Training TypeScript

Module: Common TypeScript Mistakes

ORBIT GT BELGIUM

Peter Kassenaar

info@kassenaar.com

Common mistakes in TypeScript code

1. Use `unknown` over `any`

```
namespace any_unknown{

    // Let's assume we use any and unknown for constants, like so:
    const numberAny: any = 10;
    const numberUnknown: unknown = 10;

    let msg1 : string = numberAny; // OK,
    let msg2 : string = numberUnknown; // ERROR,

    // same goes with methods:
    // - 'any' will just assume that a given method exists on a variable
    // - 'unknown' will assume a given method DOES NOT exsist:
    numberAny.someMethod(); // OK
    numberUnknown.someMethod(); // ERROR,

    // *******************
    // LESSON: Preferably use 'unknown' over 'any'!
    // *******************

}
```

../02a-any-unknown.ts

2. Use Type Inference where possible

- In TypeScript we can have implicit and explicit types.

Opinion:

"Ideally, we should always avoid adding types where they can be inferred. Redundant type annotations clutter our code which makes it harder to read. It also makes refactoring more painful"

https://medium.com/geekculture/typescript-advantages-common-mistakes-to-avoid-13ae5395dcc2

Using Type Inference

```
// Opinion: 'use type inference where possible'

    // 1. NO type information in this array, but its type can
    // furher on be inferred.
    const studentArray = [{
        name: 'Peter',
        gender: 'M'
    }, …]

    // with explicit any, this works. However, we need 'any' (since we don't have
    // a Student type here) while its type can be inferred.
    const firstStudent: any = {...studentArray[0], name: 'Johan'};
    firstStudent.rank = 3;
    // firstStudent.age = 10; // OK, even if we don't WANT an 'age' field on this type.
    studentArray[0] = firstStudent;

    // With implicit type: type is inferred (the compiler KNOWS every
    // student has a 'name' and a 'gender'
    const firstStudentSpread = {...studentArray[0], rank: 1}
    // firstStudentSpread.age=10; // INVALID, because of the inferred type
    studentArray[0] = firstStudentSpread;

    console.log(studentArray[0]);
```

../01b-more-type-inference.ts

3. Don’t use wrapper types

- E.g.: use `string` instead of `String` and `number` instead of `Number` as a type.

“String and string are not equivalent. Typescript suggests a proper solution. We should always avoid those uppercase types (wrapper objects) because they are just a Javascript-specific way to provide some methods on primitives. We usually don’t need and shouldn’t use them directly.”

../04a-wrapper-objects.ts

4. D.R.Y. Don’t Repeat Yourself

- If you have similar (but not the same) types, compose them, instead of redefining them.

```
//**************** WRONG
interface Customer {
    id: number,
    name: string,
    address: {
        city: string,
        state: string
    }
}
interface Address {
    city: string;
    state: string;
}
interface BankCustomer {
    id: number,
    name: string,
    address: {
        city: string,
        state: string
    },
    branchName: string,
    accountNo: number
}
```

```
//**************** RIGHT
interface Customer2 {
    id: number,
    name: string,
    address: Address
}
interface Address2 {
    city: string;
    state: string;
}
interface BankCustomer2 extends Customer2 {
    branchName: string,
    accountNo: number
}
```

5. Using strict mode

- Alway use `strict` mode for TypeScript.
- Set it in `tsconfig.json`, or use the `--strict` flag on commandline.
- Separate strict checks parameters are:
  - `noImplicitAny`
  - `strictNullChecks`
  - `strictFunctionTypes`
  - `strictBindCallApply`
  - `strictPropertyInitialization`
  - `noImplicitThis`
  - `alwaysStrict`

```
{
  "compilerOptions": {
    …,
    "strict": true,
    …
  }
}
```

When to use strict mode

- In new projects: always!
- In existing projects: it depends
  - If you already have a bunch of files AND enough time to refactor, it is worth the effort
  - Otherwise: start from now on, or use separate `strict` parameters
- More info: https://maxkovalevsky.com/what-is-strict-mode-in-typescript-and-why-and-when-you-should-use-it/
- Workshop: read this blogpost!

What Is Strict Mode In TypeScript, Why And When You Should Use It?

16/07/2021

#javascript #typescript #typescriptbook

(alternative: https://dev.to/briwa/how-strict-is-typescript-s-strict-mode-311a)

6. Mistake: using `type assertions` instead of `type declarations`

- Consider the following example:

```
type Employee = {
    name: string;
    gender: string;
}

// There is a HUGE difference in these two statements:
const Peter: Employee = {name: 'Peter', gender: 'M'};
const Sandra = {name: 'Sandra', gender: 'F'} as Employee;
```

First example: Type declaration – typechecks beforehand

Second example: Type assesrtion – typechecks afterwards

Also see `../04-type-assertions.ts`

7. Not using lookup types

- If you have a complex type, you can create a *lookup type* that uses parts of the original type, instead of creating a new type:

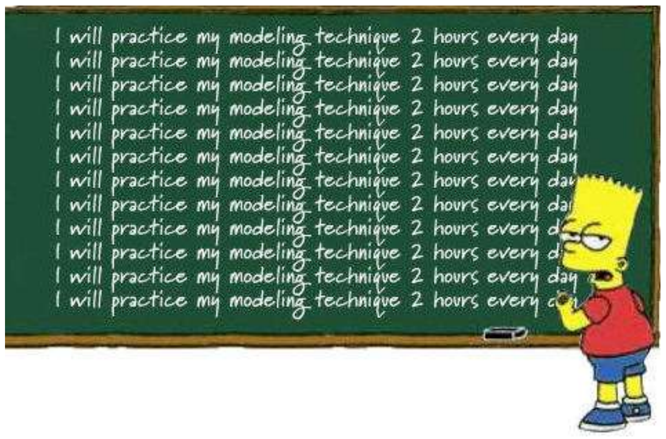
```
type Employee = {
    info: {
        name: string;
        age: number;
        gender: string;
    },
    company:{
        name: string;
        department: string;
        city:string;
    }
    // ... more properties
}

type empInfo = Employee['info'];
type empCompany = Employee['company'];
```

This looks a bit like `Pick<T, K>`, but doesn't use a Union Type

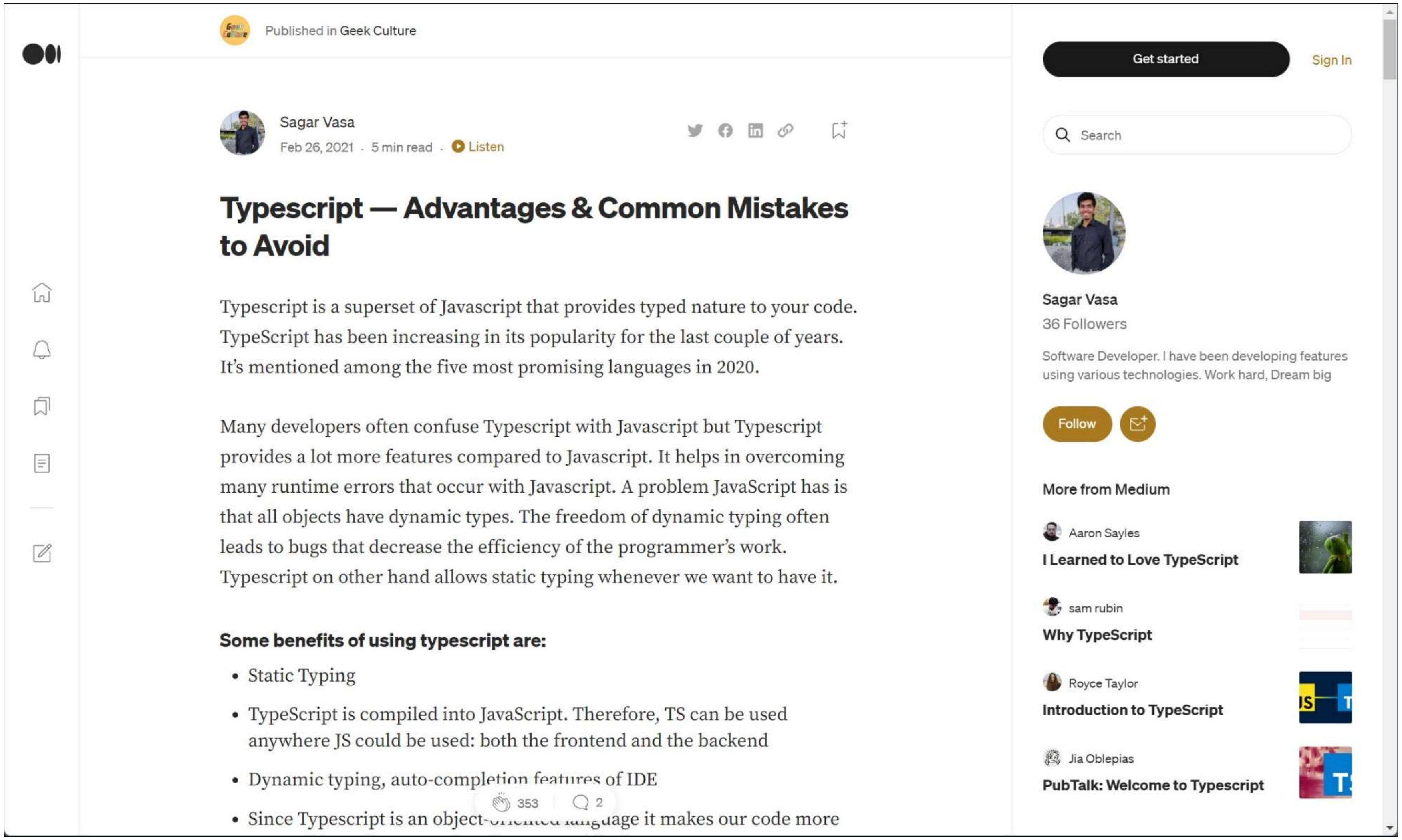
`../28-lookup-types.ts`

Workshop

- Check the different files / examples and see the workshops in:
  - `01b-more-type-inference.ts`
  - `02a-any-unknown.ts`
  - `04a-wrapper-objects.ts`
  - `04b-type-composition.ts`
  - `04c-type-assertion.ts`
  - `28-lookup-types.ts`
- Search Google for "`TypeScript mistakes`" and see if you find more information.
- Read these blogs carefully, weigh their validity and try to address the issues in your own code!



More information, for instance:



Sagar Vasa
Feb 26, 2021 · 5 min read · Listen

Typescript — Advantages & Common Mistakes to Avoid

Typescript is a superset of Javascript that provides typed nature to your code. TypeScript has been increasing in its popularity for the last couple of years. It's mentioned among the five most promising languages in 2020.

Many developers often confuse Typescript with Javascript but Typescript provides a lot more features compared to Javascript. It helps in overcoming many runtime errors that occur with Javascript. A problem JavaScript has is that all objects have dynamic types. The freedom of dynamic typing often leads to bugs that decrease the efficiency of the programmer's work. Typescript on other hand allows static typing whenever we want to have it.

Some benefits of using typescript are:

- Static Typing
- TypeScript is compiled into JavaScript. Therefore, TS can be used anywhere JS could be used: both the frontend and the backend
- Dynamic typing, auto-completion features of IDE
- Since Typescript is an object-oriented language it makes our code more

353 2

Sagar Vasa
36 Followers
Software Developer. I have been developing features using various technologies. Work hard, Dream big

More from Medium

Aaron Sayles
I Learned to Love TypeScript

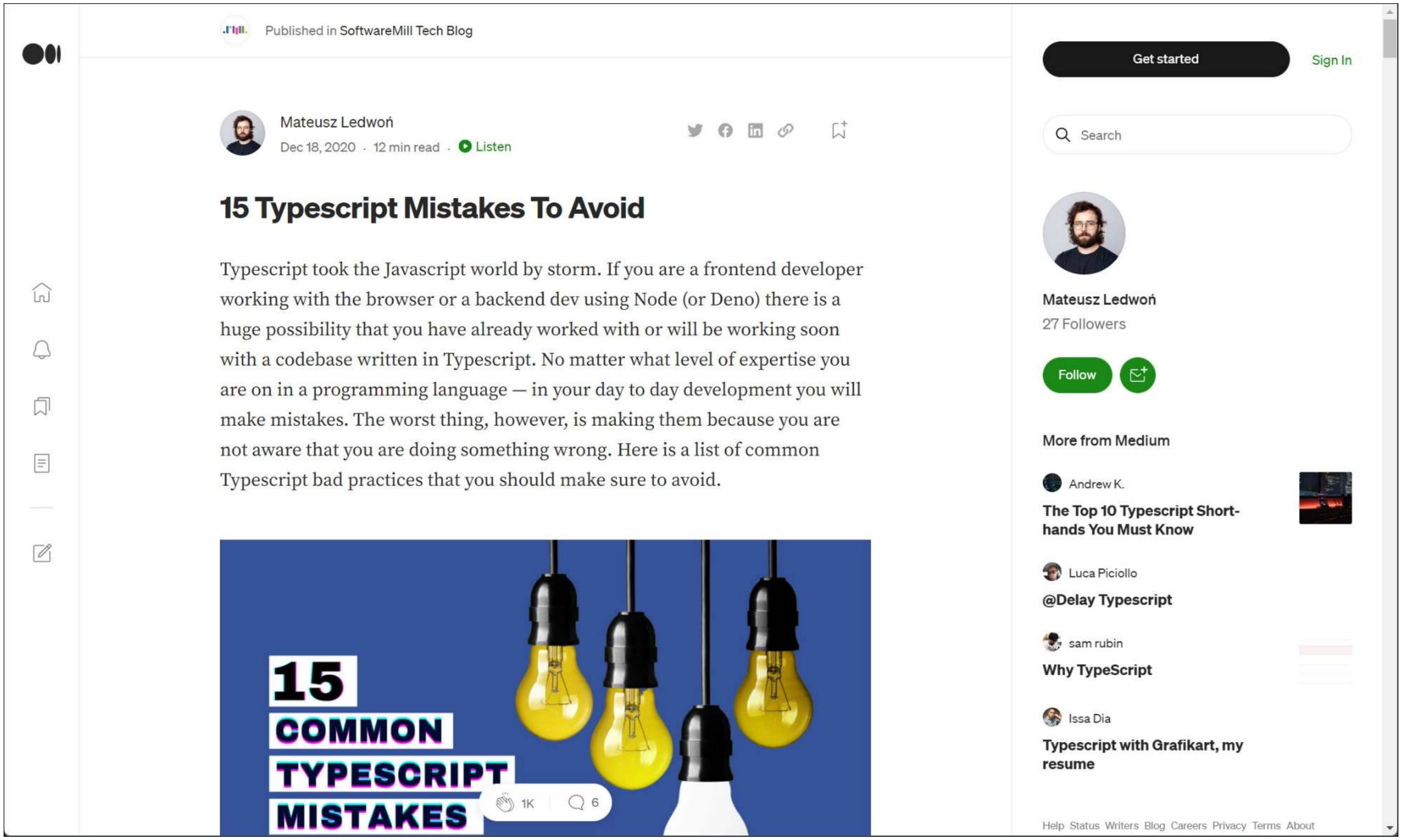
sam rubin
Why TypeScript

Royce Taylor
Introduction to TypeScript

Jia Oblepias
PubTalk: Welcome to Typescript

https://medium.com/geekculture/typescript-advantages-common-mistakes-to-avoid-13ae5395dcc2

SoftwareMill Tech Blog


Published in SoftwareMill Tech Blog

Mateusz Ledwoń
Dec 18, 2020 · 12 min read · Listen

15 Typescript Mistakes To Avoid

Typescript took the Javascript world by storm. If you are a frontend developer working with the browser or a backend dev using Node (or Deno) there is a huge possibility that you have already worked with or will be working soon with a codebase written in Typescript. No matter what level of expertise you are on in a programming language — in your day to day development you will make mistakes. The worst thing, however, is making them because you are not aware that you are doing something wrong. Here is a list of common Typescript bad practices that you should make sure to avoid.

Mateusz Ledwoń
27 Followers

More from Medium

Andrew K.
The Top 10 Typescript Short-hands You Must Know

Luca Piciollo
@Delay Typescript

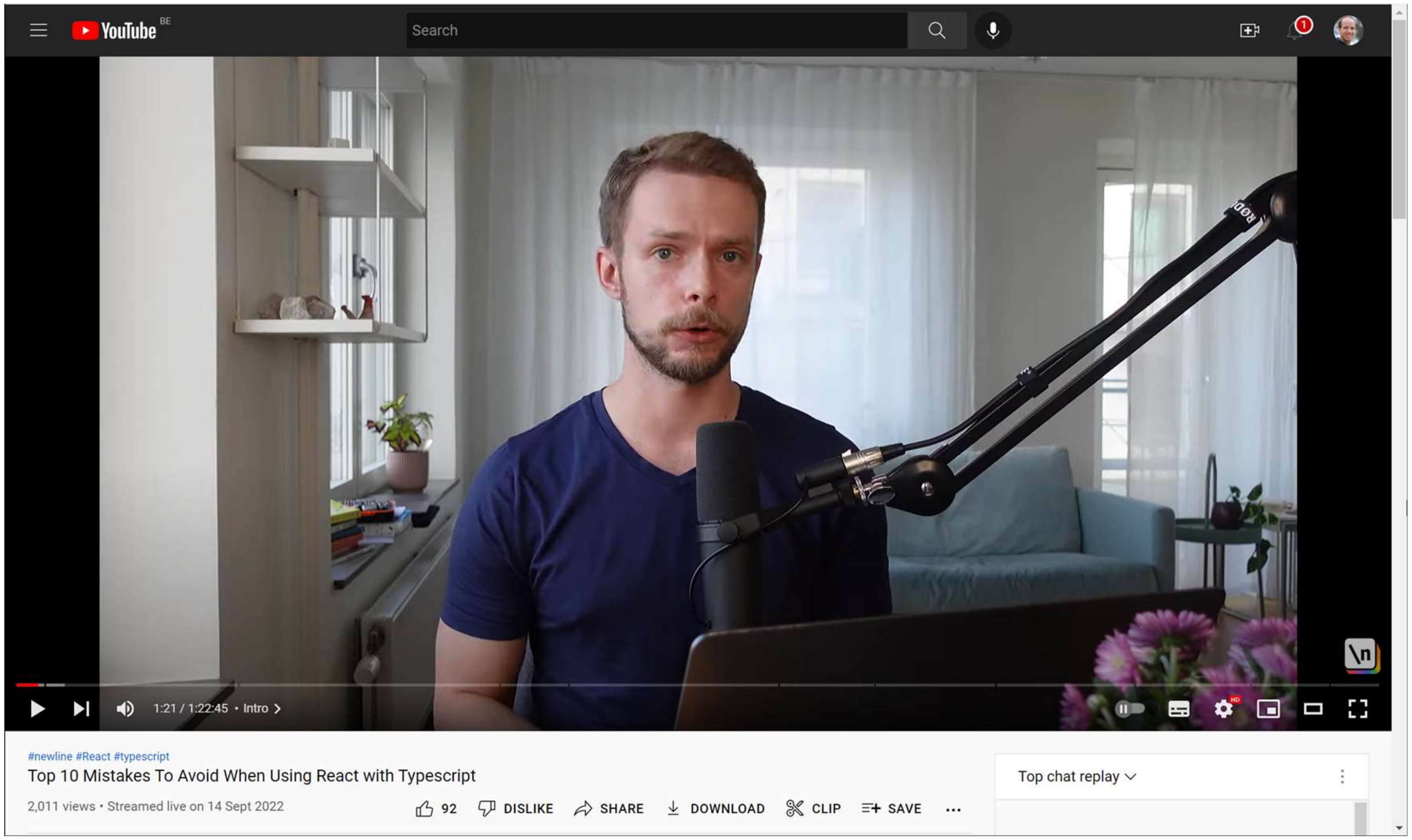
sam rubin
Why TypeScript

Issa Dia
Typescript with Grafikart, my resume

https://blog.softwaremill.com/typescript-mistakes-to-avoid-d3ab240c90eb

YouTube - Maksim Ivanov (Mojang, Spotify)

Good quality, but quite s.l.o.w.



YouTube BE
Search
1:21 / 1:22:45 • Intro
#newline #React #typescript
Top 10 Mistakes To Avoid When Using React with Typescript
2,011 views • Streamed live on 14 Sept 2022
92
DISLIKE
SHARE
DOWNLOAD
CLIP
SAVE
Top chat replay

https://www.youtube.com/watch?v=biYcEm42vOI