

# Introducción a Angular con Typescript

Elías García



# Elías García

---

Front-End Web Developer



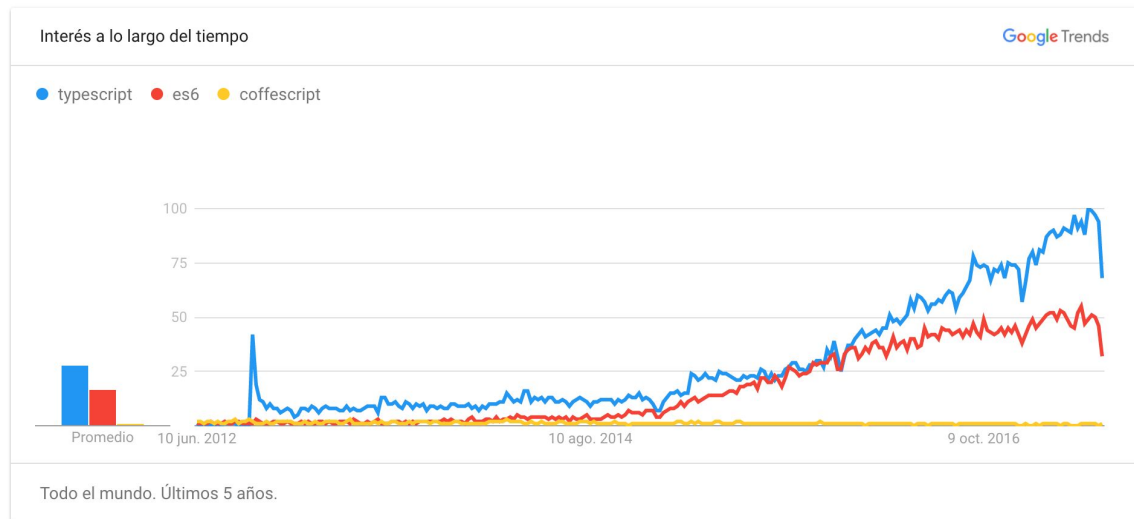
<https://eliasgarcia.es>

<https://twitter.com/egarciadev>



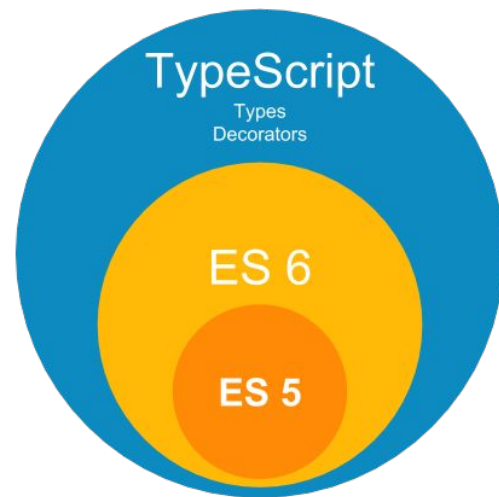
# TypeScript

# Popularidad



# ¿Por qué TypeScript?

- Superset de ES6
- Tipado estático => Autocompletado, errores...
  - Inferencia de tipos
  - Tipos opcionales
- Orientado a objetos
  - Clases
  - Interfaces
- Decoradores
- Fácil adopción para programadores Java o C#
- Escalabilidad y mantenibilidad
- Crecimiento de un 250% de 2015 a 2016 según GitHub



TypeScript is a javascript superset

# Código de ejemplo

Decorador

Clase

Modificador  
de acceso

```
new.component.ts x
1 import { Component, OnInit } from '@angular/core';
2 import { FormGroup } from '@angular/forms';
3 import { TodosService } from '../todos.service';
4
5 @Component({
6   selector: 'app-new',
7   templateUrl: './new.component.html',
8   styleUrls: ['./new.component.css']
9 })
10 export class NewComponent implements OnInit {
11
12   public description: string;
13
14   constructor(private todosService: TodosService) { }
15
16   ngOnInit() {
17   }
18
19   public onSubmit(todoForm: FormGroup) {
20     this.todosService.createTodo(this.description);
21     todoForm.reset();
22   }
23
24 }
25
```

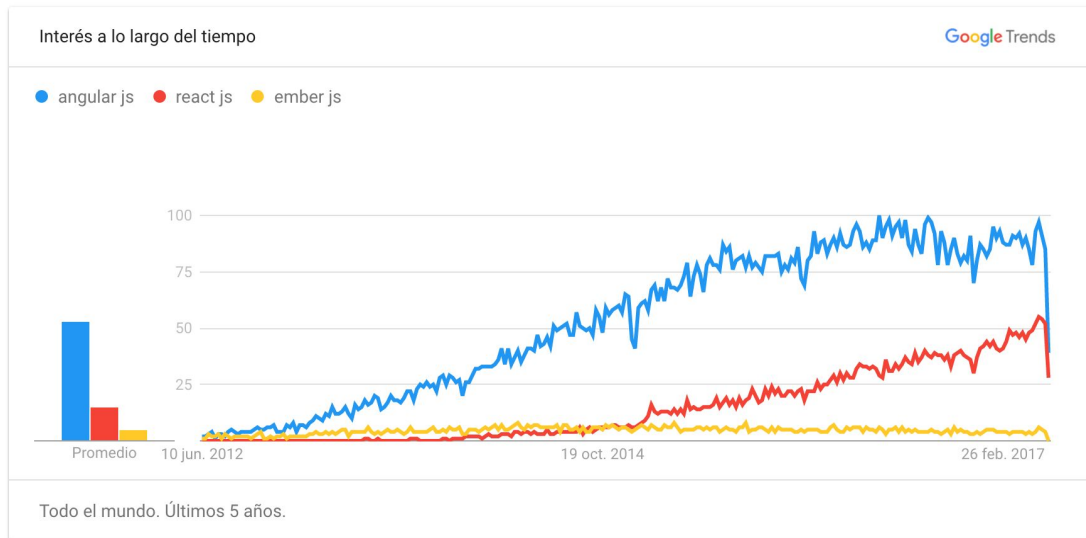
Tipado estático

¿WTFFF?

The background of the slide is a solid dark blue. In the top right corner, there is a decorative geometric pattern consisting of several squares and triangles in different shades of blue, including dark blue, medium blue, and light blue.

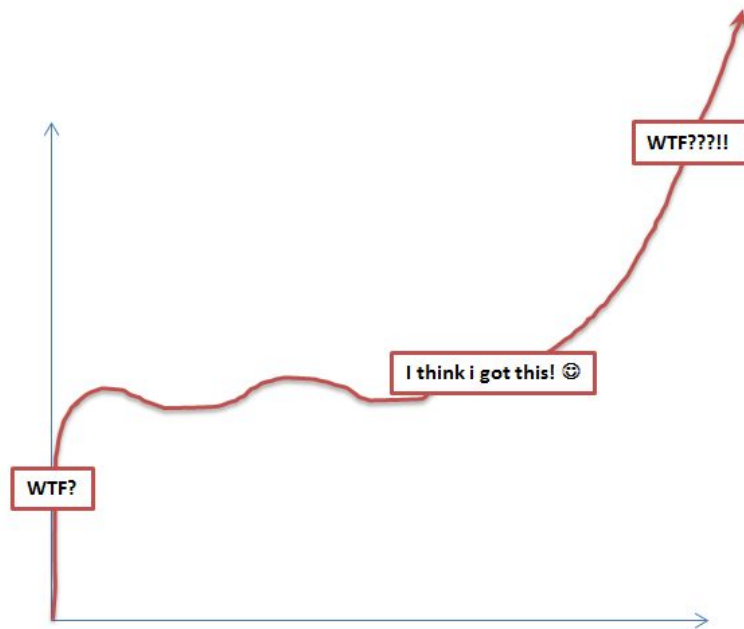
# Angular

# Popularidad















# Curva de aprendizaje



# Comunidad

Organizations with the most open source contributors

	Microsoft	16,419
	facebook	15,682
	docker	14,059
	angular	12,841
	google	12,140
	atom	9,698
	FortAwesome	9,617
	elastic	7,220
	Apache	6,999
	npm	6,815

Fuente: GitHub Octoverse 2016

# ¿Qué es Angular?

- Plataforma para desarrollo de SPA's
  - Scaffolding & Generating
  - Serving & Developing
  - Building & Distributing
  - E2E & Unit Testing
- Proporciona todo lo necesario: Router, HTTP...
- Patrón MVC
- Inyección de dependencias
- Arquitectura modular basada en componentes
- Angular !== AngularJS

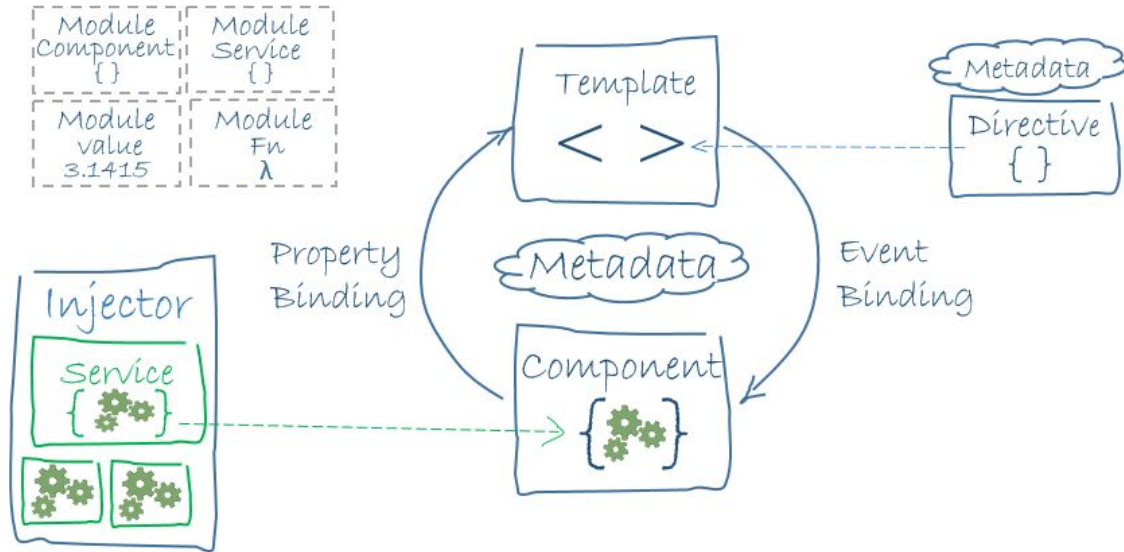


# Angular vs... React

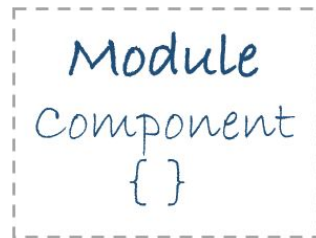
- Plataforma, NO LIBRERÍA!
- TypeScript
- Data Binding
- Modularidad
- Base estándar para proyectos
- Adecuado para grandes proyectos
- Mantenibilidad



# Arquitectura (I) - Vista general



# Arquitectura (II) - Módulos



- Agrupación lógica de funcionalidad
- Al menos un módulo (AppModule)
- Son clases TypeScript decoradas con @NgModule
- Pueden ser cargados de forma Lazy => ¡OJO!

Buenas prácticas:

- Módulo Core => Layout de la aplicación
- Módulo Shared => Agrupación de módulos comunes



# Arquitectura (III) - Módulos

```
12  @NgModule({
13    imports: [
14      CommonModule,
15      CoreModule,
16      FormsModule,
17      RouterModule
18    ],
19    declarations: [NewComponent, ListComponent, TodosComponent, FiltersComponent],
20    providers: [TodosService],
21    exports: [],
22    schemas: [ CUSTOM_ELEMENTS_SCHEMA ]
23  })
24  export class TodosModule { }
```

# Arquitectura (IV) - Componentes

- Base de una aplicación en Angular
- Son clases TypeScript decoradas con `@NgComponent`
- Cada componente está asociado a una vista (HTML), con la cual interactúa
- Cada ruta de nuestra aplicación suele ser un componente el cual, a su vez, contiene más componentes
- Siguen un ciclo de vida => `onChanges`, `onInit`, `onDestroy`...



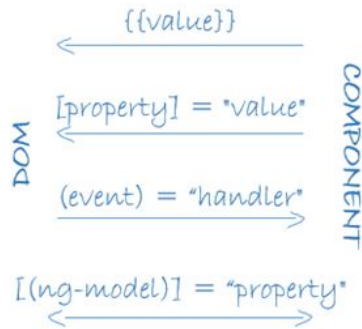


# Arquitectura (V) - Componentes

```
6  @Component({
7    selector: 'app-todos',
8    templateUrl: './todos.component.html',
9    styleUrls: ['./todos.component.css']
10 })
11 export class TodosComponent implements OnInit {
12
13   public todos: Todo[];
14
15   constructor(private route: ActivatedRoute, private todosService: TodosService) { }
16
17   ngOnInit() {
18     this.route.queryParams.subscribe(params => {
19       this.todosService.getTodos$.subscribe(todos => {
20         switch (params.filter) {
21           case 'active':
22             this.todos = todos.filter(todo => todo.completed === false);
23             break;
24           case 'completed':
25             this.todos = todos.filter(todo => todo.completed === true);
26             break;
27           default:
28             this.todos = todos;
29             break;
30         }
31       });
32     });
33   }
34
35 }
```

# Arquitectura (VI) - Data Binding

- Mecanismo para sincronizar partes de una vista con partes de un componente
- 4 formas de Data Binding
  - Interpolación: `{{ }}`
  - Property Binding: `[property]="value"`
  - Event Binding: `(event)="handler"`
  - Two Way Data Binding: `[(ngModel)]="value"`



# Arquitectura (VI) - Data Binding

```
new.component.html x new.component.ts
1 <div class="columns">
2   <div class="column col-6 col-lg-12 centered">
3     <h2 class="text-center">Create a new Todo</h2>
4     <form (ngSubmit)="onSubmit(todoForm)" #todoForm="ngForm">
5       <div class="input-group">
6         <input
7           [(ngModel)]="description"
8           name="description"
9           type="text"
10          class="form-input input-lg"
11          placeholder="Todo description..."
12          required
13          autofocus
14        />
15        <button
16          type="submit"
17          [disabled]="!todoForm.form.valid"
18          class="btn btn-primary input-group-btn btn-lg">
19          Submit
20        </button>
21      </div>
22    </form>
23  </div>
24 </div>
25
```

```
new.component.html x new.component.ts x
1 import { Component, OnInit } from '@angular/core';
2 import { FormGroup } from '@angular/forms';
3 import { TodosService } from '../todos.service';
4
5 @Component({
6   selector: 'app-new',
7   templateUrl: './new.component.html',
8   styleUrls: ['./new.component.css']
9 })
10 export class NewComponent implements OnInit {
11   public description: string;
12
13   constructor(private todosService: TodosService) { }
14
15   ngOnInit() {
16   }
17
18   public onSubmit(todoForm: FormGroup) {
19     this.todosService.createTodo(this.description);
20     todoForm.reset();
21   }
22
23 }
24
25
```

# Arquitectura (VIII) - Servicios

- Un servicio es una clase TypeScript anotada con `@Injectable`
- La funcionalidad de los servicios debe de ser muy concreta
  - Logger
  - Data Service
- En Angular suelen utilizarse para comunicarse con API's REST
- Los servicios son proporcionados a los componentes mediante la inyección de dependencias

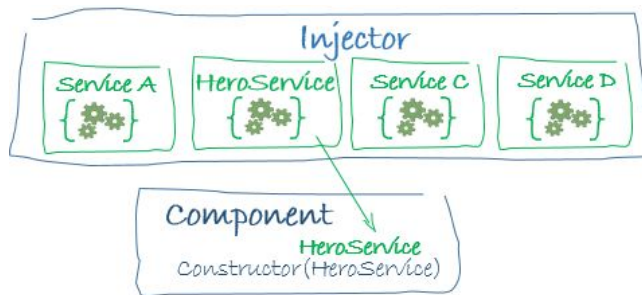


# Arquitectura (IX) - Servicios

```
6  @Injectable()
7  export class TodosService {
8
9      private todos: Todo[] = [];
10     private todos$: BehaviorSubject<Todo[]>;
11
12     constructor() {
13         this.todos$ = new BehaviorSubject(this.todos);
14     }
15
16     createTodo(description: string) {
17         this.todos.push(new Todo(description));
18         this.todos$.next(this.todos);
19     }
20
21     getTodos(): Observable<Todo[]> {
22         return this.todos$.asObservable();
23     }
24
25     deleteTodo(id: number): void {
26         this.todos = this.todos.filter(todo => todo.id !== id);
27         this.todos$.next(this.todos);
28     }
29
30 }
31
```

# Arquitectura (X) - Inyección de dependencias

- Es uno de los puntos fuertes de Angular
- Proporciona una instancia de una clase con sus dependencias: HTTP...
- Cada componente tiene un inyector
- Al igual que los componentes, los inyectores son jerárquicos



# Arquitectura (XI) - Inyección de dependencias

```
new.component.ts x
1 import { Component, OnInit } from '@angular/core';
2 import { FormGroup } from '@angular/forms';
3 import { TodosService } from '../todos.service';
4
5 @Component({
6   selector: 'app-new',
7   templateUrl: './new.component.html',
8   styleUrls: ['./new.component.css']
9 })
10 export class NewComponent implements OnInit {
11
12   public description: string;
13
14   constructor(private todosService: TodosService) { }
15
16   ngOnInit() {
17   }
18
19   public onSubmit(todoForm: FormGroup) {
20     this.todosService.createTodo(this.description);
21     todoForm.reset();
22   }
23
24 }
25
```

Inyección de  
un servicio



# Arquitectura (XII) - Otros elementos

- Directivas
  - \*ngIf
  - \*ngFor
  - etc.
- Pipes
  - async
  - json
  - etc.





# Comunicación entre componentes (I)

- Es un punto clave en una aplicación desarrollada con Angular
- Los componentes se pueden comunicar de dos formas
- Los componentes pueden recibir datos de un componente padre a través del decorador `@Input()`. El padre se los enviará haciendo un data binding de propiedad `[property]="value"`.
- Los componentes pueden emitir datos a su padre a través del decorador `@Output()`. El padre recibirá los datos haciendo un data binding de evento `(event)="value"`



# Comunicación entre componentes (II)

Hijo (.ts)

```
1 <app-new></app-new>
2 <app-list [todos]="todos"></app-list>
```

Padre (.html)

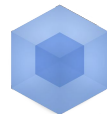
```
5 @Component({
6   selector: 'app-list',
7   templateUrl: './list.component.html',
8   styleUrls: ['./list.component.css']
9 })
10 export class ListComponent implements OnInit {
11
12   @Input() public todos: Todo[];
13
14   constructor(private todosService: TodosService) { }
15
16   ngOnInit() { }
17
18 }
```

# Comunicación entre componentes (III)

- Los componentes también pueden comunicarse de otra forma, utilizando el patrón Observador.
- Este patrón define una relación uno a muchos entre objetos, de forma que cuando uno cambia su estado (Observable), notifica a todos los dependientes (Observers)
- La implementación de este patrón está muy presente en Angular (Router por ejemplo)
- Librería RxJs



# Angular CLI (I)



- Interfaz de línea de comandos que nos permite:
  - Inicializar un proyecto base
  - Generar esqueletos de componentes, módulos, directivas etc.
  - Ejecutar un servidor de desarrollo con funcionalidad de live reload
  - Construir y preparar nuestra aplicación para entornos de producción
  - Ejecutar las pruebas unitarias y E2E
- Nos permite centrarnos en lo importante y olvidarnos del tedioso proceso de configuración de Angular, Webpack, Karma, Protractor etc.
- Nos permite ejecutar y desplegar nuestra aplicación de forma sencilla
- `npm i -g @angular/cli`

# Angular CLI (II)

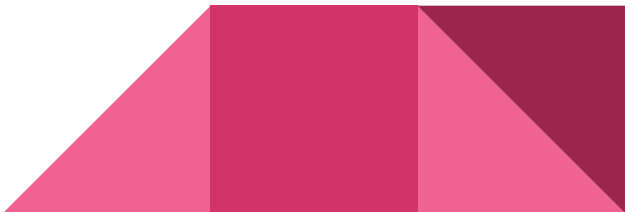
Comandos relevantes:

- `ng new`: crea una aplicación base con todo lo necesario
- `ng generate`: crea componentes, módulos, directivas...
- `ng serve`: ejecuta la aplicación en un servidor de desarrollo local
- `ng test`: ejecuta las pruebas de unidad
- `ng e2e`: ejecuta las pruebas E2E
- `ng build`: construye la aplicación para producción



# AOT

Angular soporta AOT (Ahead Of Time), lo cual nos permite desplegar una versión ya compilada de nuestra aplicación. Las mejoras sobre la compilación JIT (Just In Time) son bastante importantes:

- Aplicación precompilada => Renderizado más rápido
  - HTML + CSS Inline => Menos respuestas asíncronas
  - No necesitamos el compilador => El tamaño de Angular se reduce a la mitad
  - Detección de errores en los templates
  - No HTML, solo JS => Mejoras de seguridad
- 

# Lazy Loading

- El router de Angular nos permite cargar módulos de forma lazy, es decir, nos permite descargar módulos a medida que los vamos necesitando. De esta forma nos evitamos descargar toda la aplicación cada vez que accedamos a ella.
- Una buena implementación de esta técnica puede producir grandes resultados.



# Angular Universal

- Nuestra aplicación final estará formada sólo por JS. Tanto los templates HTML como las hojas de estilo CSS se añadirán al código JS de nuestros componentes => ¿SEO?
- Angular Universal nos permite pre renderizar ciertas partes de nuestra aplicación en el lado del servidor, de forma que cuando se le solicite una o varias rutas determinadas éste le proporcionará un fichero HTML pre renderizado





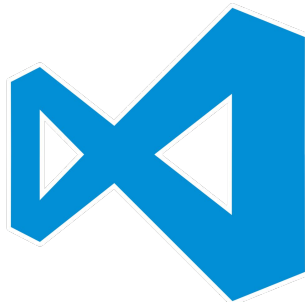
# Angular y los frameworks CSS

- La mayoría de los frameworks como Bootstrap utilizan jQuery
- Añadir una dependencia así a nuestro proyecto supone un gran impacto de rendimiento
- Para solucionar ésto, la comunidad ha ido adaptando estos frameworks a Angular y se han ido creando librerías de componentes y directivas basadas en estos frameworks
- La mayoría están muy verdes
- Gran alternativa: Spectre.css



# Entorno de desarrollo

- Visual Studio Code
  - Intellisense
  - Debugging
  - Git
  - Extensiones
- Angular Essentials - Extension Pack for VSCode
  - Creado por John Papa, miembro del Core Team de Angular
  - Path Intellisense, Angular Snippets, TSLint...



# Aplicación de ejemplo

# Angular Todo - Introducción

- La aplicación de ejemplo consiste en una SPA para gestionar tareas:
  - Añadir una nueva tarea
  - Editar una tarea
  - Eliminar una tarea
  - Marcar o desmarcar como completada una tarea
  - Filtrar tareas por activas o completadas



# Angular Todo - Instalación y ejecución

- git clone <https://github.com/elias-garcia/angular-todo.git>
- cd angular-todo
- npm i
- npm start



# Angular Todo - Vista general



Create a new Todo

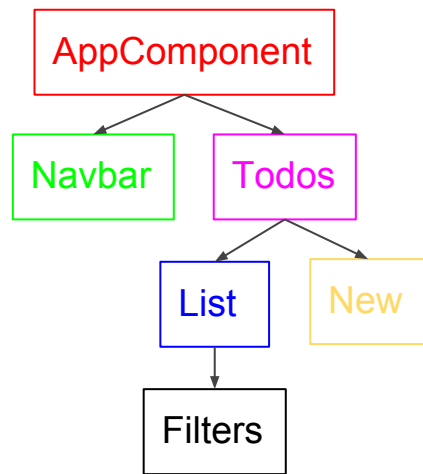
 

Todo List

**All** Active Completed

Status	Todo	Actions
<input type="checkbox"/>	Bajar al perro a pasear	<input type="button" value="🗑"/>
<input checked="" type="checkbox"/>	Hacer la cama	<input type="button" value="🗑"/>

# Angular Todo - Componentes



SourceANGULAR TODOContact

Create a new Todo

Todo description...Submit

Todo List

AllActiveCompleted

Status	Todo	Actions
<input type="checkbox"/>	Bajar al perro a pasear	
<input checked="" type="checkbox"/>	Hacer la cama	

