

# Apps modulares en Angular 1.X con TypeScript

Jorge Mario Lenis

CTO Kerberos Ingeniería S.A.S



@mlenislibreros



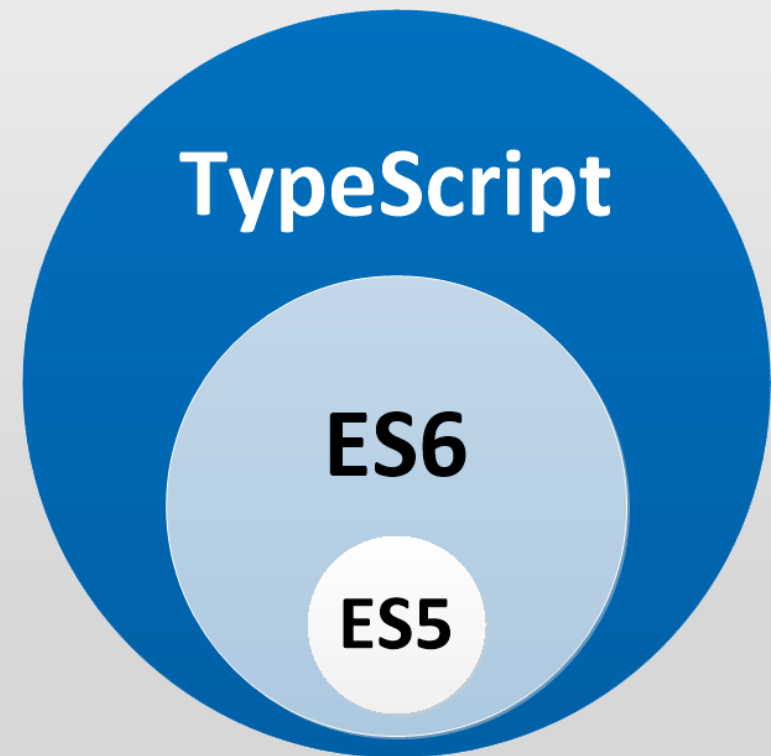
mariolenis

# Agenda

1. Qué es TypeScript.
2. Qué es Angular y por qué usar aún Angular 1.X.
3. Por qué usar TypeScript.
4. Cómo luce?
5. Modularizando apps.
6. Entorno de desarrollo.
7. Código (yeah!).
8. Mitos de TypeScript para apps Modulares.

# Qué es TypeScript?

1. Lenguaje POO creado por Microsoft que añade tipado a las variables y funciones extendidas a JavaScript.
2. Superset de JavaScript.
3. TypeScript cuenta con:
  - Módulos (ES6) => { 'ES2015' }
  - Clases (ES6) => { 'ES2015' }
  - Tipos.
  - Anotaciones.
  - Interfaces.
  - Enumeraciones.
4. Permite mezclar JavaScript + TS
5. Compila a ES5 - (ES6) ES2015



# Qué es Angular 1.X

AngularJS es un framework para Javascript mantenido por Google para el desarrollo de apps web e híbridas en móviles.

## Key Factors!

1. One way binding + Two way binding.
2. Framework enfocado en directivas.
3. Controladores, Servicios, factorías, proveedores, constantes, etc.
4. Single Page Applications (SPA)

# Por qué usar Angular 1.X

1. Es estable, robusta, muy bien documentada y muy usada en el entorno laboral.
2. El mercado ya hizo sus desarrollos en Angular 1.X
3. Existen toneladas de ejemplos, preguntas y respuestas en internet.
4. Construir aplicaciones rápidas para MVPs o de escala media es más sencillo.
5. Escalar una solución debería ser más fácil. (MVW = Model View Whatever).

# Por qué usar TypeScript

1. Programar en TypeScript es como programar en ES2015 con tipado de variables.
2. Este lenguaje te permite crear aplicaciones modulares para grandes (largas) aplicaciones.
3. Arquitectura en módulos que hará más fácil la colaboración en grupos de desarrollo de +1 dev.
4. Estas en modo Need for Speed? Chequear el tipado de variables en pocas líneas.

{

```
public miCadena: string;  
private miNumero: number;
```

{

```
import servicio from './servicios'  
import * as ctrlrs from './controllers'
```

{

```
App  
|- /Controller.ts  
|- /Servicios  
    |- /LoginService.ts
```

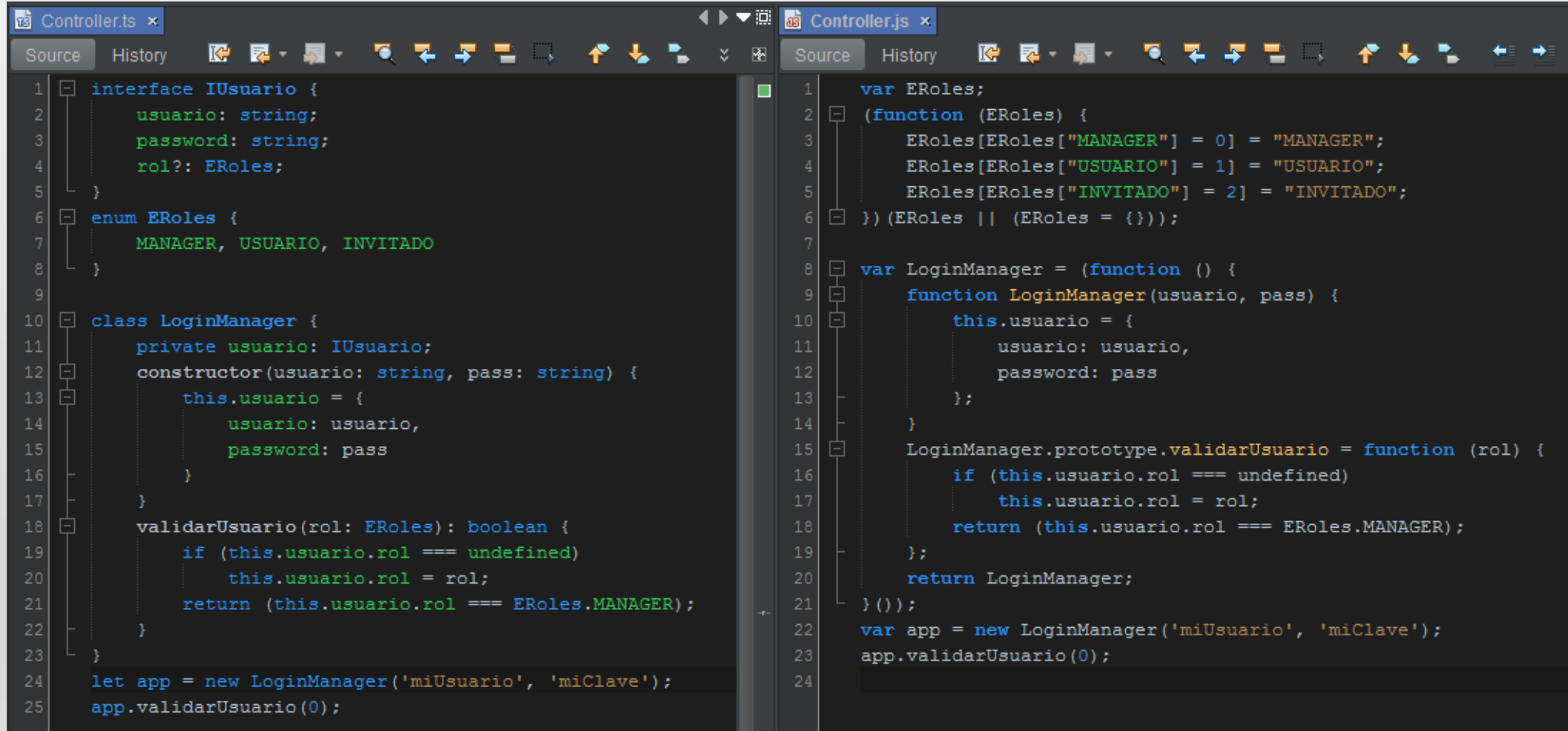
{

```
Argument of type 'string' is not assignable to parameter of type 'number'.  
-----  
(Alt-Enter shows hints)  
app.validarUsuario('manager');
```

# Cómo luce la compilación

TypeScript

ES5



The image shows a side-by-side comparison of TypeScript and ES5 code for a login manager. The left pane, titled 'Controller.ts', shows the TypeScript source code. The right pane, titled 'Controller.js', shows the compiled ES5 JavaScript code. The TypeScript code uses interfaces, enums, and classes, while the ES5 code uses plain JavaScript with functions and objects.

```
1 interface IUserario {
2     usuario: string;
3     password: string;
4     rol?: ERoles;
5 }
6 enum ERoles {
7     MANAGER, USUARIO, INVITADO
8 }
9
10 class LoginManager {
11     private usuario: IUserario;
12     constructor(usuario: string, pass: string) {
13         this.usuario = {
14             usuario: usuario,
15             password: pass
16         }
17     }
18     validarUsuario(rol: ERoles): boolean {
19         if (this.usuario.rol === undefined)
20             this.usuario.rol = rol;
21         return (this.usuario.rol === ERoles.MANAGER);
22     }
23 }
24 let app = new LoginManager('miUsuario', 'miClave');
25 app.validarUsuario(0);
```

```
1 var ERoles;
2 (function (ERoles) {
3     ERoles[ERoles["MANAGER"] = 0] = "MANAGER";
4     ERoles[ERoles["USUARIO"] = 1] = "USUARIO";
5     ERoles[ERoles["INVITADO"] = 2] = "INVITADO";
6 })(ERoles || (ERoles = {}));
7
8 var LoginManager = (function () {
9     function LoginManager(usuario, pass) {
10         this.usuario = {
11             usuario: usuario,
12             password: pass
13         };
14     }
15     LoginManager.prototype.validarUsuario = function (rol) {
16         if (this.usuario.rol === undefined)
17             this.usuario.rol = rol;
18         return (this.usuario.rol === ERoles.MANAGER);
19     };
20     return LoginManager;
21 })();
22 var app = new LoginManager('miUsuario', 'miClave');
23 app.validarUsuario(0);
24
```

# Modularizando Apps

```
1 import { Config } from './config';
2 import * as controller from './controllers/controllers';
3 import * as service from './services/services';
4
5 let appKerberos = angular.module('kerberosQuery', ['ui.router']);
6 appKerberos.config(Config);
7
8 appKerberos.service('QueueService', service.QueueService);
9 appKerberos.service('AgentService', service.AgentService);
10 appKerberos.service('LlamadaQueueService', service.QueueCallService);
11 appKerberos.service('UsuarioService', service.LoginService);
12
13 appKerberos.controller(controller.IntroController.getName(), controller.IntroController);
14 appKerberos.controller(controller.Callcenter.getName(), controller.Callcenter);
15 appKerberos.controller(controller.Consola.getName(), controller.Consola);
16 appKerberos.controller(controller.CCAgente.getName(), controller.CCAgente);
17
18 appKerberos.controller(controller.Agente.getName(), controller.Agente);
```

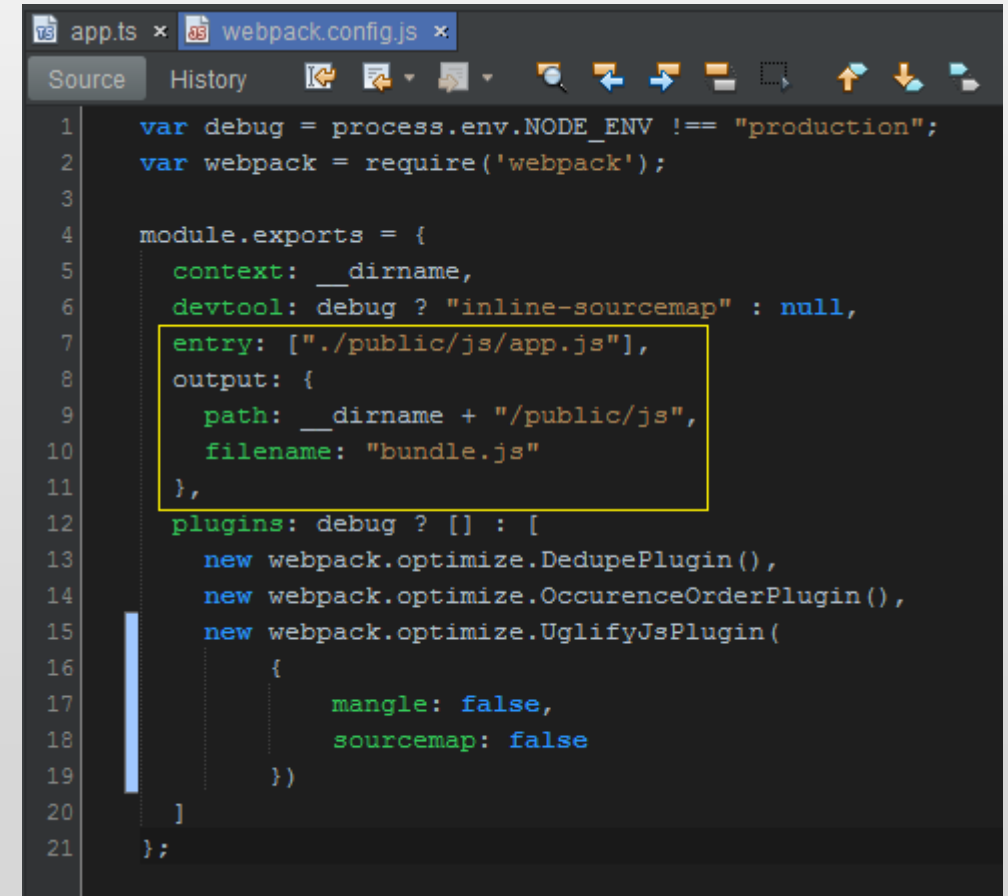
```
1 export * from './service.queues';
2 export * from './service.agente';
3 export * from './service.llamadasQueue';
4 export * from './service.login';
```

```
28 export interface ILoginService
29 {
30     getUsuario(): IUsuario;
31     authUser(credenciales: IAuthCredentials): IAuthResponse;
32 }
33
34 export class LoginService implements ILoginService
35 {
36     private usuario: IUsuario;
37     constructor() {}
38
39     getUsuario(): IUsuario
40     {
41         return this.usuario;
42     }
43
44     authUser(login: IAuthCredentials): IAuthResponse
45     {
46         if (login.usuario === "demo" && login.password === "demo")
47         {
48             this.usuario = {...5 lines };
49             return {...5 lines };
50         } else if (login.usuario === "401" && login.password === "401")
51         {
52             this.usuario = {...5 lines };
53             return {...5 lines };
54         }
55     }
56 }
```



# Entorno de Desarrollo.

1. NodeJS.  
<https://nodejs.org/en/>
2. TypeScript.  
`npm install typescript --global`
3. Typings.  
`npm install typings --global`
4. Webpack.  
`npm install webpack --global`
5. IDE / Code Editor  
Netbeans || VSCode || WebStorm || SublimeText



```
1  var debug = process.env.NODE_ENV !== "production";
2  var webpack = require('webpack');
3
4  module.exports = {
5    context: __dirname,
6    devtool: debug ? "inline-sourcemap" : null,
7    entry: ["/public/js/app.js"],
8    output: {
9      path: __dirname + "/public/js",
10     filename: "bundle.js"
11   },
12   plugins: debug ? [] : [
13     new webpack.optimize.DedupePlugin(),
14     new webpack.optimize.OccurenceOrderPlugin(),
15     new webpack.optimize.UglifyJsPlugin(
16       {
17         mangle: false,
18         sourcemap: false
19       }
20     )
21   ]
22 };
23
```

# Let's code!

AngularJS + TypeScript + Webpack

# Mitos de TypeScript en Apps modulares.

1. Programar en TypeScript te convierte en un programador de segunda clase.
2. Las apps modulares en TypeScript no son compatibles con módulos escritos en JS.
3. TypeScript desaparecerá cuando ES2015 o ES6 sean realmente estándares.
4. TypeScript no permite usar todos los beneficios de ES2015.

# Gracias!