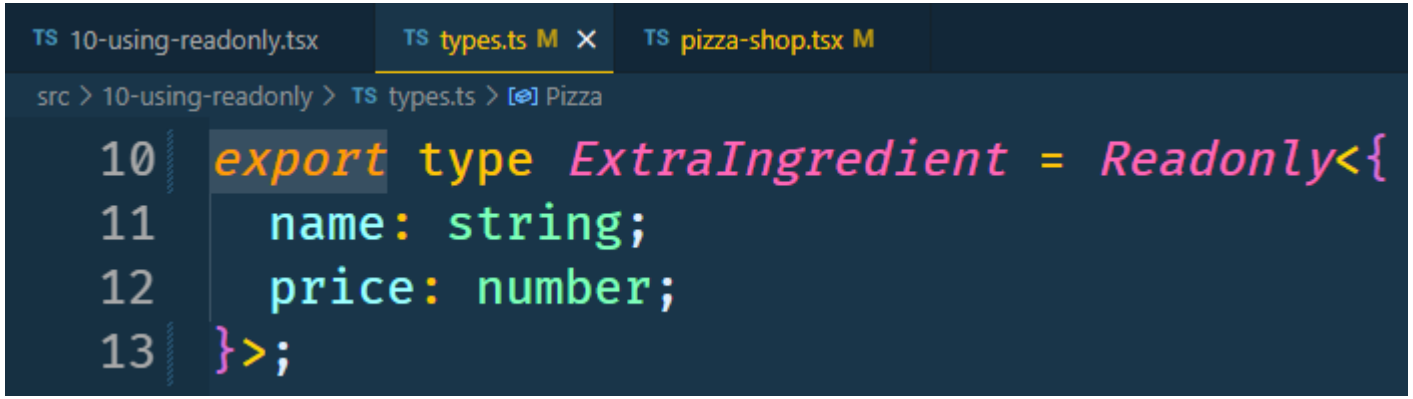
Using Readonly<>

Readonly<T>

- The *Readonly<T>* mapped type **creates a read-only mapped type**
  - Can't change properties anymore
  - Or use "array.push()" etc.
- ⚠ *Readonly<T>* is **not recursive** ⚠
  - Only the first level of properties becomes read-only
- 💡 Recommended for **function arguments** to show intent 💡
  - And AJAX responses etc.

- Place order with Pizza Quattro Formaggi with extra cheese twice and notice the price difference ☹

types.ts



pizza-shop.tsx

10-using-readonly.tsx | types.ts M | pizza-shop.tsx M

src > 10-using-readonly > pizza-shop.tsx > ...

```tsx
14  const onPlaceOrder = () => {
15    const extrasForAEuro = order.flatMap((pizza) =>
16      pizza.extraIngredients.filter((extra) => extra.price === 1)
17    );
18    console.log('Extras for a Euro', extrasForAEuro);
```

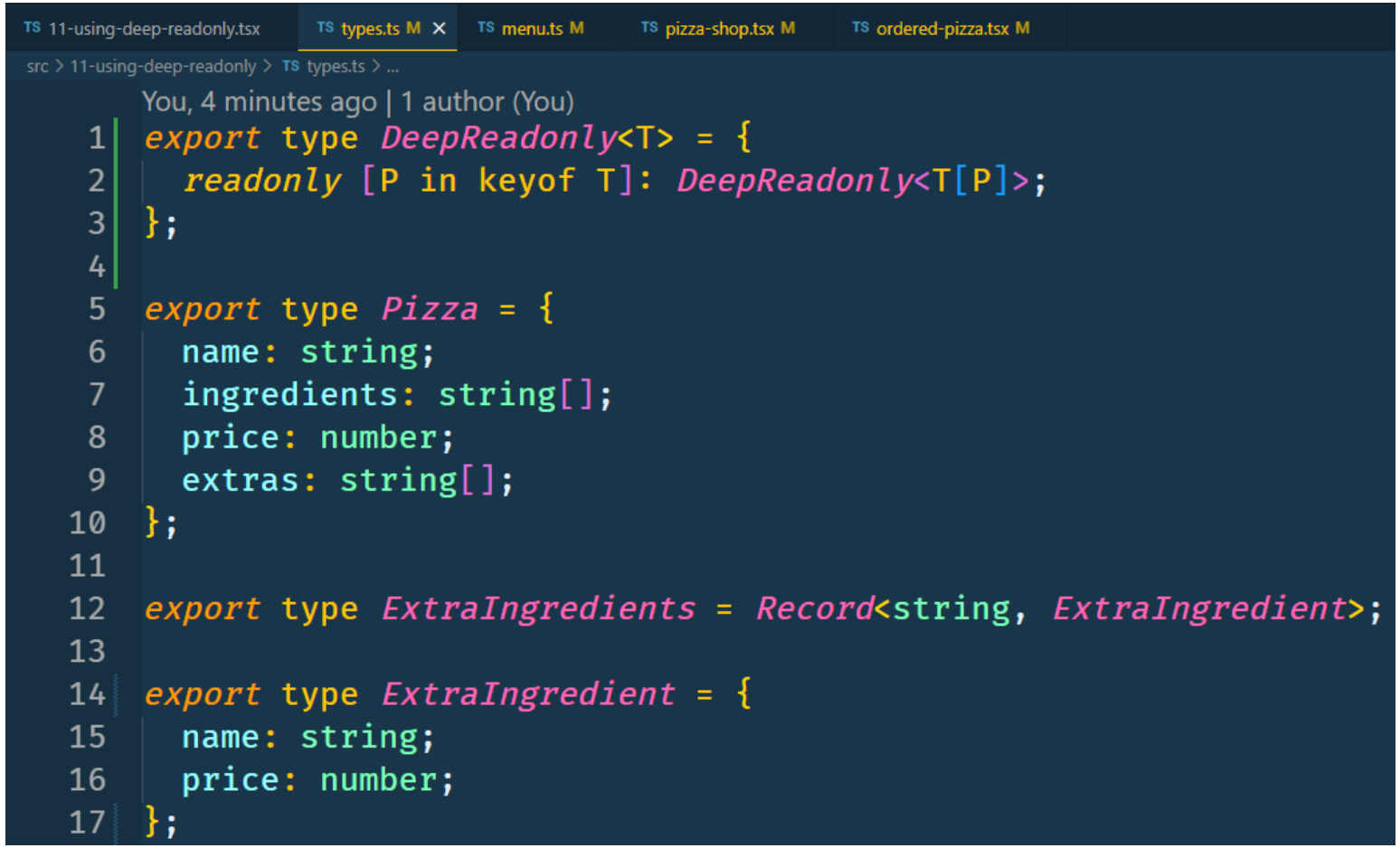
Custom type mapping DeepReadonly<>

DeepReadonly<T>

- Make a whole **nested object structure read-only**
  - Recursive mapped types are very powerful
  - An improvement over the default "*Readonly<T>*"
- Source: https://gist.github.com/basarat/1c2923f91643a16a90de638e76bce0ab

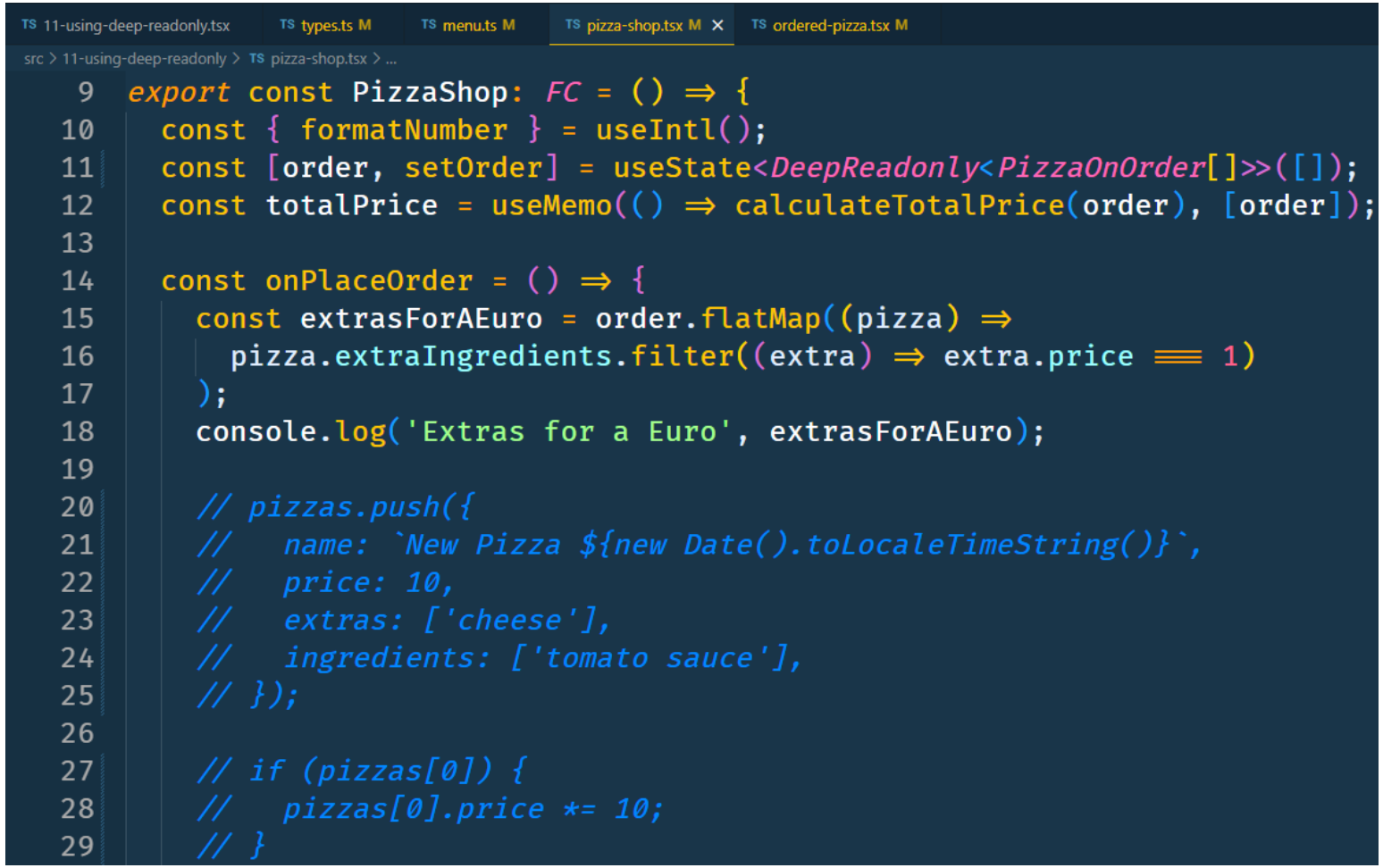
Place order Pizza Margherita twice and notice the price difference ☹

types.ts



menu.ts

pizza-shop.tsx



ordered-pizza.tsx

TS 11-using-deep-readonly.tsx | TS types.ts M | TS menu.ts M | TS pizza-shop.tsx M | TS ordered-pizza.tsx M

src > 11-using-deep-readonly > TS ordered-pizza.tsx > ...

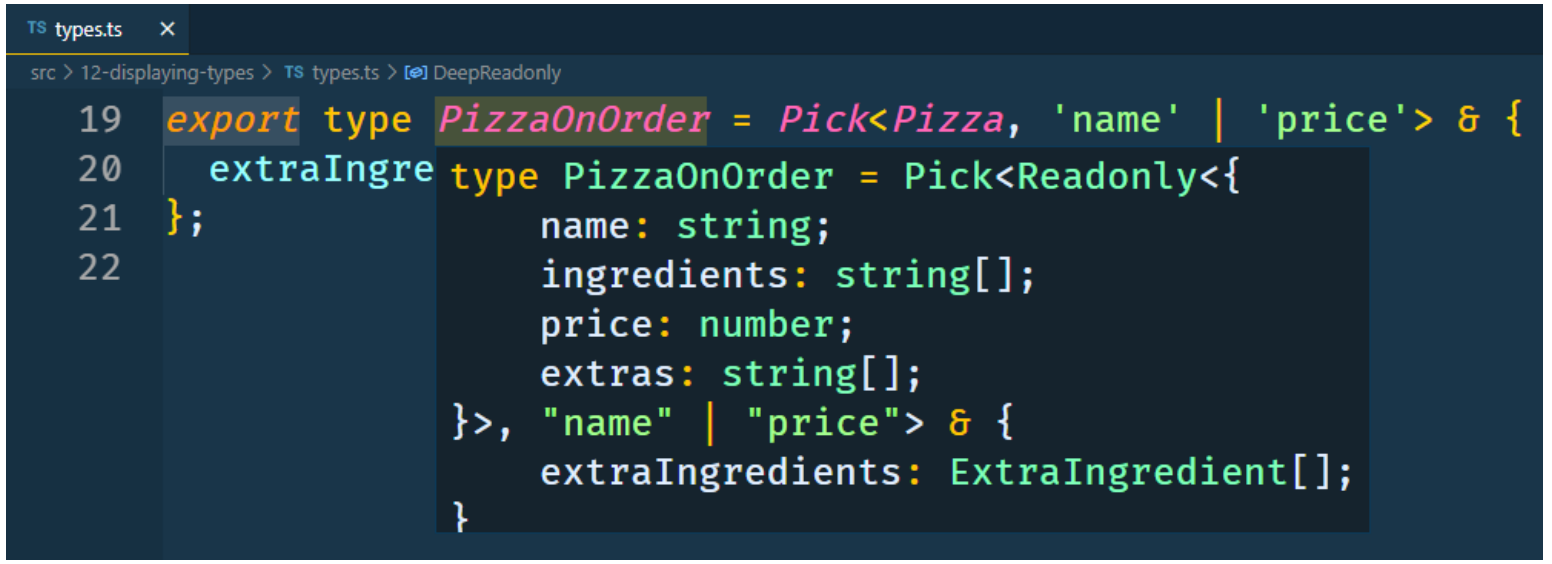
```tsx
type Props = {
  pizza: DeepReadonly<PizzaOnOrder>;
};

export const OrderedPizza: FC<Props> = ({ pizza }) => {
```

Displaying types

Displaying Types

- A **disadvantage of mapped types** is that the type definition in tooltips becomes **hard to read**
  - It shows how a type is constructed instead of the resulting type
- The "*Resolve<T>*" turns this into **the resulting type** instead
  - Source: https://effectivetypescript.com/2022/02/25/gentips-4-display/

types.ts



types.ts


```
TS types.ts M ×
src > 12-displaying-types > TS types.ts > ...
18
19  // Taken fro                                    /25/gentips-4-display/
20  type Resolve                                   yof T]: T[K] };
21
22  export type PizzaOnOrder = Resolve<
23    Pick<Pizza, 'name' | 'price'> & {
24      extraIngredients: ExtraIngredient[];
25    }
26  >;
```

```
type PizzaOnOrder = {
    readonly name: string;
    readonly price: number;
    extraIngredients: ExtraIngredient[];
}
```

Type Predicate Functions

Type Predicate Functions

- Often a **TypeScript cast** is used when types don't quite line up
  - But that is **just silencing the compiler**
    - ⚠ Casting via "unknown" will even allow any (invalid) type cast ⚠
  - There is no runtime checking or guarantee
- A **type predicate** can do a cast in a runtime safe manner
  - 💡 Checks **both at runtime and compile time** 💡
  - A **function that returns a "boolean"** to indicate if the type matches

types.ts

item-on-sale.tsx

TS 13-predicates-assertions.tsx | TS types.ts M | TS item-on-sale.tsx M

src > 13-predicates-assertions > TS item-on-sale.tsx > ...

```tsx
type Props = {
  item: ItemsOnSale;
};

export const ItemOnSale: FC<Props> = ({ item }) => {
  if (isBook(item)) {
    return <BookOnSale book={item} />;
  } else if (isMagazine(item)) {
    return <MagazineOnSale magazine={item} />;
  }
  // case 'pen':
  //   return <PenOnSale pen={item as Pen} />;

  return null;
};
```

Type Assertion Functions

Type Assertion Functions

- **Type assertion functions** can be even easier
  - Throw an error if the type doesn't match
- Often a **better alternative** then a cast
  - The code will not continue if the assumption is wrong

types.ts

TS 13-predicates-assertions.tsx | TS types.ts M | TS item-on-sale.tsx M

src > 13-predicates-assertions > TS types.ts > Book

```ts
export function assertBook(item: ItemsOnSale): asserts item is Book {
  if (!isBook(item)) {
    throw new Error('Item is not a book');
  }
}

export function assertMagazine(item: ItemsOnSale): asserts item is Magazine {
  if (!isMagazine(item)) {
    throw new Error('Item is not a magazine');
  }
}

export function assertPen(item: ItemsOnSale): asserts item is Pen {
  if (!isPen(item)) {
    throw new Error('Item is not a pen');
  }
}
```

item-on-sale.tsx

TS 13-predicates-assertions.tsx | TS types.ts M | TS item-on-sale.tsx M

src > 13-predicates-assertions > TS item-on-sale.tsx > ...

```tsx
type Props = {
  item: ItemsOnSale;
};

export const ItemOnSale: FC<Props> = ({ item }) => {
  switch (item.type) {
    case 'book':
      assertBook(item);
      return <BookOnSale book={item} />;
    case 'magazine':
      assertMagazine(item);
      return <MagazineOnSale magazine={item} />;
    // case 'pen':
    //   return <PenOnSale pen={item as Pen} />;
  }

  return null;
};
```

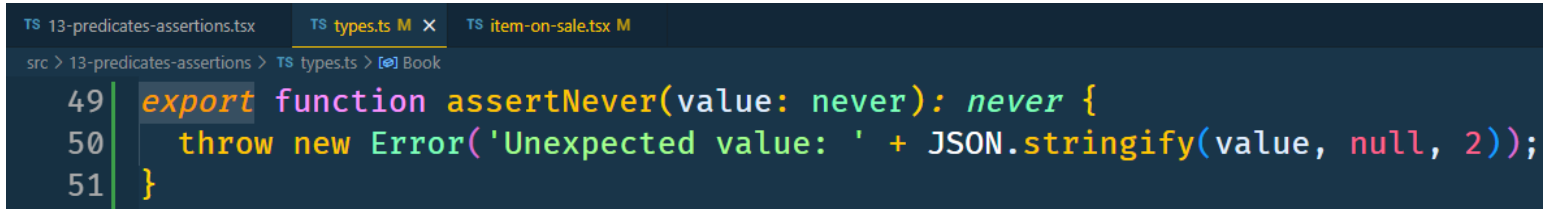
Exhaustiveness Checking

Exhaustiveness Checking

- The TypeScript compiler doesn't tell us if **every case** is provided
  - It's easy to forget to add a switch case when an enumeration is expanded
- The "**never" type** is a great way to make sure
  - A compile error if the default case can be reached
  - 💡 Make sure to add an exception or error logging at runtime 💡

types.ts

types.ts



item-on-sale.tsx

TS 13-predicates-assertions.tsx | TS types.ts M | TS item-on-sale.tsx M

src > 13-predicates-assertions > TS item-on-sale.tsx > ...

```tsx
type Props = {
  item: ItemsOnSale;
};

export const ItemOnSale: FC<Props> = ({ item }) => {
  switch (item.type) {
    case 'book':
      return <BookOnSale book={item} />;
    case 'magazine':
      return <MagazineOnSale magazine={item} />;
    case 'pen':
      return <PenOnSale pen={item} />;
    default:
      assertNever(item);
  }

  return null;
};
```

Conclusion

- TypeScript's **strict settings** help catch many errors
  - Make sure to turn on the additional strict features as well
- TypeScript offers a lot of features for **React component props**
  - Infer or mutate prop types and detect invalid combinations of values
- **Validate all data at boundaries**
  - Not just from the user, also from API's
- Use **mapped types** to create new types
  - The possibilities are almost endless
- Use **type predicates and assertions** both at compile and run-time
  - Instead of just casting at compile time
- Enable **exhaustiveness checking** with the "never" type
  - Make sure to log unexpected cases at runtime

Maurice de Beijer

@mauricedb

maurice.de.beijer@gmail.com