



September 21 & 22, 2023
4pm CEST / 7am PST / 10am EST

The Type of Conference Developers Need

Nº1 MOST ADOPTED
TECHNOLOGY
Based on year-on-year growth

20+ INDUSTRY
EXPERTS
Sharing know-hows

5K+ WEB DEVS
Gathering in the
cloud

[Stream page](#)

[Reserve a spot](#)

Practice TypeScript Techniques Building React Server Components App

Maurice de Beijer
@mauricedb



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



Topics

- **Compiling** the code to catch type errors early
- Using the *satisfies* operator
- Use **type mappings** to optimize your code
- **Custom type mappings** are really powerful
 - Can help detect all sorts of errors quickly
 - Can make types more readable in IntelliSense
- Use **Template Literal Types** to manipulate types
- Use **Opaque Types** to help with type safety
- Make TypeScript even stricter and **catch more potential errors**
- Prevent objects from being **mutated by accident**
- **Improve performance** when importing large JSON files

Type it out
by hand?

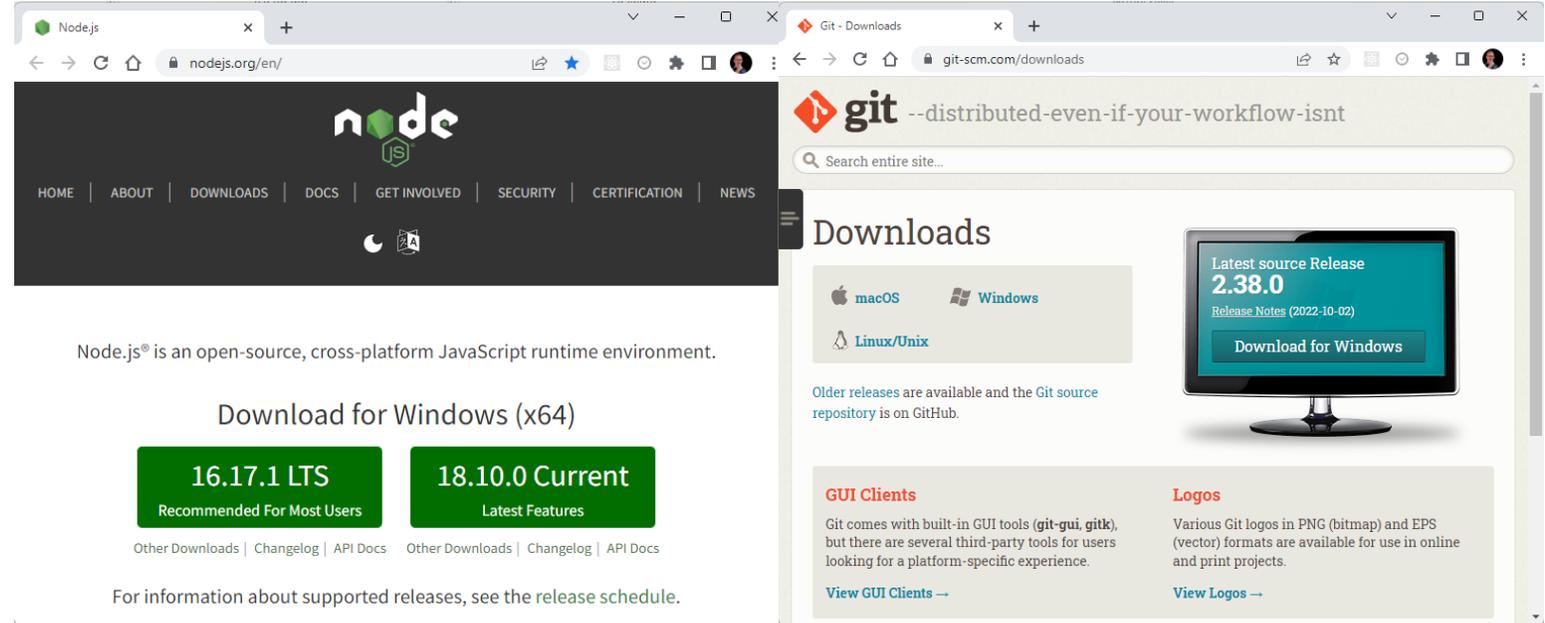
"Typing it drills it into your brain much better than simply copying and pasting it. You're forming new neuron pathways. Those pathways are going to help you in the future. Help them out now!"

Prerequisites

Install Node & NPM

Install the GitHub repository

Install Node.js & NPM



```
TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE  AZURE

PS C:\Repos\ts-conf-2023-ws> node --version
• v16.17.1
PS C:\Repos\ts-conf-2023-ws> git --version
• git version 2.41.0.windows.1
PS C:\Repos\ts-conf-2023-ws> npm --version
9.7.2
```

Following Along

```
src > components > TS shopping-cart.tsx > ShoppingCartProvider
 8 | type ShoppingCartMovie = Pick<Movie, 'id' | 'title'>
 9 |
10 | const ShoppingCartContext = createContext({
11 |   addMovie: (movie: ShoppingCartMovie) => {},
12 |   checkout: () => {},
13 |   itemCount: 0,
14 | })
15 |
16 | export const useShoppingCart = () => {
17 |   return useContext(ShoppingCartContext)
18 | }
19 | export function ShoppingCartProvider({ children }: PropsWithChildren) {
20 |   const [checkoutOpen, setCheckoutOpen] = useState(false)
21 |   const [movies, setMovies] = useState<ShoppingCartMovie[]>([])
22 |
23 |   return (
24 |     <ShoppingCartContext.Provider
25 |       value={{
26 |         addMovie: (movie: ShoppingCartMovie) => {
27 |           setMovies((movies) => [...movies, movie])
28 |         },
```

- Repo: <https://github.com/mauricedb/ts-conf-2023-ws>
- Slides: <https://bit.ly/ts-conf-2023-ws>

Create a new Next.js app

```
PS C:\Repos> npx create-next-app@latest
Need to install the following packages:
  create-next-app@13.4.19
Ok to proceed? (y)
✓ What is your project named? ... ts-conf-2023-ws
✓ Would you like to use TypeScript? ... No / Yes
✓ Would you like to use ESLint? ... No / Yes
✓ Would you like to use Tailwind CSS? ... No / Yes
✓ Would you like to use `src/` directory? ... No / Yes
✓ Would you like to use App Router? (recommended) ... No / Yes
✓ Would you like to customize the default import alias? ... No / Yes
Creating a new Next.js app in C:\Repos\ts-conf-2023-ws.
```

Adding Shadcn components

```
PS C:\Repos\ts-conf-2023-ws> npx shadcn-ui@latest init
√ Would you like to use TypeScript (recommended)? ... no / yes
√ Which style would you like to use? » Default
√ Which color would you like to use as base color? » Zinc
√ Where is your global CSS file? ... src\app\globals.css
√ Would you like to use CSS variables for colors? ... no / yes
√ Where is your tailwind.config.js located? ... tailwind.config.ts
√ Configure the import alias for components: ... @/components
√ Configure the import alias for utils: ... @/lib/utils
√ Are you using React Server Components? ... no / yes
√ Write configuration to components.json. Proceed? ... yes

√ Writing components.json...
√ Initializing project...
√ Installing dependencies...

Success! Project initialization completed.
```

The changes



mauricedb / ts-conf-2023-ws

Code Issues Pull requests Actions Projects Wiki Settings

Commits

main

Commits on Sep 6, 2023

- Typing JSON files mauricedb committed 3 hours ago ✓ [372bfeb](#)
- Template Literal Types mauricedb committed 3 hours ago [c45f523](#)

Commits on Sep 1, 2023

- Custom type mapping DeepReadonly<> mauricedb committed 5 days ago ✓ [a5536af](#)
- Using Readonly<> mauricedb committed 5 days ago [1319836](#)
- More Strict Features & noUncheckedIndexedAccess mauricedb committed 5 days ago [0af8664](#)

Commits on Aug 31, 2023

- Opaque Types mauricedb committed last week [a38d503](#)
- The display of types mauricedb committed last week ✓ [45997bc](#)
- Custom type mapping mauricedb committed last week [99fa27d](#)

Commits on Aug 30, 2023

- Prevent over fetching mauricedb committed last week [2343ef6](#)
- Satisfies operator mauricedb committed last week [93247db](#)
- Compile and Test on GitHub [7-07270](#)

Clone the GitHub Repository

```
Windows PowerShell
PS C:\demos> git clone https://github.com/mauricedb/ts-conf-2023-ws.git
Cloning into 'ts-conf-2023-ws'...
remote: Enumerating objects: 229, done.
remote: Counting objects: 100% (229/229), done.
remote: Compressing objects: 100% (144/144), done.
Receiving objects: 93% (213/229)used 187 (delta 73), pack-reused 0Receiving objects: 81% (186/229)
Receiving objects: 100% (229/229), 1.43 MiB | 6.25 MiB/s, done.
Resolving deltas: 100% (104/104), done.
PS C:\demos>
```

Install NPM Packages

```
Windows PowerShell
PS C:\demos> cd .\ts-conf-2023-ws\
PS C:\demos\ts-conf-2023-ws> npm install

> ts-conf-2023-ws@0.1.0 postinstall
> prisma migrate dev --name init

Environment variables loaded from .env
Prisma schema loaded from prisma\schema.prisma
Datasource "db": SQLite database "dev.db" at "file:./dev.db"

SQLite database dev.db created at file:./dev.db

Applying migration `20230824122654_init`

The following migration(s) have been applied:

migrations/
├─ 20230824122654_init/
└─ migration.sql

Your database is now in sync with your schema.

✓ Generated Prisma Client (v5.2.0) to .\node_modules\@prisma\client in 68ms

Running seed command `ts-node --compiler-options {"module":"CommonJS"} prisma/seed.ts` ...

The seed command has been executed.

added 453 packages, and audited 454 packages in 22s

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

found 0 vulnerabilities
```

Start branch

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

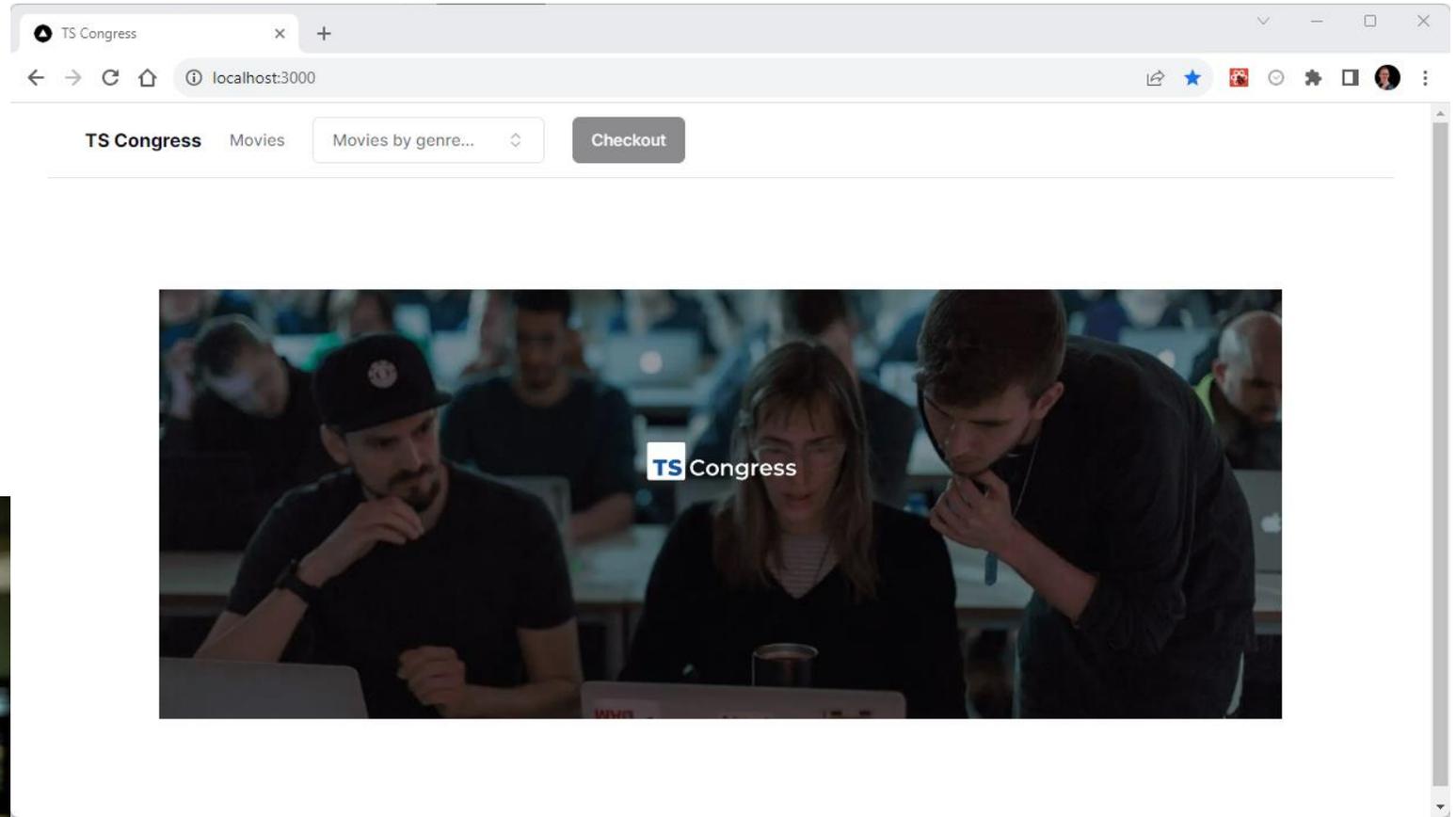
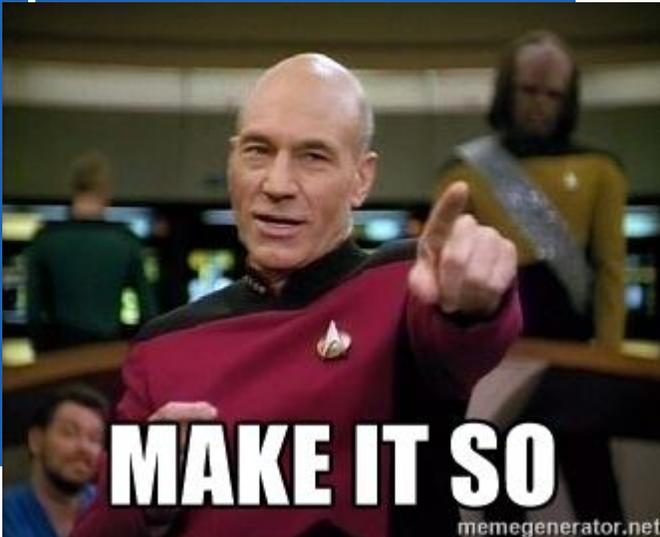
Start the application

```
next-render-worker-pages x + v
PS C:\demos\ts-conf-2023-ws> npm run dev
> ts-conf-2023-ws@0.1.0 dev
> next dev

- info Loaded env from C:\demos\ts-conf-2023-ws\.env
- warn You have enabled experimental feature (serverActions) in next.config.js.
- warn Experimental features are not covered by semver, and may cause unexpected or broken application behavior. Use at your own risk.

- ready started server on [::]:3000, url: http://localhost:3000
- event compiled client and server successfully in 243 ms (20 modules)
- wait compiling...
- event compiled client and server successfully in 162 ms (20 modules)
- info Loaded env from C:\demos\ts-conf-2023-ws\.env
- info Loaded env from C:\demos\ts-conf-2023-ws\.env
- wait compiling /page (client and server)...
- event compiled client and server successfully in 4.4s (785 modules)
- wait compiling...
- event compiled successfully in 246 ms (404 modules)
- wait compiling /favicon.ico/route (client and server)...
- event compiled client and server successfully in 1440 ms (831 modules)
```

The application



Compiling the code

Compiling the code

- Quite often **TypeScript code is not type checked** during development
 - Create React App use Babel (JavaScript)
 - Vite uses ESBuild (Go)
 - Next.js uses SWC (Rust)
- These **remove TypeScript annotations** and treat the code as modern JavaScript
 - This means that TypeScript type errors can go unnoticed 😞
- Run the **TypeScript compiler to check the code**

package.json

```
{ } package.json M x
{ } package.json > ...
  ▶ Debug
5   "scripts": {
6     "dev": "next dev",
7     "build": "next build",
8     "start": "next start",
9     "lint": "next lint",
10    "compile": "tsc --noEmit",
11    "compile:watch": "tsc --noEmit --watch",
12    "postinstall": "prisma migrate dev --name init",
```

npm run compile

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE AZURE GITLENS COMMENTS

[10:47:59] Starting compilation in watch mode...

```
src/components/movie-card.tsx:50:61 - error TS2345: Argument of type 'Pick<{ id: number; title: string; overview: string; release_date: string; backdrop_path: string; poster_path: string; popularity: number; vote_average: number; vote_count: number; }, "title" | "id" | "overview" | "backdrop_path" | "vote_average">' is not assignable to parameter of type '{ id: number; title: string; overview: string; release_date: string; backdrop_path: string; poster_path: string; popularity: number; vote_average: number; vote_count: number; }'.
```

```
Type 'Pick<{ id: number; title: string; overview: string; release_date: string; backdrop_path: string; poster_path: string; popularity: number; vote_average: number; vote_count: number; }, "title" | "id" | "overview" | "backdrop_path" | "vote_average">' is missing the following properties from type '{ id: number; title: string; overview: string; release_date: string; backdrop_path: string; poster_path: string; popularity: number; vote_average: number; vote_count: number; }': release_date, poster_path, popularity, vote_count
```

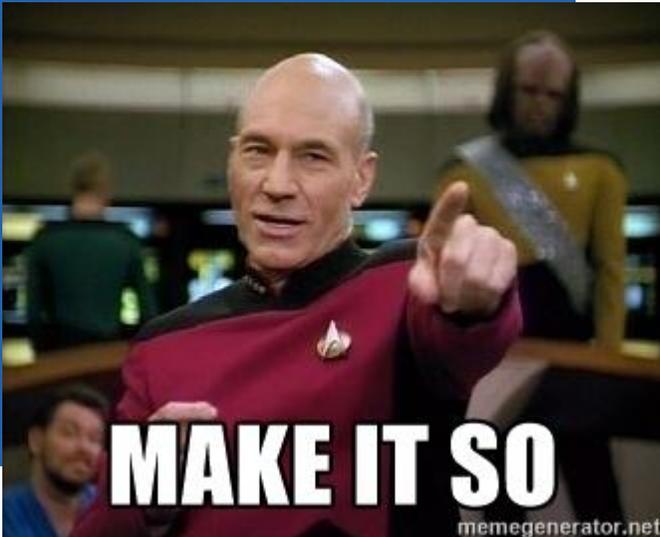
```
50 <Button variant="secondary" onClick={() => addMovie(movie)}>
```

[10:48:01] Found 1 error. Watching for file changes.

shopping- cart.tsx

```
{} package.json M TS movie-card.tsx TS shopping-cart.tsx M X
src > components > TS shopping-cart.tsx > ShoppingCartProvider
 8 | type ShoppingCartMovie = Pick<Movie, 'id' | 'title'>
 9 |
10 | const ShoppingCartContext = createContext({
11 |   addMovie: (movie: ShoppingCartMovie) => {},
12 |   checkout: () => {},
13 |   itemCount: 0,
14 | })
15 |
16 | export const useShoppingCart = () => {
17 |   return useContext(ShoppingCartContext)
18 | }
19 | export function ShoppingCartProvider({ children }: PropsWithChildren) {
20 |   const [checkoutOpen, setCheckoutOpen] = useState(false)
21 |   const [movies, setMovies] = useState<ShoppingCartMovie[]>([])
22 |
23 |   return (
24 |     <ShoppingCartContext.Provider
25 |       value={{
26 |         addMovie: (movie: ShoppingCartMovie) => {
27 |           setMovies((movies) => [...movies, movie])
28 |         },
```

npm run compile



```
TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE  AZURE  GITLENS  COMMENTS  
[10:52:09] Starting compilation in watch mode...  
[10:52:11] Found 0 errors. Watching for file changes.
```

Compile and Test on GitHub

Compile and Test on GitHub

- Make sure to **check your code with each pull request** on GitHub
 - Include ESLint and Next build
- **Generate types** from other sources if appropriate
 - Prisma, GraphQL, OpenAPI etc.
 - When included in GitHub they can be out of date 😞

compile-and-test.yml

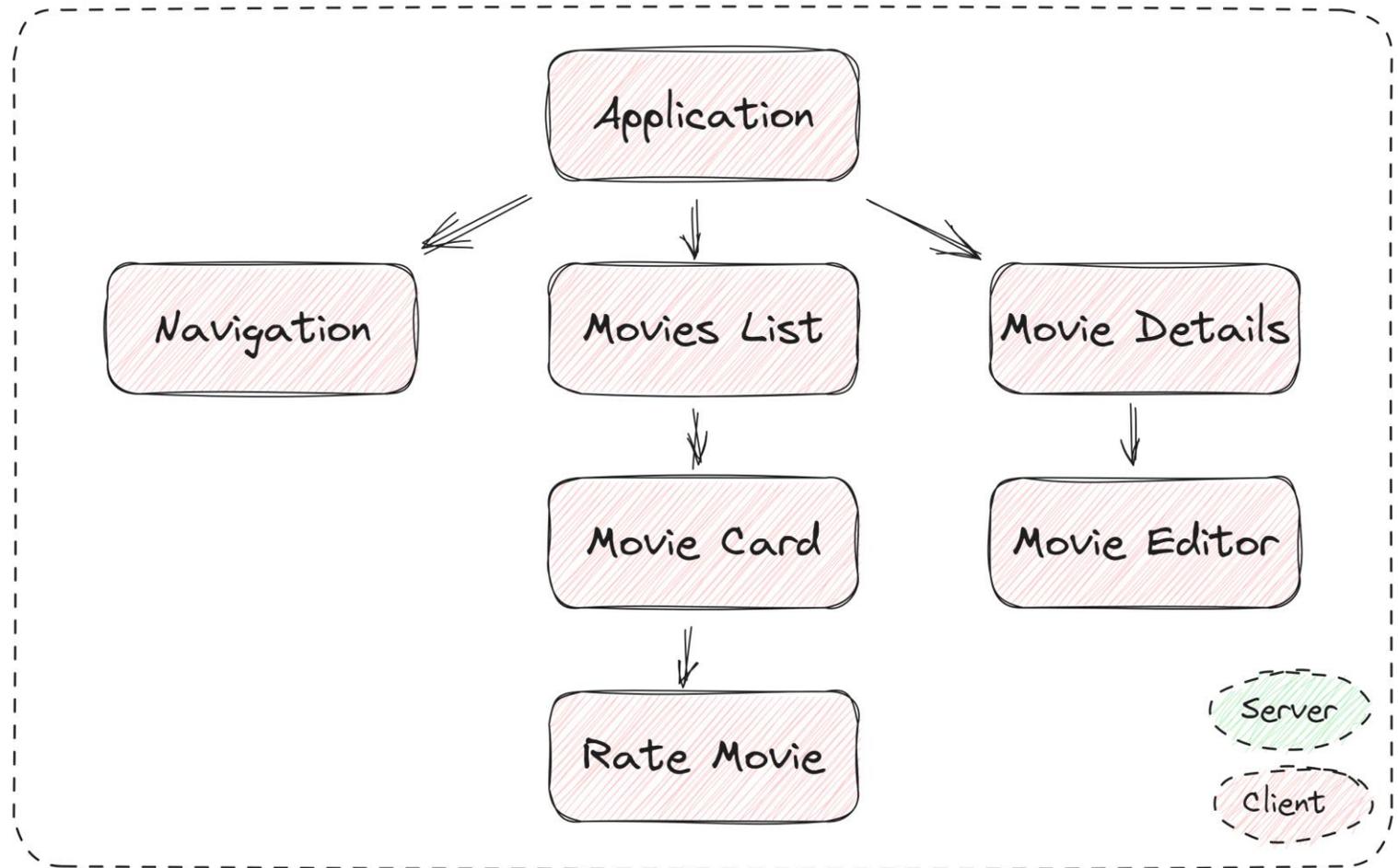
```
compile-and-test.yml U X
.github > workflows > compile-and-test.yml
1 name: Compile and Test
2 on:
3   push:
4     branches: [main]
5   pull_request:
6     branches: [main]
7
8 jobs:
9   test:
10    timeout-minutes: 15
11    runs-on: ubuntu-latest
12
13    steps:
14      - uses: actions/checkout@v3
15      - uses: actions/setup-node@v3
16        with:
17          node-version: 18
18          cache: 'npm'
19      - name: Install dependencies
20        run: npm ci --no-audit --prefer-offline
21      - name: Type check
22        run: npm run compile
23      - name: Lint
24        run: npm run lint
25      # - name: Test
26        # run: npm run test
27      - name: Build
28        run: npm run build
```

What are React Server Components?

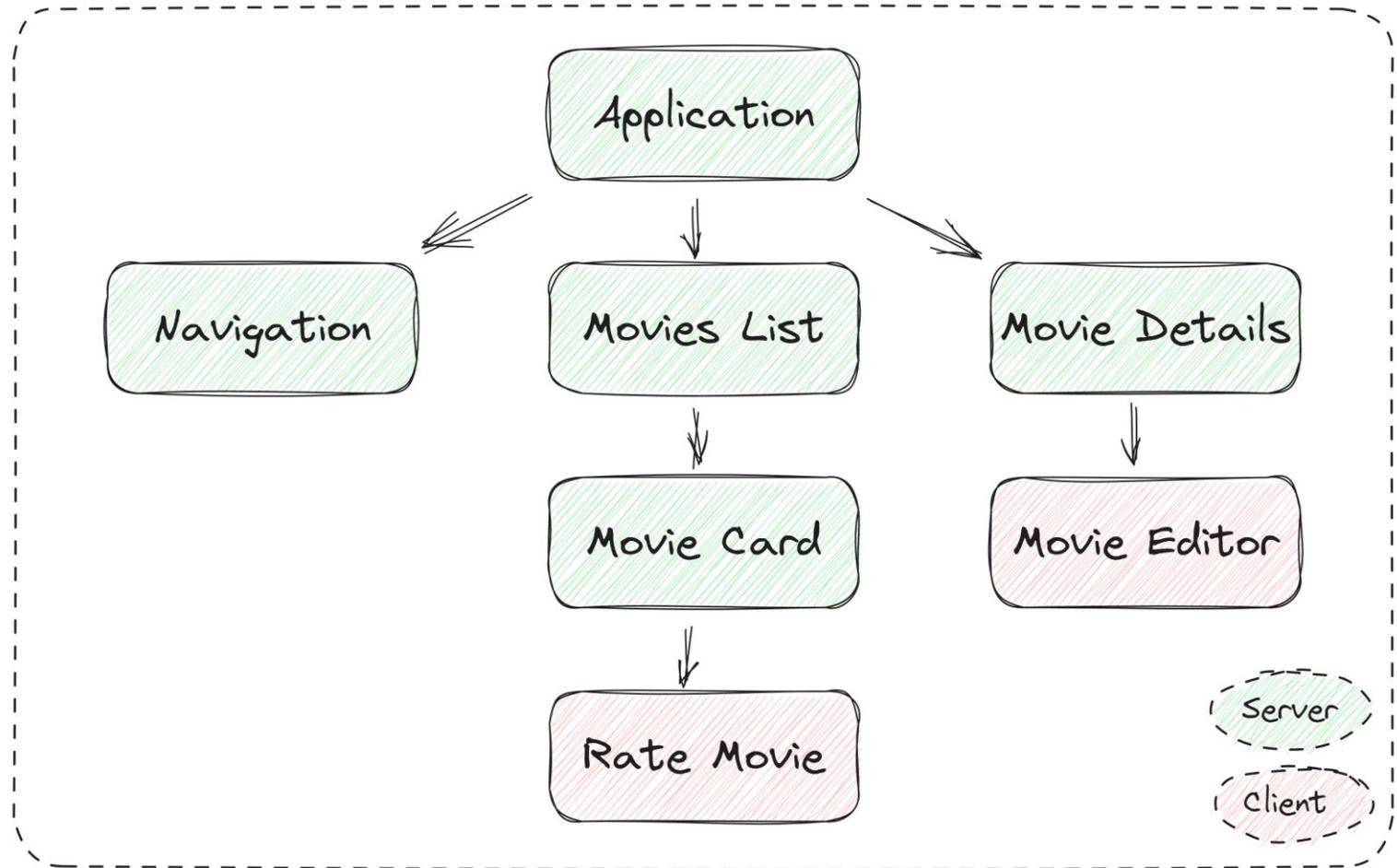
React Server Components

- React Server Components **only execute on the server**
 - Traditionally React components always execute in the browser
- This is **not the same as Server Side Rendering**
 - With SSR components are executed both on the client and server
- Applications are a **combination of server and client components**
- The result is that the back and front-end **code are more integrated**
 - Leading to **better type safety** 😊

Before RSC



With RSC



React Server Components

- Server components can be **asynchronous**
 - Great to load data from some API
- Server components **render just once**
 - No re-rendering with state changes or event handling
- The server component **code is not send to the browser**
 - Can safely use secure API key's etc.
 - Smaller bundle sizes
- 📌 React Server Components require TypeScript 5.1 📌

Client Component

- **Server components can render server and client components**
 - Client components can only render client components
- Adding **'use client'** to the top of a component makes it a client component

```
TS movie-form.tsx ×
src > components > TS movie-form.tsx > ...
1  'use client'
2
3  import { zodResolver } from '@hookform/resolvers/zod'
4  import * as z from 'zod'
```

Server Component

```
TS genre-loader.tsx ×
src > components > TS genre-loader.tsx > ...
You, 2 weeks ago | 1 author (You)
1 import { prisma } from '@lib/db'
2 import { GenreSelector } from '@components/genre-selector'
3
4 export async function GenreLoader() {
5   const genres = await prisma.genre.findMany({
6     orderBy: { name: 'asc' },
7   })
8
9   return <GenreSelector genres={genres} />
10 }
```

Satisfies operator

Satisfies operator

- The satisfies operator lets us **validate that the type of an expression matches some type**, without changing the resulting type of that expression.
- 💡 The goal is to **report type error where they are caused** 💡

movie-card.tsx

```
TS movie-card.tsx M x TS page.tsx M
src > components > TS movie-card.tsx > [e] MovieCard
33   return (
34     <Card className="flex flex-col">
35       <CardHeader>
36         <CardTitle>{movie.title}</CardTitle>
37         <CardDescription>
38           Vote average: {movie.vote_average} (
39             {movie.vote_count.toLocaleString()} votes)
40         </CardDescription>
41     </CardHeader>
```

movies/page.tsx



```
TS movie-card.tsx M TS page.tsx M X
src > app > movies > TS page.tsx > getMovies > genre
12  async function getMovies(
13    page: number,
14    genreId?: string,
15  ): Promise<MovieRequiredForCard[]> {
16    const select = {
17      id: true,
18      title: true,
19      overview: true,
20      backdrop_path: true,
21      vote_average: true,
22      vote_count: true,
23    } satisfies Prisma.MovieSelect
```

Prevent Overfetching

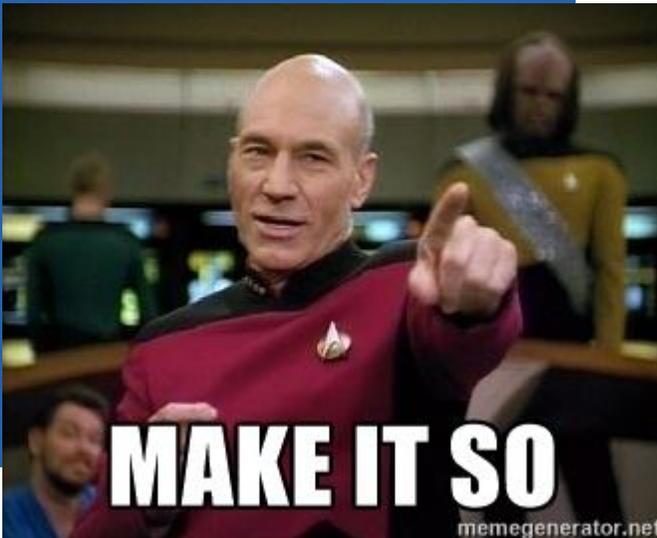
Prevent Overfetching

- Using **satisfies Prisma.MovieSelect** is not optimal
- **Allows Overfetching** of movie data
 - Making the application slightly slower than required
- This **can be prevented** with a type mapping
 - Create a **MovieSelect** like object with just the required keys

movies/page.tsx

```
TS page.tsx M X
src > app > movies > TS page.tsx > getMovies > genre > include
12  type ExactMovieSelect = Required<
13  |   Pick<Prisma.MovieSelect, keyof MovieRequiredForCard>
14  >
15
16  async function getMovies(
17  |   page: number,
18  |   genreId?: string,
19  ): Promise<MovieRequiredForCard[]> {
20  |   const select = {
21  |     id: true,
22  |     title: true,
23  |     overview: true,
24  |     backdrop_path: true,
25  |     vote_average: true,
26  |     vote_count: true,
27  |   } satisfies ExactMovieSelect
```

movies/page.tsx



```
TS page.tsx M x
src > app > movies > TS page.tsx > getMovies > genre > include
12 type PrismaSelect<TRow> = {
13   [Prop in keyof TRow]: true
14 }
15
16 type ExactMovieSelect = PrismaSelect<MovieRequiredForCard>
17
18 async function getMovies(
19   page: number,
20   genreId?: string,
21 ): Promise<MovieRequiredForCard[]> {
22   const select = {
23     id: true,
24     title: true,
25     overview: true,
26     backdrop_path: true,
27     vote_average: true,
28     vote_count: true,
29   } satisfies ExactMovieSelect
```

Break time



Photo by [MindSpace Studio](#) on [Unsplash](#)

Custom type mapping

Custom type mapping

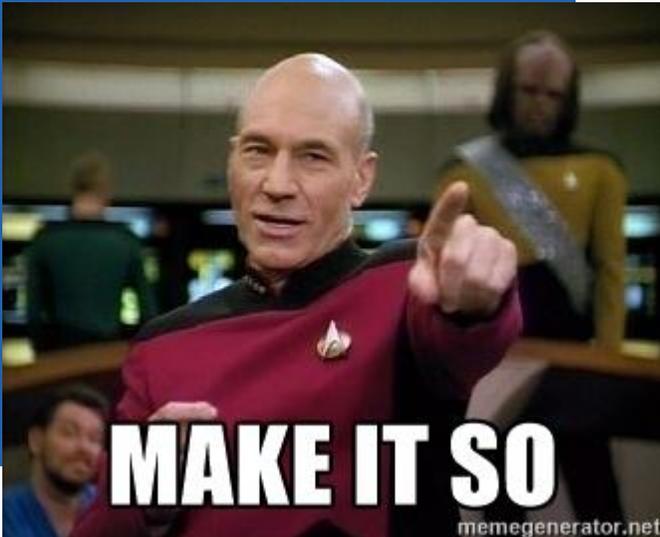
- The TypeScript type system itself is a **programming language**
 - With variables, loops, conditional logic etc.
- Example: turn Zod schema into it's TypeScript type

```
TS type-mappng.ts M x
src > lib > TS type-mappng.ts > ...
1 import { User } from 'lucide-react'
2 import { z } from 'zod'
3
4 const userSchema = z.object({
5   firstName: z.string(),
6   lastName: z.string(),
7   age: z.number().optional(),
8 })
9
10 // extract the inferred type
11 type User = z.infer<typeof userSchema>
12
13 const user: User = {
14   firstName: 'Maurice',
15   lastName: 'de Beijer',
16 }
```

Type mapping Boolean logic

```
TS type-mappng.ts 1, M X
src > lib > TS type-mappng.ts > copyOfMovie
1  import { Movie } from '@prisma/client'
2
3  type AllProps<TObject> = {
4    [Prop in keyof TObject]: TObject[Prop]
5  }
6
7  type OnlyPropsOfTypeString<TObject> = {
8    [Prop in keyof TObject as TObject[Prop] extends string
9     ? Prop
10     : never]: TObject[Prop]
11 }
12
13 type AllPropsOfAnObject<TObject extends Record<keyof any, unknown>> = {
14   [Prop in keyof TObject]: TObject[Prop]
15 }
16
17 type CopyOfMovie = AllProps<Movie>
18 type StringPartsOfMovie = OnlyPropsOfTypeString<Movie>
19 type AllPropsOfMovie = AllPropsOfAnObject<Movie>
20 type AllPropsOfString = AllPropsOfAnObject<string>
```

type-mapping.ts



```
TS type-mapping.ts M ×
src > lib > TS type-mapping.ts > ...
23 type FinalOnlyPropsOfType<
24   TObject extends Record<keyof any, unknown>,
25   Target,
26 > = {
27   [Prop in keyof TObject as TObject[Prop] extends Target
28     ? Prop
29     : never]: TObject[Prop]
30 }
31
32 type FinalStringPartsOfMovie = FinalOnlyPropsOfType<Movie, string>
33
34 const partOfMovie: FinalStringPartsOfMovie = {
35   backdrop_path: '',
36   overview: '',
37   poster_path: '',
38   release_date: '',
39   title: '',
40 }
```

The display of types

The display of types

- Type mappings sometimes lead to **hard to read types**
- **Easy to fix** with another type mapping

type-mapping.ts

```
TS type-mappng.ts X
src > lib > TS type-mappng.ts > FinalOnlyPropsOfType
26 > = {
27   [Prop in keyof TO type FinalStringPartsOfMovie = {
28     ? Prop title: string;
29     : never]: TObj overview: string;
30   } release_date: string;
31   backdrop_path: string;
32   type FinalStringPar poster_path: string; <Movie, string>
33   }
34   const partOfMovie: FinalStringPartsOfMovie = {
35     backdrop_path: '',
36     overview: '',
37     poster_path: '',
38     release_date: '',
39     title: '',
40   }
```

type-mapping.ts

```
TS type-mappng.ts X
src > lib > TS type-mapp
26 > = {
27   [Prop]
28   ?
29   :
30 }
31
32 type F
33
34 const partOfMovie: FinalStringPartsOfMovie = {
35   backdrop_path: '',
36   overview: '',
37   poster_path: '',
38   release_date: '',
39   title: '',
40 }

const partOfMovie: FinalOnlyPropsOfType<{
  id: number;
  title: string;
  overview: string;
  release_date: string;
  backdrop_path: string;
  poster_path: string;
  popularity: number;
  vote_average: number;
}> = {
  backdrop_path: '',
  overview: '',
  poster_path: '',
  release_date: '',
  title: ''
};
```

type-mapping.ts



```
TS type-mappng.ts M •
src > lib > TS type-mappng.ts > ...
32 // Taken from https://effectivetypescript.com/2022/02/25/gentips-4-display/
33 type Resolve<T> = T extends Function ? T : { [K in keyof T]: T[K] }
34
35 type FinalStringPartsOfMovie = Resolve<FinalOnlyPropsOfType<Movie, string>>
36
37
38     const partOfMovie: {
39       title: string;
40       overview: string;
41       release_date: string;
42       backdrop_path: string;
43       poster_path: string;
44     }
45 const partOfMovie: FinalStringPartsOfMovie = {
46   backdrop_path: '',
47   overview: '',
48   poster_path: '',
49   release_date: '',
50   title: '',
51 }
```

Opaque Types

Opaque Types

- A lot of business data ultimately end up as a **primitive data type**
 - They are all modeled as string, number etc.
- The **compiler doesn't know the difference** between them
 - A PO box number and invoice total amount are both type "number"
 - The same for the compiler
 - Very different for the business case
- **Opaque types can make it easier to reason about code**
 - By providing distinct types and a clear separation

Console output

```
checkout for 1234  
  Charging account 'Maurice de Beijer' for €19.90  
  Schiping movies to user Maurice de Beijer  
█  
< main ↻ 🔗 ⊗ 0 ⚠ 0 ✂ 1 🔗 Live Share
```

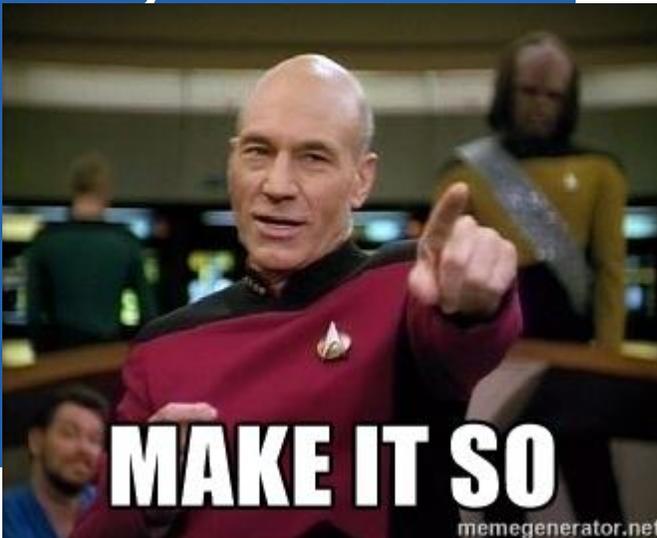
checkout- shopping- cart.ts

```
TS checkout-shopping-cart.ts M x TS checkout-dialog.tsx M
src > server > TS checkout-shopping-cart.ts > ...
3 declare const _type: unique symbol
4
5 type Opaque<A, B> = A & {
6   readonly [_type]: B
7 }
8
9 type Account = Opaque<string, 'Account'>
10
11 function isAccount(value: string): value is Account {
12   return typeof value === 'string' && value.length === 4
13 }
14
15 function assertAccount(value: string): asserts value is Account {
16   if (!isAccount(value)) {
17     throw new Error(`Expected account with 4 digits, got ${value}`)
18   }
19 }
20
21 type Amount = Opaque<number, 'Amount'>
```

checkout- shopping- cart.ts

```
TS checkout-shopping-cart.ts M x TS checkout-dialog.tsx M
src > server > TS checkout-shopping-cart.ts > ...
47 export async function checkoutShoppingCart(
48     account: string,
49     name: string,
50     amount: number,
51 ) {
52     assertAccount(account)
53     assertClientName(name)
54     assertAmount(amount)
55
56     console.group(`checkout for ${name}`)
57     chargeAccount(account, amount)
58     schipMovies(name)
59     console.groupEnd()
60 }
61
62 function chargeAccount(account: Account, amount: Amount) {
63     console.log(
64         `Charging account '${account}' for ${amount.toLocaleString('en', {
65             style: 'currency',
66             currency: 'EUR',
67         })}`,
68     )
69 }
```

checkout- dialog.tsx



```
TS checkout-shopping-cart.ts M TS checkout-dialog.tsx M X
src > components > TS checkout-dialog.tsx > CheckoutDialog
52   const onSubmit = async (data: CheckoutForm) => {
53     try {
54       await checkoutShoppingCart(data.account, data.name, totalAmount)
55       toast({
56         title: 'Success',
57         description: 'Checkout completed',
58       })
59       setCheckoutOpen(false)
60       clearCart()
61     } catch (error) {
62       const description =
63         error instanceof Error ? error.message : 'Something went wrong'
64       toast({
65         title: 'Oops',
66         description,
67         variant: 'destructive',
68       })
69     }
70   }
```

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

noUncheckedIndexedAccess

- 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
 - Whether the element is the *array element type* or *undefined*
- Try showing the **Horror** movies and observe a runtime error 😞

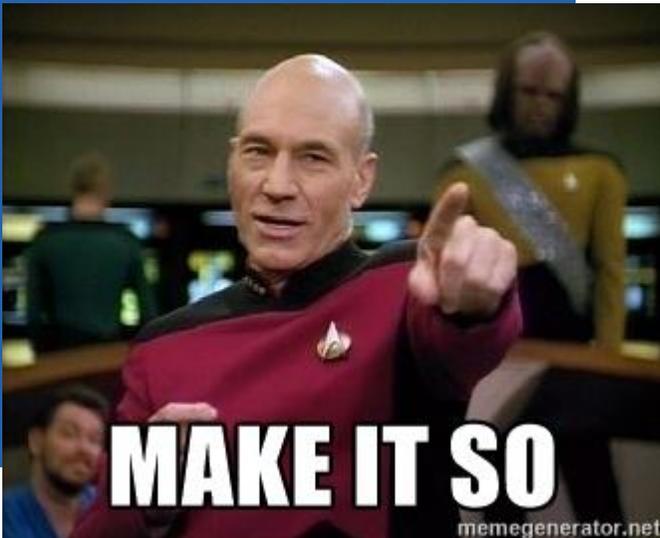
tsconfig.json

```
tsconfig.json M x TS page.tsx 1
tsconfig.json > {} compilerOptions
2   "compilerOptions": {
3     "target": "es5",
4     "lib": ["dom", "dom.iterable", "esnext"],
5     "allowJs": true,
6     "skipLibCheck": true,
7     "strict": true,
8     "noUncheckedIndexedAccess": true,
9     "noEmit": true,
```

TERMINAL PROBLEMS 1 OUTPUT DEBUG CONSOLE AZURE GITLENS COMMENTS

```
[15:36:02] File change detected. Starting incremental compilation...
src/app/movies/page.tsx:84:45 - error TS18048: 'topMovie' is possibly 'undefined'.
84     The top rated movie here is: &quot;{topMovie.title}&quot;;
                                     ~~~~~~
[15:36:02] Found 1 error. Watching for file changes.
```

movies/page.tsx



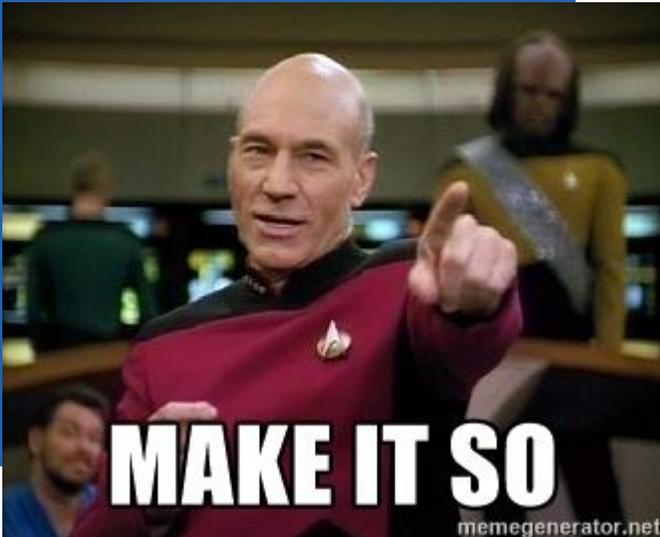
```
tsconfig.json M TS page.tsx M X
src > app > movies > TS page.tsx > MoviesPage
80     return (
81       <main className="flex-1 space-y-4 p-8 pt-6">
82         <h2 className="text-3xl font-bold tracking-tight">Movies</h2>
83         {topMovie ? (
84           <p className="p-6 text-center font-bold">
85             The top rated movie here is: &quot;{topMovie.title}&quot;;
86           </p>
87         ) : (
88           <p className="p-6 text-center font-bold">
89             There are no movies in this genre.
90           </p>
91         )}
}
```

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
- There is also a *ReadonlyArray<T>* mapped type
- 💡 Recommended for **function arguments** to show intent 💡
 - And AJAX responses etc.

movie-card.tsx



```
TS movie-card.tsx 1. M x
src > components > TS movie-card.tsx > MovieCard
18 type Props = {
19   movie: Readonly<
20     Pick<
21       Movie,
22       'id'
23       'title'
24       'overview'
25       'backdrop_path'
26       'vote_average'
27       'vote_count'
28     >
29   >
30 }
31
32 export const MovieCard = ({ movie }: Props) => {
33   const { addMovie } = useShoppingCart()
34
35   movie.title = 'This should not be allowed'
36 }

TERMINAL  PROBLEMS 1  OUTPUT  DEBUG CONSOLE  AZURE  GITLENS  COMMENTS

[16:58:53] File change detected. Starting incremental compilation...

src/components/movie-card.tsx:35:9 - error TS2540: Cannot assign to 'title' because it is a read-only property.

35   movie.title = 'This should not be allowed'
      ~~~~~

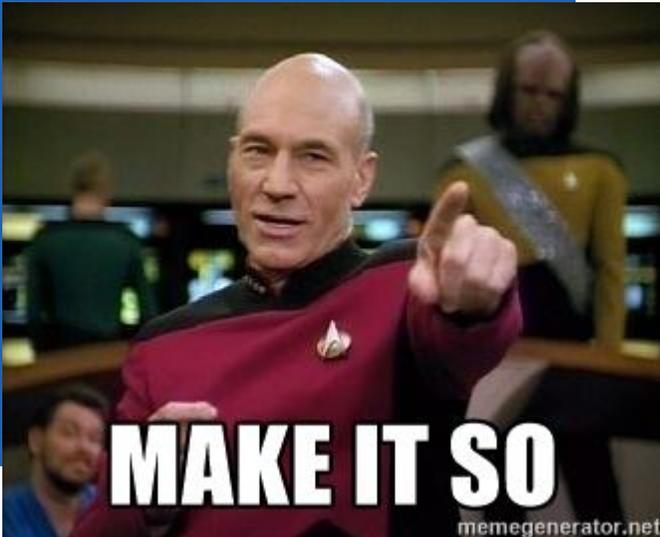
[16:58:53] Found 1 error. Watching for file changes.
```

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/1c2923f91643a16agode638e76bceoab>

movies/page.tsx



```
src > app > movies > TS page.tsx > getMovies
25   readonly [P in keyof T]: DeepReadonly<T[P]>
26 }
27
28 async function getMovies(
29   page: number,
30   genreId?: string,
31 ): Promise<DeepReadonly<MovieRequiredForCard[]>> {
32   const select = {
33     id: true,
34     title: true,
35     overview: true,
36     backdrop_path: true,
37     poster_path: true,
38     popularity: true,
39     vote_average: true,
40     vote_count: true,
41   }
42   const response = await fetch(
43     `https://api.themoviedb.org/3/movie/popular?&page=${page}&with_genres=${genreId}&select=${JSON.stringify(select)}`,
44     {
45       headers: {
46         'Authorization': `Bearer ${process.env.TMDB_TOKEN}`,
47       },
48     },
49   )
50   const json = await response.json()
51   return json.results as MovieRequiredForCard[]
52 }
53
54 movies.push({
55   backdrop_path: 'This should not be allowed',
56   id: 0,
57   overview: '',
58   title: '',
59   vote_average: 0,
60   vote_count: 0,
61 })
```

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE AZURE GITLENS COMMENTS

[17:12:58] File change detected. Starting incremental compilation...

src/app/movies/page.tsx:84:10 - error TS2339: Property 'push' does not exist on type 'readonly DeepReadonly<Readonly<Pick<{ id: number; title: string; overview: string; release_date: string; backdrop_path: string; poster_path: string; popularity: number; vote_average: number; vote_count: number; }, "id" | ... 4 more ... | "vote_count">>[]>'.
84 movies.push({

[17:12:58] Found 1 error. Watching for file changes.

Template Literal Types

Template Literal Types

- Builds on string literal types but **manipulates type definitions**
 - For example change a types snake case to camel case key names

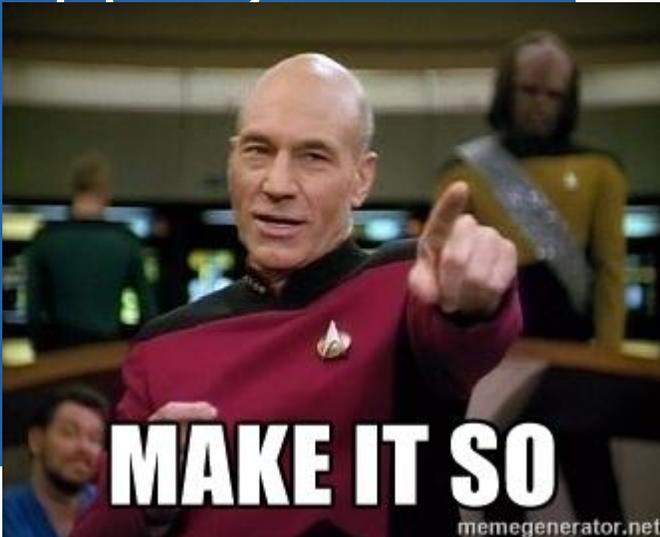
type-mapping.ts

```
TS type-mappng.ts M X
src > lib > TS type-mappng.ts > ...
47 type CamelCase<S extends string> = S extends `${infer First}_${infer Rest}`
48   ? `${Lowercase<First>}${Capitalize<CamelCase<Rest>>}`
49   : Lowercase<S>
50
51 type CamelCaseProps<TObject extends Record<keyof any, unknown>> = {
52   [Prop in keyof TObject as CamelCase<Prop & string>]: TObject[Prop]
53 }
54
55 type CamelCaseMovie = CamelCaseProps<Movie & { this_movie_is_popular: boolean }>
```

type-mapping.ts

```
TS type-mappng.ts M X
src > lib > TS type-mappng.ts > ...
51 type CamelCaseProps<TObject extends Record<keyof any, unknown>> = {
52   [Prop in keyof TObject as CamelCase<Prop & string>]: TObject[Prop]
53 }
54
55 type CamelCaseMovie = CamelCaseProps<Movie & { this_movie_is_popular: boolean }>
56   overview: string;
   releaseDate: string;
   backdropPath: string;
   posterPath: string;
   popularity: number;
   voteAverage: number;
   voteCount: number;
   thisMovieIsPopular: boolean;
}
```

type- mapping.ts



```
TS type-mappng.ts M X
src > lib > TS type-mappng.ts > ...
57 type Names = 'foo' | 'bar'
58 type Suffixes = 'Key' | 'Value'
59
60 type Pairs = `${Names}${Suffixes}`
61
```

'Pairs' is declared but never used. ts(6196)

△ Error(TS6196) [🔗] | [🌐]

Pairs is declared but never used.

```
type Pairs = "fooKey" | "fooValue" | "barKey" | "barValue"
```

Quick Fix... (Ctrl+.)

Typing JSON files

Typing JSON files

- **IntelliSense will parse an imported JSON file** to determine the types
 - This can be very slow with large files
- Add a **<filename>.d.json.ts** to explicitly type the imported data
 - ⚠ Will not validate that the JSON actually has the correct shape ⚠

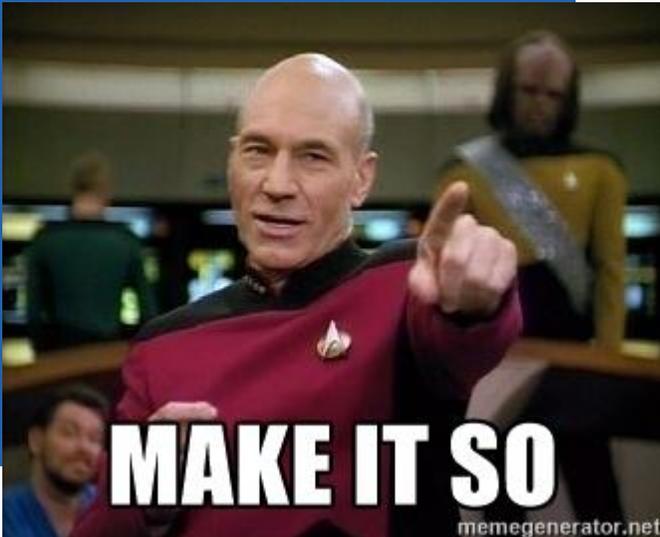
genres.d.json.ts

```
TS genres.d.json.ts U x TS tsconfig.json M TS seed
prisma > TS genres.d.json.ts > ...
1  type Genre = {
2     id: number
3     name: string
4     type?: string
5  }
6
7  const genres: Genre[]
8
9  export default genres
```

tsconfig.json

```
TS genres.d.json.ts U  TS tsconfig.json M X  TS seed.ts
TS tsconfig.json > ...
2  "compilerOptions": {
3    "target": "es5",
4    "lib": ["dom", "dom.iterable", "esnext"],
5    "allowJs": true,
6    "allowArbitraryExtensions": true,
7    "skipLibCheck": true,
```

seed.ts



```
TS genres.d.json.ts U  TS tsconfig.json M  TS seed.ts X
prisma > TS seed.ts > main
1  import { PrismaClient } from '@prisma/client'
2
3  const p (alias) const genres: Genre[]
4  import import genres
5  import genres from './genres.json'
```

Conclusion

- **Compiling** the code to catch type errors early
 - Compile and Test your code with each pull request
- Using the *satisfies* operator
- Use **type mappings** to optimize your code
 - Prevent over fetching of data with Prisma
- **Custom type mappings** are really powerful
 - Can help detect all sorts of errors quickly
- Make mapped types more **readable**
 - Using another type mapping 😊
- **Opaque Types** can help with type safety
 - Not all strings represent the same data but for TypeScript a string is a string
- Use **more strict features** beyond the basics
 - Using `noUncheckedIndexedAccess` can help catch potential runtime errors
- Using **Readonly<>** and **ReadonlyArray<>**
 - Prevent objects from being mutated by accident
 - Custom type mapping `DeepReadonly<>`
- Use **Template Literal Types** to manipulate types
- **Typing JSON files** can help performance when importing large JSON files

Maurice de Beijer

[@mauricedb](#)

maurice.de.beijer@gmail.com

