NILS HARTMANN | HTTPS://NILSHARTMANN.NET

Let's type!

A practical introduction to TypeScript

Slides: http://bit.ly/voxxed-vienna-typescript

NILS HARTMANN

Software Developer from Hamburg

Java
JavaScript, TypeScript, React

Trainings, Workshops

nils@nilshartmann.net
https://github.com/nilshartmann

"TypeScript is probably the most important language right now (...)

TypeScript makes JavaScript twice as good, and that's a conservative estimate (...)

In terms of impact, TypeScript is the most important thing right now possibly."

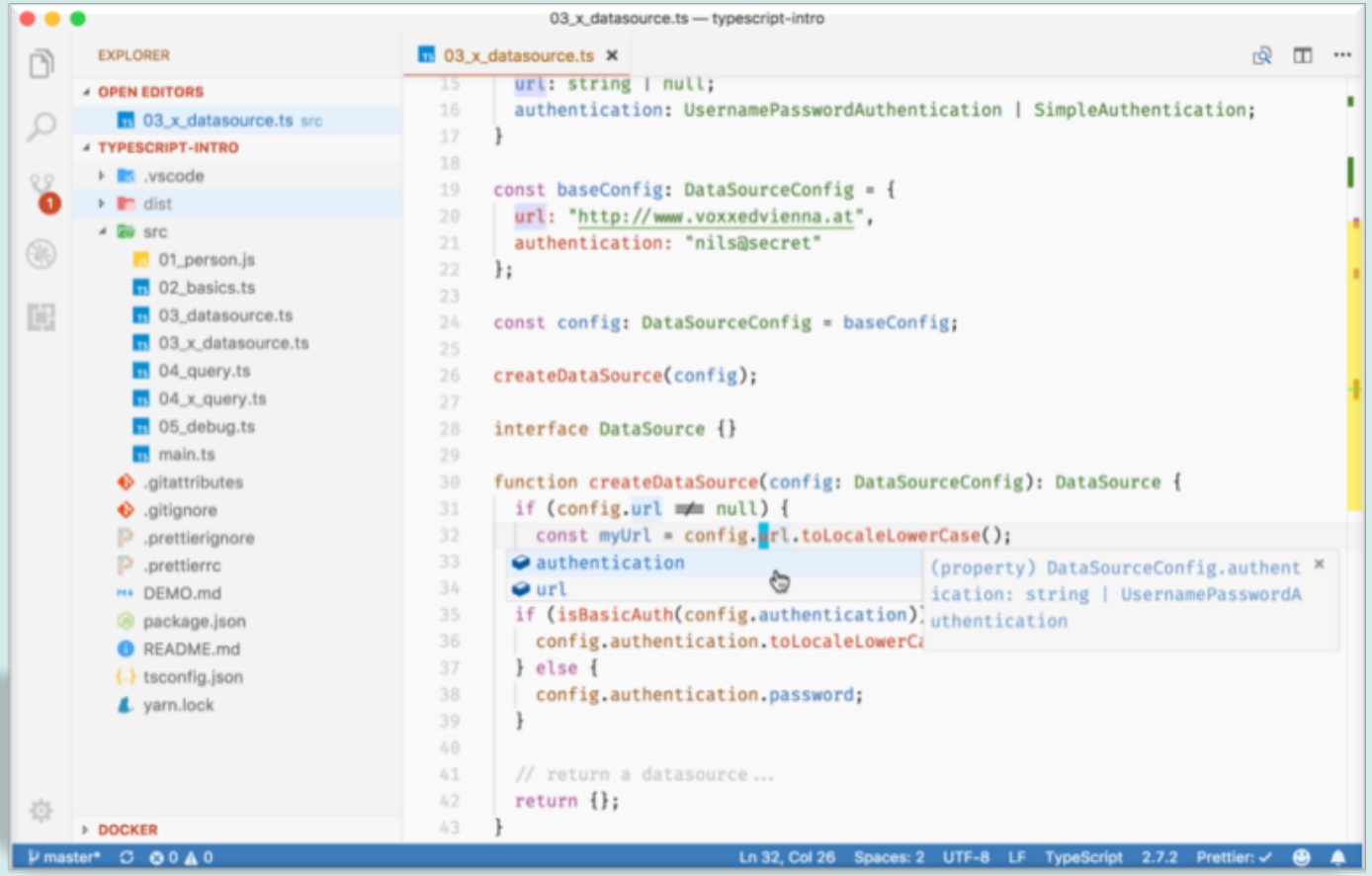
-- Rod Johnson (Creator of Spring Framework)
(https://thenewstack.io/spring-rod-johnson-enterprise-java/)

TypeScript

TYPESCRIPT AT A GLANCE

TypeScript: Superset of JavaScript

- Every JavaScript Code is valid TypeScript code (should be...)
- Adds Type Annotations, Visibility, Enums and Decorators
- Compiler compiles TypeScript- into JavaScript (ES3, ES5, ES6)-Code
- Build by Microsoft
  - http://www.typescriptlang.org/



Demo

PRACTICAL INTRODUCTION!

Syntax

TYPE ANNOTATIONS

Using Types

Variables

```
let foo: string; // built-in types, for example: string, number, boolean
```

TYPE ANNOTATIONS

Using Types

Variables

```
let foo: string; // built-in types, for example: string, number, boolean
```

Functions

```
function sayIt(what: string) {
   return `Saying: ${what}`;
}
```

TYPE ANNOTATIONS

Using Types

Variables

```
let foo: string; // built-in types, for example: string, number, boolean
```

Functions

```
function sayIt(what: string) {
   return `Saying: ${what}`;
}
```

Specifying Types is optional, Types will then be inferred:

```
let result = 7; inferred Type: number
result = sayIt('Lars') // Error (inferred type of sayIt: string)
```

TYPE ANNOTATIONS

any Type

```
let foo: any; // allow usage of all types, no typechecking

foo = "Klaus"; // OK
foo = 7; // OK
foo = null; // OK
```

In "strict mode" TypeScript will never assign any to a type (but you can use it explicitly)

UNION TYPES

A *Union Type* indicates that a value can be one of *several types*

```
let foo: string | number;

foo = 7; // OK
foo = "Seven"; // also OK

foo = false; // ERROR
```

NULL AND UNDEFINED

null and *undefined* are own types in TypeScript

- Types are not nullable and cannot be undefined (with "strictNullChecks")

```
let a: string = "Klaus";
a = null; // Error
```

Use union type to allow null:

```
let a: string | null = "Klaus;
a = null; // OK
```

Same for undefined:

```
let a: string | undefined;
```

String Literal Type

- You can define what actual value a string might have

```
type Language = "Java" | "TypeScript";

const java:Language = "Java"; // OK
const cpp:Language = "C++"; // ERROR
```

Defining own Types – interfaces define Shape of an Object

```
interface Person {              // alternative: type
  firstName: string,
  lastName: string|null,        // nullable Type ("a String or null")
  age?: number                  // optional type (might be undefined)
}
```

Defining own Types - Usage

```
interface Person {            // alternative: type
  firstName: string,
  lastName: string|null,      // nullable Type ("a String or null")
  age?: number                // optional type (might be undefined)
}

function sayHello(p: Person) {
  console.log(`Hello, ${p.lastName}`);
  p.lastName.toUpperCase(); // Error: Object is possibly null
}

sayHello({firstName: 'Klaus', lastName: null}); // OK
sayHello({firstName: 'Klaus', lastName: 777}); // Error: lastName not a string
sayHello({firstName: 'Klaus', lastName: 'Mueller', age: 32}); // OK
```

STRUCTURAL IDENTITY

TypeScript uses *Structural Identity* for Types

```
interface Person {
  name: string
}

interface Animal {
  name: string
}

// create a Person
const p:Person = { name: 'Klaus' };

// assign the Person to an Animal
const a:Animal = p; // OK as Person and Animal have same structure
                    // (would not work in Java/C#)
```

Class Syntax same as ES6 but with visibility

```
class Person {
  private name: string

  constructor(name: string) {
    this.name = name;
  }
}

const p = new Person("Klaus");
console.log(p.name); // ERROR
```

Generics

```
interface Person { name: string };
interface Movie { title: string };

let persons:Array<Person> = [];
let movies:Array<Movie> = [];

persons.push({name: 'Klaus'});       // OK
movies.push({title: 'Batman'});      // OK
persons.push({title: 'Casablanca'}) // error ('title' not in Person)
```

TYPE CHECKING JAVASCRIPT CODE

You can enable type checking even in a JS file!

- Use the *ts-check directive* at the beginning of a file
- You can add type information using JSDoc syntax

```
// @ts-check

/**
 * @param {string} name  The name
 * @param {number} age The age
 */
function newPerson(name, age) {
  name.toLowerCase(); // OK
  age.toLowerCase(); // ERROR Property 'toLowerCase' does not exist on type 'number'
}
```

Thank you!

Slides: http://bit.ly/voxxed-vienna-typescript

Questions?