YOU DON'T KNOW TYPESCRIPT

https://przemyslawjanpietrzak.github.io/przemyslawjanpietrzak.github.io/you-dont-know-ts/dist

Espeo
Software

PART I
Migration

DEMO (MIGRACJA)

package.json

```
"type-check": "tsc bundle.js --allowJs --noEmit",
"build": "npm run lint && npm test && npm run type-check && npm run bundle"
```

MIGRATE

```
for f in src/**/*.js; do
  git mv "$f" "${f%.js}.ts"
done
```

PART II
Compiler options

Dead code elimination

```
"noUnusedParameters": true,
"noUnusedLocals": true,

const fn = (_unusedArg) => 42 // OK
```

"strictFunctionTypes": true,

```
const fn = (arg: number, arg1: (string) => string) => {}

fn(42, str => str / 2); // ERROR
```

"noImpicitAny": true,

```
// WRONG
const fn = (arg) => arg;

// GOOD
const fn1 = (arg: any) => arg;

// GOOD
const fn2 = (arg: number) => arg;

// ALSO GOOD
[1,2,3].map(item => item + 1);
```

"strictNullChecks": true,

```
document.querySelector('#id').getAttribute('class') // ERROR;

(document.querySelector('#id') as HTMLElement).getAttribute('class')

const element = document.querySelector('#id');
if (element !== null) {
  element.getAttribute('#id);
}
```

PART III
TSLint & Sonar

no-unsafe-any: {“severity”: “warning”}

```
const fn = (arg) => 42; // ERROR
let arr = []; // ERROR
let scoped; // ERROR
```

```
if [ $(npm run lint | grep WARNING | wc -l) -gt 100 ]; then exit 1; fi
```

noImplicitReturns": true

```
const fn = () => {
    if (true) {
        return; // ERROR: Not all code paths return a value.
    }
    if (false) {
        return 42;
    }
}
```

no-commented-code: true

```
// const arg = fn(42); ERROR
```

Make code simple again

“parameters-max-number”: [true, 10]
“cognitive-complexity”: [true, 10]
“no-big-function”: [true, 42]

```
// tslint:disable-next-line:cognitive-complexity
public complexMethod() {
```

"no-inferrable-types": true

```
@Output() onChange = new EventEmitter(); // Explicit type parameter needs to be provided to the constructor
@Output() onChange = new EventEmitter<number>(); // OK
@Output() onChange = new EventEmitter<any>(); // also OK
```

PART IV
Tricks

Property Accessing

```
interface Data {
  field: {
    name: string;
  }
}

export const fn = (arg: Data['field']) => {
  return arg.name; // { name: string }
}

export const fn1 = (name: Data['field']['name']) => {
  return name; // string
}
```

Big integer

```
const bigNumber = 123_456_789;
```

readonly & abstract

```
abstract class AbstractService {
  public method() {}
}

class Service extends AbstractService {
  public readonly field = [42];
}

const service = new Service();
service.field.push(42); // OK
service.field = [43]; // ERROR

const abstractService = new AbstractService(); // ERROR
```

Tuple and dict

```
const fn = (arg: { [key: string]: number }) => {
  const val =  arg.key1 + arg.key2 + arg.key3; // number
  const val1 = arg.totallyRandomKey; // number
  const val2 = arg['wpłynąłem na suchego przestwór oceanu']; // number
};

let tuple: [string, number];
tuple = ["hello", 10]; // OK
tuple = [10, "hello"]; // Error
let str = tuple[0]; // string
let num = tuple[1]; // number
```

Ampersand operator

```
const fn = (arg: { key: string } & { key1: number }) => 42;

fn({ key: '42' }); // ERROR
fn({ key1: 42 }); // ERROR
fn({ key: '42', key1: 42 }); // GOOD

type tableRow = Item & { selected?: boolean };
```

Optional types

```
interface Data {
  fn(arg: string): Array<string>
  fn(arg: number): null
}

let data: Data;
const a = data.fn(42); // null
const b = data.fn("str"); // Array<string>
```

Optional types #2

```
export interface API {
    "/users": { params: [], response: IUser[]}
    "/user/:id": { params: [number], response: IUser}
  }
```

Optional types #3

```
type If<A, T, U> = A extends true ? T : U;

let a: If<true, string, number>; // string
let b: If<false, string, number>; // number
```

Maped types

```
export type DeepReadonlyObject<A> = { readonly [K in keyof A]: DeepReadonly<A[K]> };
type DeepReadonlyObject<A> = { readonly [K in keyof A]: DeepReadonly<A[K]> }

type X = DeepReadonlyObject<{ key: string, key1: number }>; // { readonly key: any; readonly key1: any; }
```

Optional maped types

```
export type Omit<A extends object, K extends string | number | symbol> = Pick<A, Exclude<keyof A, K>>

type X = Omit<{ key: string, key1: string }, "key"> //  { key1: string; }

type Diff<A extends object, K extends keyof A> = Omit<A, K> & Partial<Pick<A, K>>
```

Grande finale

```
type ZeroTuple = [];
type PrependTuple<A, T> = T extends Array<any>
  ? (((a: A, ...b: T) => void) extends (...a: infer I) => void ? I : [])
  : [];
type TupleLength<T extends Array<any>> = T["length"];

type NumberToTuple<N extends number, L extends Array<any> = ZeroTuple> = {
  true: L;
  false: NumberToTuple<N, PrependTuple<1, L>>;
}[TupleLength<L> extends N ? "true" : "false"];

type Increment<N extends number> = TupleLength<PrependTuple<1, NumberToTuple<N>>>;

type T = Increment<42>
```

Btw We're hiring!

#chopokodzic

espeo.eu

Links

- Sonar to TS
- TS type utils
- Author of code from last slide
- compiler config
- linter config

Thank you :*