plain concepts

SINGLE PAGE APPLICATION (SPA)

- La página se descarga completeta una vez desde el servidor
  - Una sola página
- Gestionamos la navegación en cliente (routing)
  - Se integra con el histórico del navegador
  - Podemos ir hacia atrás utilizando el navegador
- El servidor en el resto del ciclo de vida:
  - Proporciona plantillas parciales de páginas bajo demanda
  - Proporciona javascript bajo demanda (frameworks)
  - Devuelve y procesa información mediante llamadas XHR (API)
  - No renderiza HTML

https://angular.io/

INTRODUCCIÓN

- Es un framework para desarrollo de SPA.
- Permite extender el HTML con etiquetas propias.
- Interfaz basado en componentes (no en páginas) .
- Podemos desarrollar con él utilizando **TypeScript o** JavaScript.

ACTUALIDAD

- AngularJS
  - V1.8.2
  - Última versión estable: octubre de 2020

- Angular (Angular 2 o Angular2+)
  - V14.2.10
  - Última versión estable: noviembre 2022
  - Typescript
  - Nuevas directivas
  - Componentes sustituye a *controllers* y *$scope*

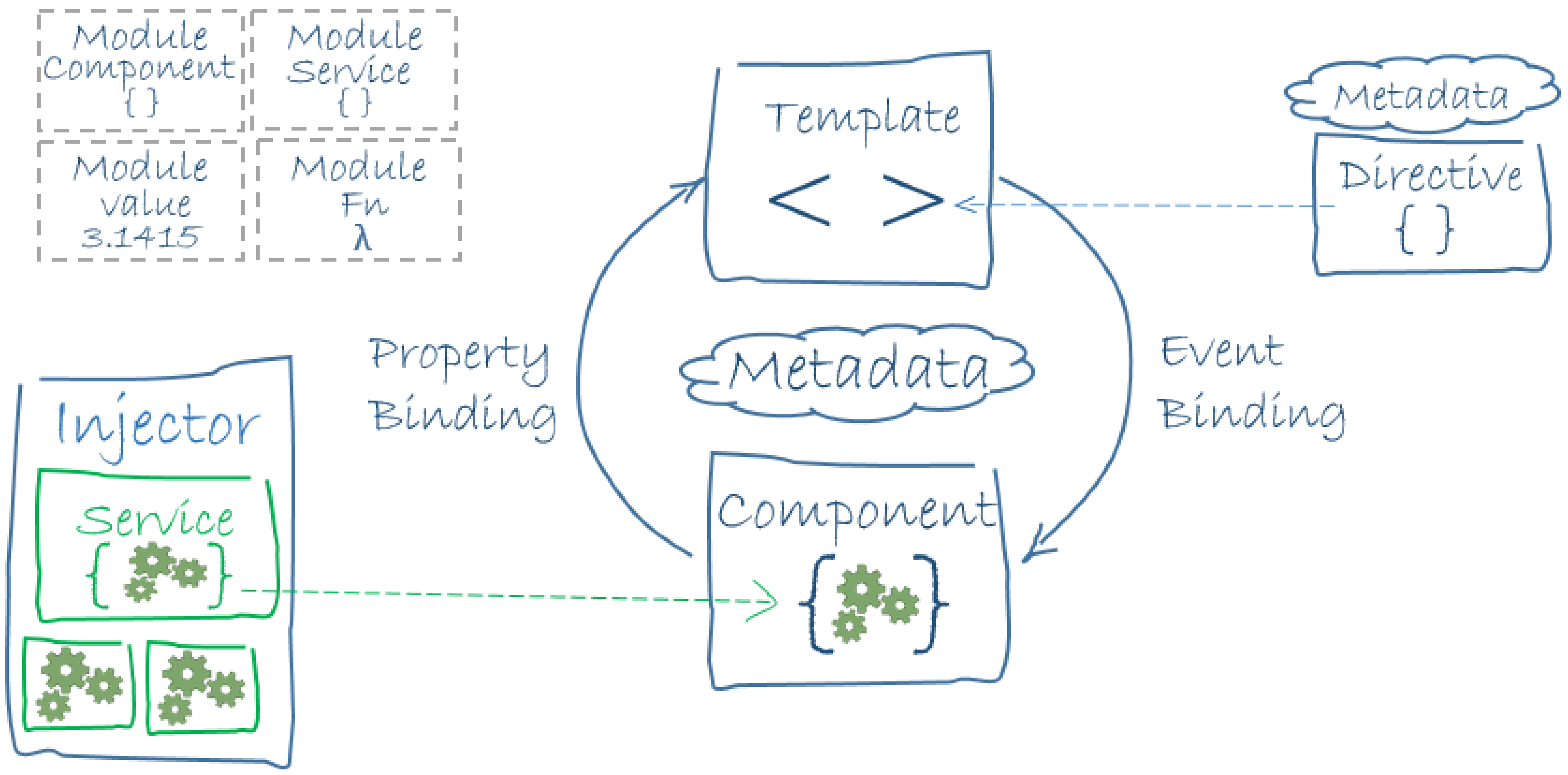
FUNCIONALIDADES

- Inyección de dependencias
- Servicios
- Cliente http
- Navegación por la app (Router)
- Animaciones
- Internacionalización
- Soporte para tests unitarios y e2e
- Librerías de componentes: material design, ng-bootstrap, …
- Renderizado en el servidor
- CLI
- PWA

ARQUITECTURA

- Módulos:
  - Bloques de la aplicación, que agrupan componentes, servicios, rutas, …
- Componentes:
  - Contiene una vista (template: html + css), datos, y/o lógica; y forma parte del árbol DOM.
- Servicios:
  - Capa de datos, lógica de negocio, no acoplada a los componentes.
  - Peticiones a nuestro API.
- Routing:
  - Responsable de la navegación de la aplicación.
  - Renderiza componentes basándose en el estado de la URL.
- Directivas:
  - Añade comportamiento, extiende y/o transforma un elemento del DOM o componente.
  - Construir las nuestras o usar las de angular.

ARQUITECTURA



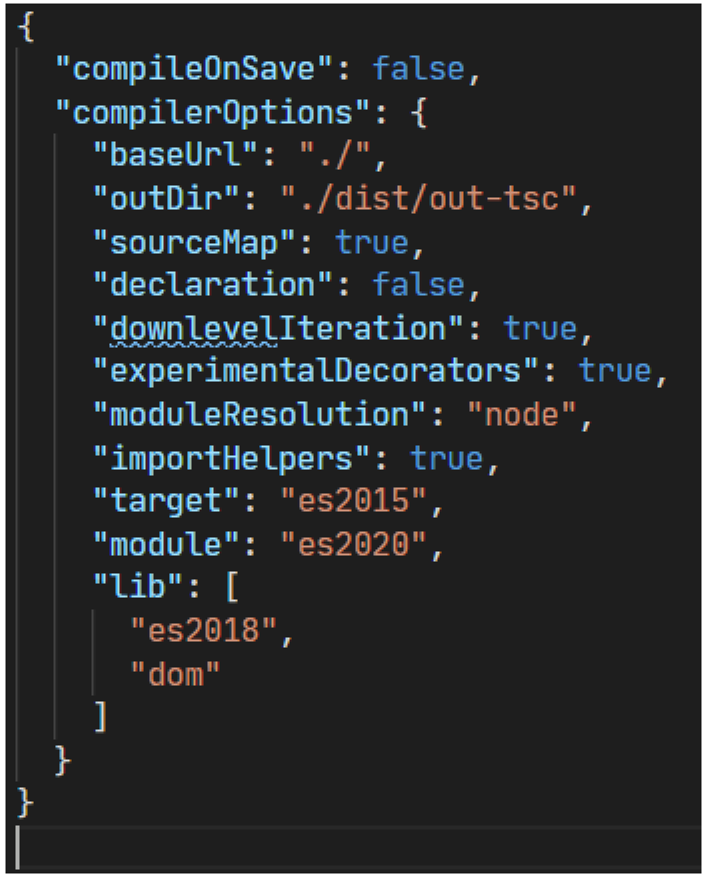
https://angular.io/docs/ts/latest/guide/architecture.html

plain concepts

BOOTSTRAPPING

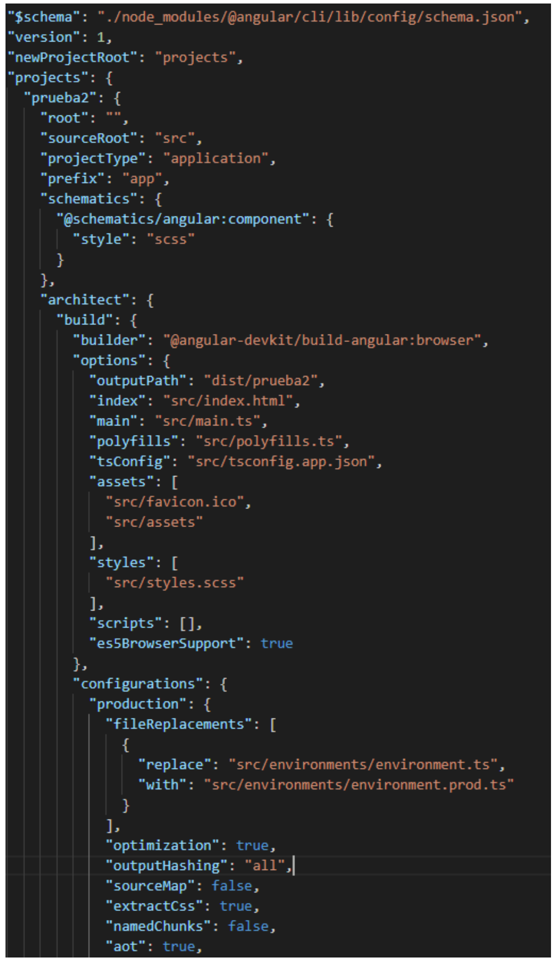
- *app/app.component.ts* – Donde definimos el componente raiz de nuestra aplicación.
- *app/app.module.ts* – La definición del modulo de entrada a ser arrancado.
- *index.html* – Página html donde el componente será renderizado.
- *main.ts* – El “pegamento” que combina el componente y la página html.

TSCONFIG.JSON

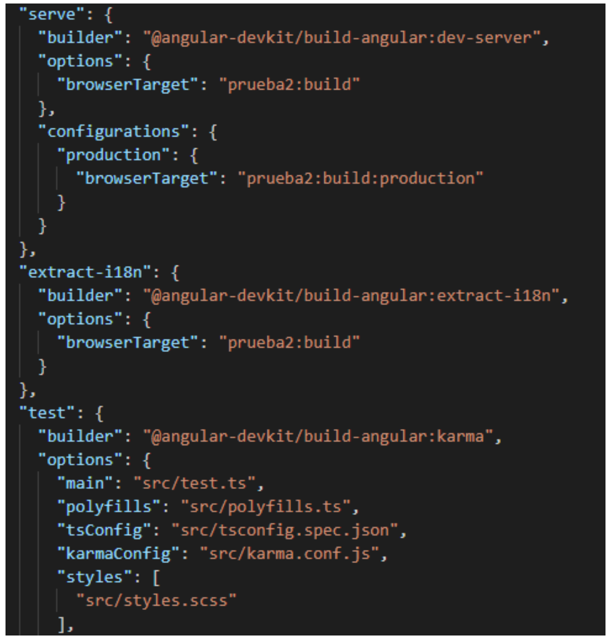


```json
{
  "compileOnSave": false,
  "compilerOptions": {
    "baseUrl": "./",
    "outDir": "./dist/out-tsc",
    "sourceMap": true,
    "declaration": false,
    "downlevelIteration": true,
    "experimentalDecorators": true,
    "moduleResolution": "node",
    "importHelpers": true,
    "target": "es2015",
    "module": "es2020",
    "lib": [
      "es2018",
      "dom"
    ]
  }
}
```

ANGULAR.JSON



```
"$schema": "./node_modules/@angular/cli/lib/config/schema.json",
"version": 1,
"newProjectRoot": "projects",
"projects": {
  "prueba2": {
    "root": "",
    "sourceRoot": "src",
    "projectType": "application",
    "prefix": "app",
    "schematics": {
      "@schematics/angular:component": {
        "style": "scss"
      }
    },
    "architect": {
      "build": {
        "builder": "@angular-devkit/build-angular:browser",
        "options": {
          "outputPath": "dist/prueba2",
          "index": "src/index.html",
          "main": "src/main.ts",
          "polyfills": "src/polyfills.ts",
          "tsConfig": "src/tsconfig.app.json",
          "assets": [
            "src/favicon.ico",
            "src/assets"
          ],
          "styles": [
            "src/styles.scss"
          ],
          "scripts": [],
          "es5BrowserSupport": true
        },
        "configurations": {
          "production": {
            "fileReplacements": [
              {
                "replace": "src/environments/environment.ts",
                "with": "src/environments/environment.prod.ts"
              }
            ],
            "optimization": true,
            "outputHashing": "all",
            "sourceMap": false,
            "extractCss": true,
            "namedChunks": false,
            "aot": true,
```



```
"serve": {
  "builder": "@angular-devkit/build-angular:dev-server",
  "options": {
    "browserTarget": "prueba2:build"
  },
  "configurations": {
    "production": {
      "browserTarget": "prueba2:build:production"
    }
  }
},
"extract-i18n": {
  "builder": "@angular-devkit/build-angular:extract-i18n",
  "options": {
    "browserTarget": "prueba2:build"
  }
},
"test": {
  "builder": "@angular-devkit/build-angular:karma",
  "options": {
    "main": "src/test.ts",
    "polyfills": "src/polyfills.ts",
    "tsConfig": "src/tsconfig.spec.json",
    "karmaConfig": "src/karma.conf.js",
    "styles": [
      "src/styles.scss"
    ],
```

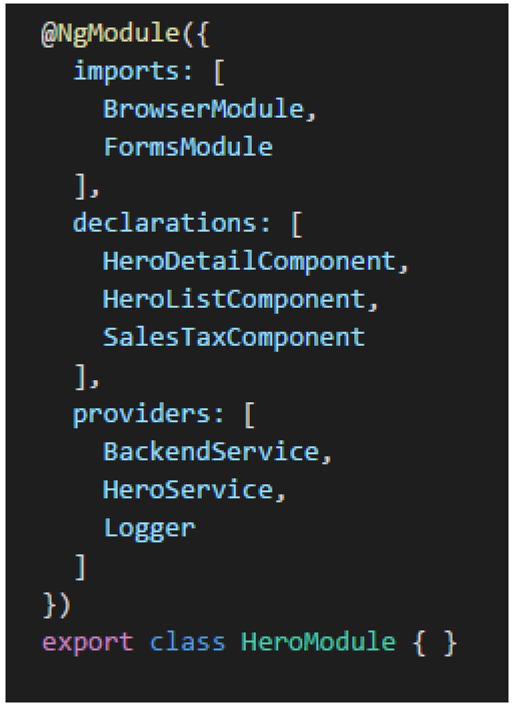
CLI

COMMAND	ALIAS	DESCRIPTION
add		Adds support for an external library to your project.
analytics		Configures the gathering of Angular CLI usage metrics. See https://angular.io/cli/usage-analytics-gathering.
build	b	Compiles an Angular app into an output directory named dist/ at the given output path. Must be executed from within a workspace directory.
config		Retrieves or sets Angular configuration values in the angular.json file for the workspace.
deploy		Invokes the deploy builder for a specified project or for the default project in the workspace.
doc	d	Opens the official Angular documentation (angular.io) in a browser, and searches for a given keyword.
e2e	e	Builds and serves an Angular app, then runs end-to-end tests using Protractor.
generate	g	Generates and/or modifies files based on a schematic.
help		Lists available commands and their short descriptions.
lint	l	Runs linting tools on Angular app code in a given project folder.
new	n	Creates a new workspace and an initial Angular app.
run		Runs an Architect target with an optional custom builder configuration defined in your project.
serve	s	Builds and serves your app, rebuilding on file changes.
test	t	Runs unit tests in a project.
update		Updates your application and its dependencies. See https://update.angular.io/
version	v	Outputs Angular CLI version.
xi18n	i18n-extract	Extracts i18n messages from source code.

plain concepts

MÓDULOS

- Permiten modularizar nuestra aplicación en base a funcionalidades o lo que queramos.
- Siempre hay que definir uno (root module)

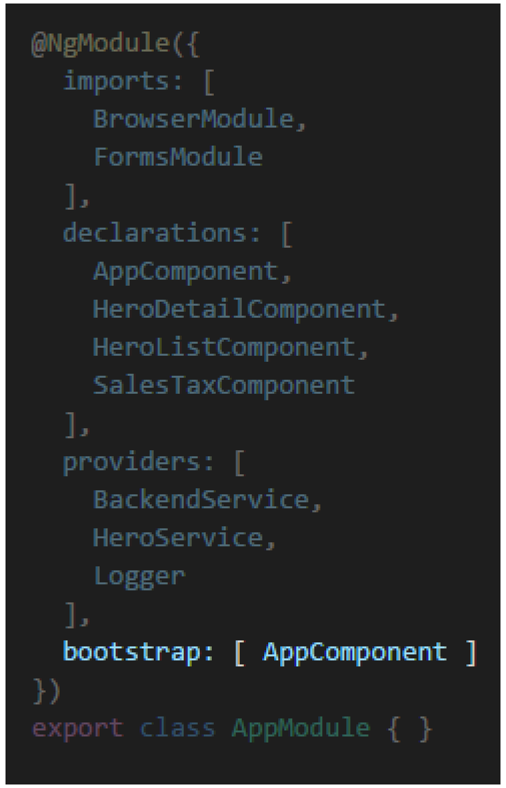


```
@NgModule({
  imports: [
    BrowserModule,
    FormsModule
  ],
  declarations: [
    HeroDetailComponent,
    HeroListComponent,
    SalesTaxComponent
  ],
  providers: [
    BackendService,
    HeroService,
    Logger
  ]
})
export class HeroModule { }
```

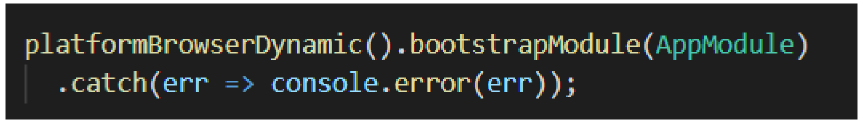
plain concepts

ROOT MODULE

- Define cuál será el componente raíz que Angular creará e insertará en el index.html



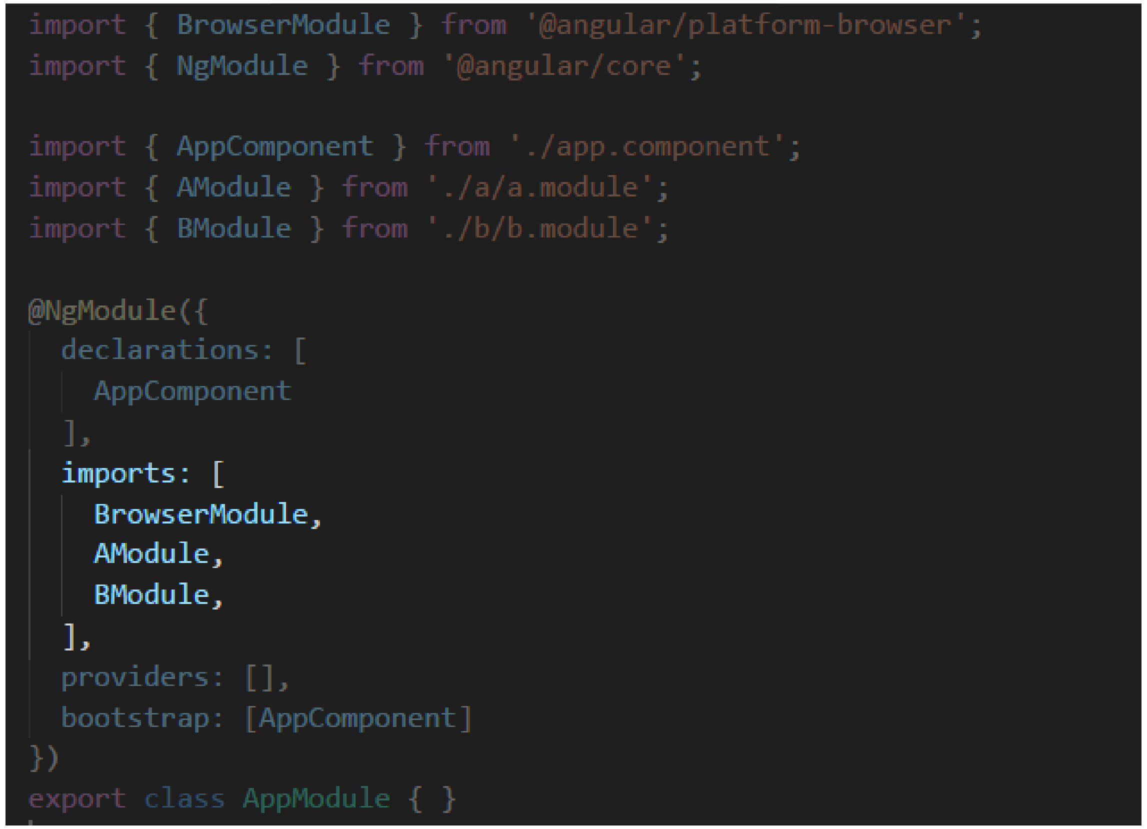
```
@NgModule({
  imports: [
    BrowserModule,
    FormsModule
  ],
  declarations: [
    AppComponent,
    HeroDetailComponent,
    HeroListComponent,
    SalesTaxComponent
  ],
  providers: [
    BackendService,
    HeroService,
    Logger
  ],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
```



```
platformBrowserDynamic().bootstrapModule(AppModule)
  .catch(err => console.error(err));
```

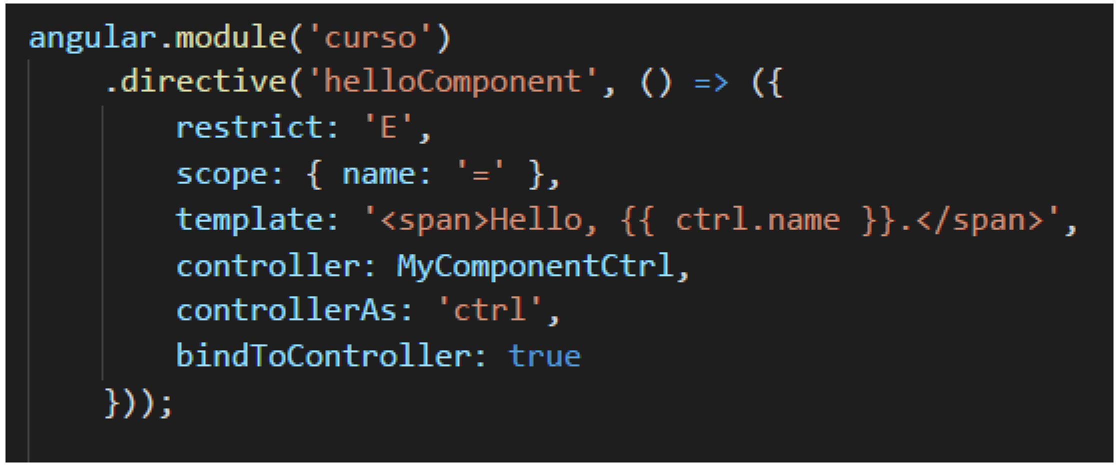
main.ts

IMPORTANDO OTROS MÓDULOS



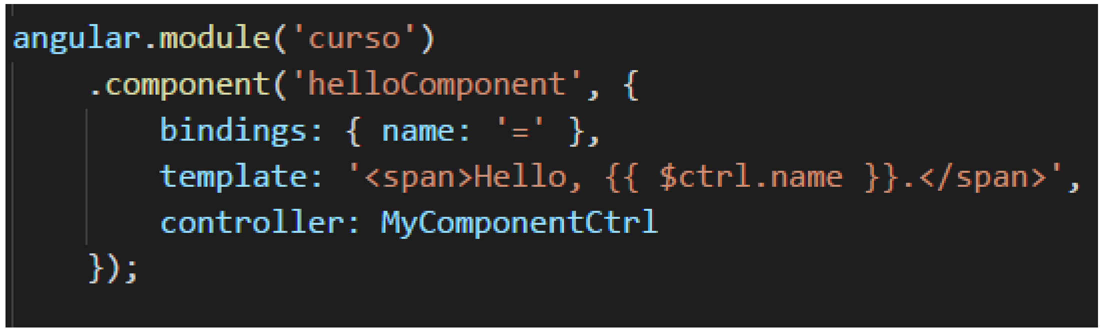
```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppComponent } from './app.component';
import { AModule } from './a/a.module';
import { BModule } from './b/b.module';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    AModule,
    BModule,
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

ANGULARJS



```
angular.module('curso')
    .directive('helloComponent', () => ({
        restrict: 'E',
        scope: { name: '=' },
        template: '<span>Hello, {{ ctrl.name }}.</span>',
        controller: MyComponentCtrl,
        controllerAs: 'ctrl',
        bindToController: true
    }));
```

angular 1.x

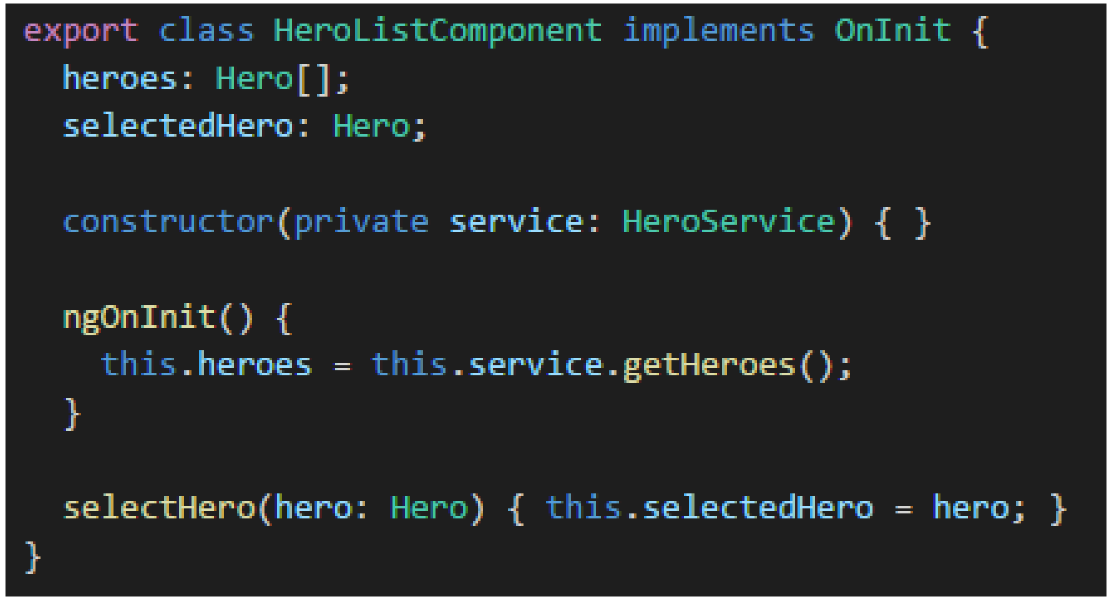


```
angular.module('curso')
    .component('helloComponent', {
        bindings: { name: '=' },
        template: '<span>Hello, {{ $ctrl.name }}.</span>',
        controller: MyComponentCtrl
    });
```

angular 1.5

CLASE

- Controla la lógica de una porción de vista y puede interactuar con esta.

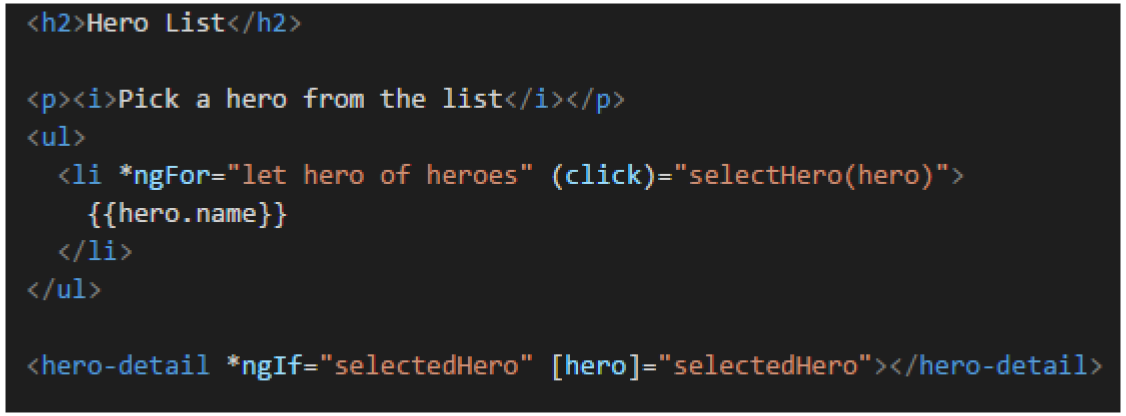


```
export class HeroListComponent implements OnInit {
  heroes: Hero[];
  selectedHero: Hero;

  constructor(private service: HeroService) { }

  ngOnInit() {
    this.heroes = this.service.getHeroes();
  }

  selectHero(hero: Hero) { this.selectedHero = hero; }
}
```

hero-list.component.ts

TEMPLATE

- Parte de html que determina cómo renderizar el modelo expuesto por un componente.

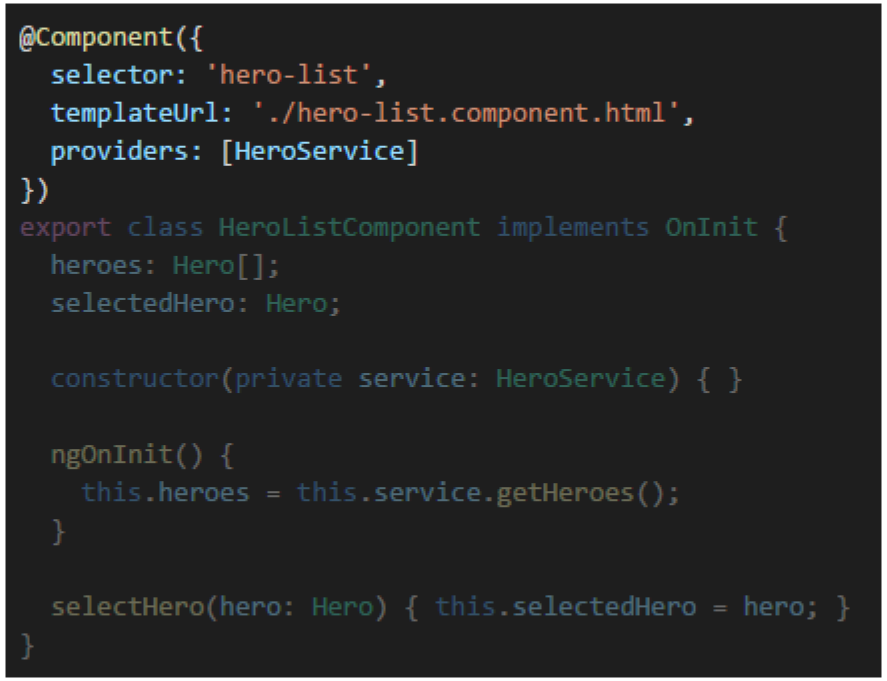


```html
<h2>Hero List</h2>

<p><i>Pick a hero from the list</i></p>
<ul>
  <li *ngFor="let hero of heroes" (click)="selectHero(hero)">
    {{hero.name}}
  </li>
</ul>

<hero-detail *ngIf="selectedHero" [hero]="selectedHero"></hero-detail>
```

hero-list.component.html

METADATA

- Le dice a Angular cómo procesar una clase.



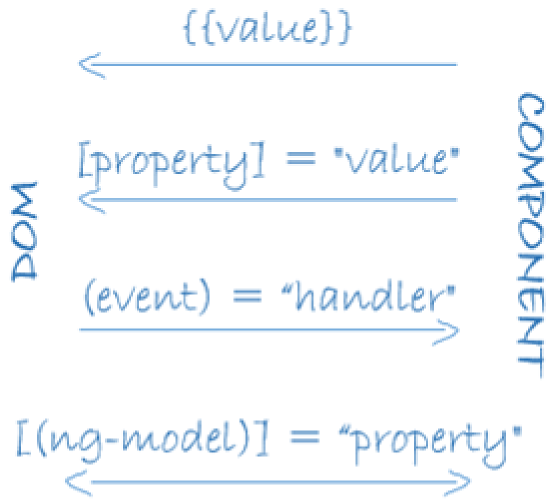
```
@Component({
  selector: 'hero-list',
  templateUrl: './hero-list.component.html',
  providers: [HeroService]
})
export class HeroListComponent implements OnInit {
  heroes: Hero[];
  selectedHero: Hero;

  constructor(private service: HeroService) { }

  ngOnInit() {
    this.heroes = this.service.getHeroes();
  }

  selectHero(hero: Hero) { this.selectedHero = hero; }
}
```

hero-list.component.ts

DATA BINDING

- Permite conectar partes de un template con partes de un componente.

- 4 formas:
  - Interpolation
  - Property binding
  - Event binding
  - Two way binding



plain concepts

TEMPLATE SYNTAX

```
<p>My current hero is {{currentHero.name}}</p>
```

```
<input [value]="firstName">
```

```
<div [attr.role]="myAriaRole">
```

```
<div [class.extra-sparkle]="isDelightful">
```

```
<div [ngClass]="{active: isActive, disabled: isDisabled}">
```

```
<section *ngIf="showSection">
```

```
<li *ngFor="let item of list">
```

```
<button (click)="deleteHero()">Delete hero</button>
```

```
<input [(ngModel)]="userName">
```

```
<video #movieplayer ...>
  <button (click)="movieplayer.play()">
</video>
```

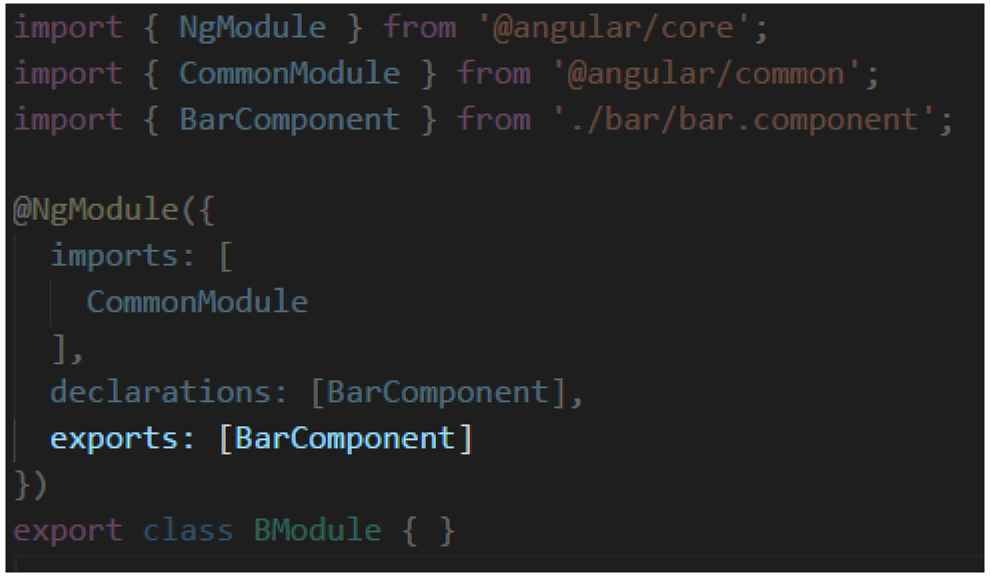
LIFECYCLE-HOOKS

- **ngOnChanges**: llamado cada vez que un input cambia de valor.
- **ngOnInit**: llamado despues del primer ngOnChanges.
- **ngOnDestroy**: antes de que el componente sea eliminado.

- **ngDoCheck**, **ngAfterContentInit**, **ngAfterContentChecked**, **ngAfterViewInit**, **ngAfterViewChecked**, …

EXPORTANTO COMPONENTES

- Para poder usar componentes fuera de un módulo, estos tienen que ser exportados.

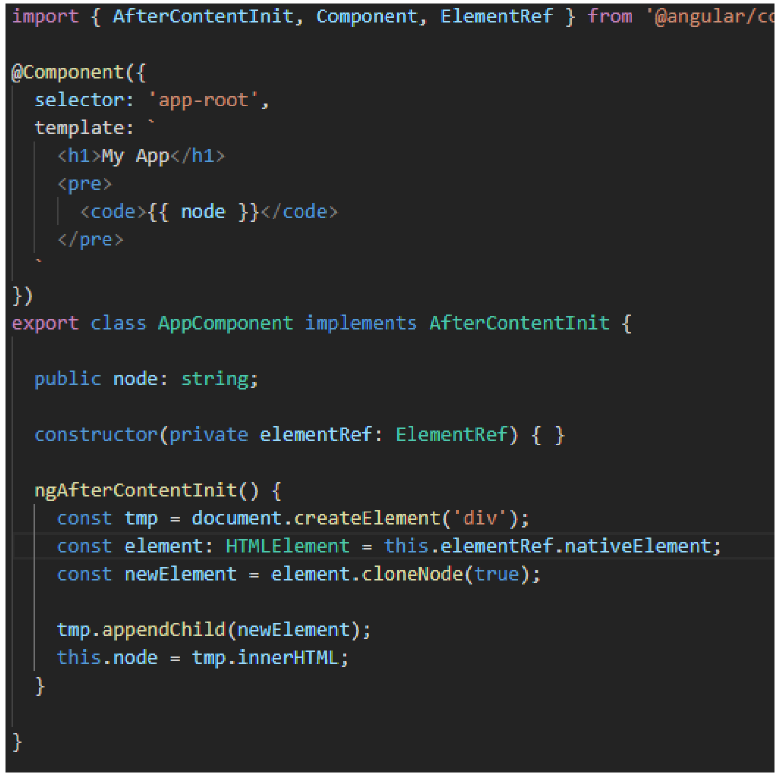


```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { BarComponent } from './bar/bar.component';

@NgModule({
  imports: [
    CommonModule
  ],
  declarations: [BarComponent],
  exports: [BarComponent]
})
export class BModule { }
```

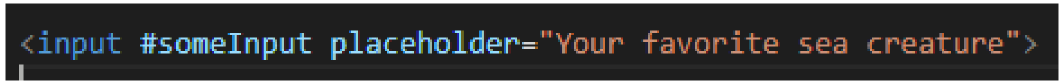
plain concepts

ELEMENT REF

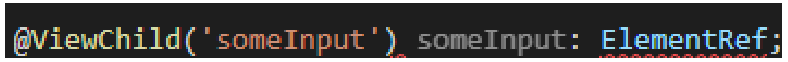
- Proporciona acceso al elemento nativo asociado (en el caso de navegador, elementos del DOM).



```
import { AfterContentInit, Component, ElementRef } from '@angular/co

@Component({
  selector: 'app-root',
  template: `
    <h1>My App</h1>
    <pre>
      <code>{{ node }}</code>
    </pre>
  `
})
export class AppComponent implements AfterContentInit {

  public node: string;

  constructor(private elementRef: ElementRef) { }

  ngAfterContentInit() {
    const tmp = document.createElement('div');
    const element: HTMLElement = this.elementRef.nativeElement;
    const newElement = element.cloneNode(true);

    tmp.appendChild(newElement);
    this.node = tmp.innerHTML;
  }

}
```



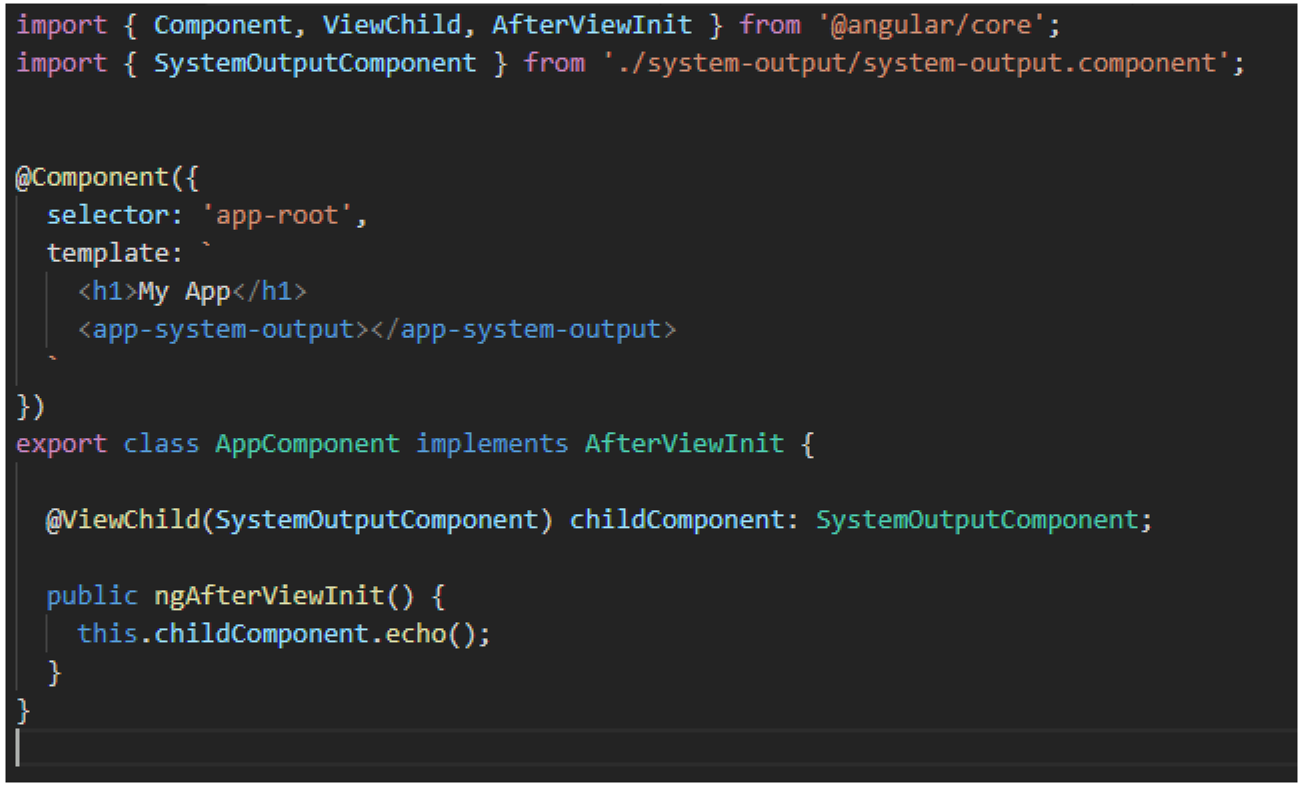
```
<input #someInput placeholder="Your favorite sea creature">
```



```
@ViewChild('someInput') someInput: ElementRef;
```

VIEWCHILD, VIEWCHILDREN

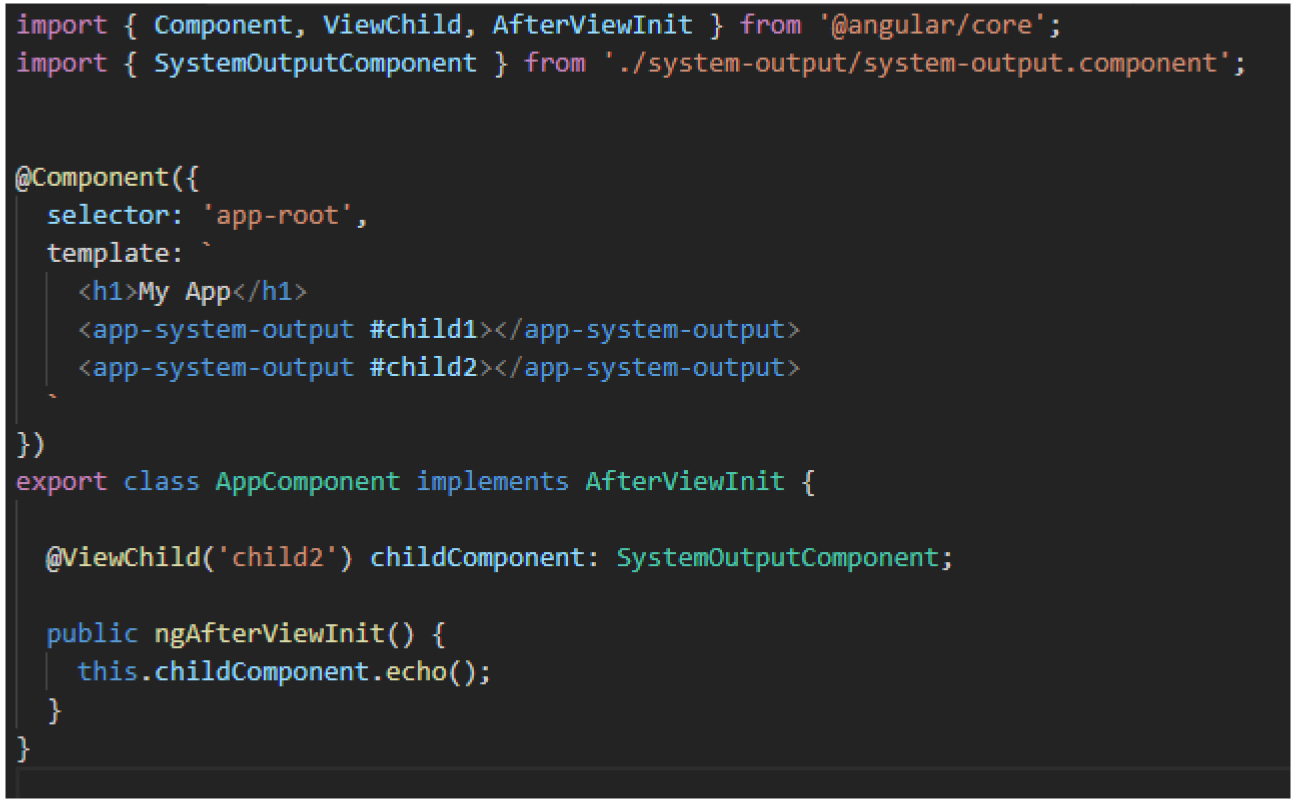
- Permite acceder a componente(s) hijo(s) a través del nombre del componente.



```
import { Component, ViewChild, AfterViewInit } from '@angular/core';
import { SystemOutputComponent } from './system-output/system-output.component';


@Component({
  selector: 'app-root',
  template: `
    <h1>My App</h1>
    <app-system-output></app-system-output>
  `
})
export class AppComponent implements AfterViewInit {

  @ViewChild(SystemOutputComponent) childComponent: SystemOutputComponent;

  public ngAfterViewInit() {
    this.childComponent.echo();
  }
}
```

VIEWCHILD, VIEWCHILDREN

- Permite acceder a componente(s) hijo(s) a través de una *template* variable

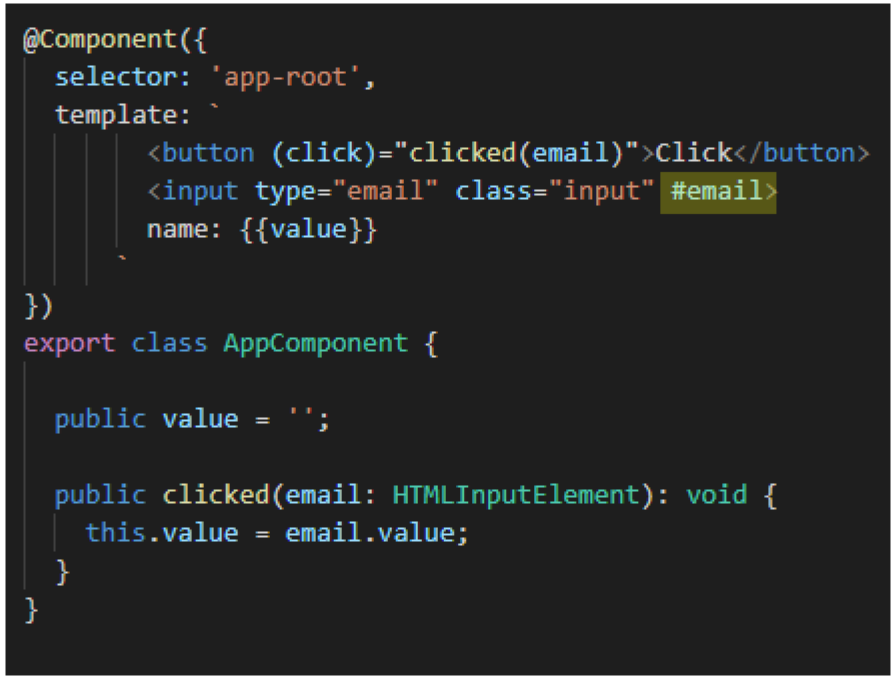


```
import { Component, ViewChild, AfterViewInit } from '@angular/core';
import { SystemOutputComponent } from './system-output/system-output.component';


@Component({
  selector: 'app-root',
  template: `
    <h1>My App</h1>
    <app-system-output #child1></app-system-output>
    <app-system-output #child2></app-system-output>
  `
})
export class AppComponent implements AfterViewInit {

  @ViewChild('child2') childComponent: SystemOutputComponent;

  public ngAfterViewInit() {
    this.childComponent.echo();
  }
}
```

plain concepts

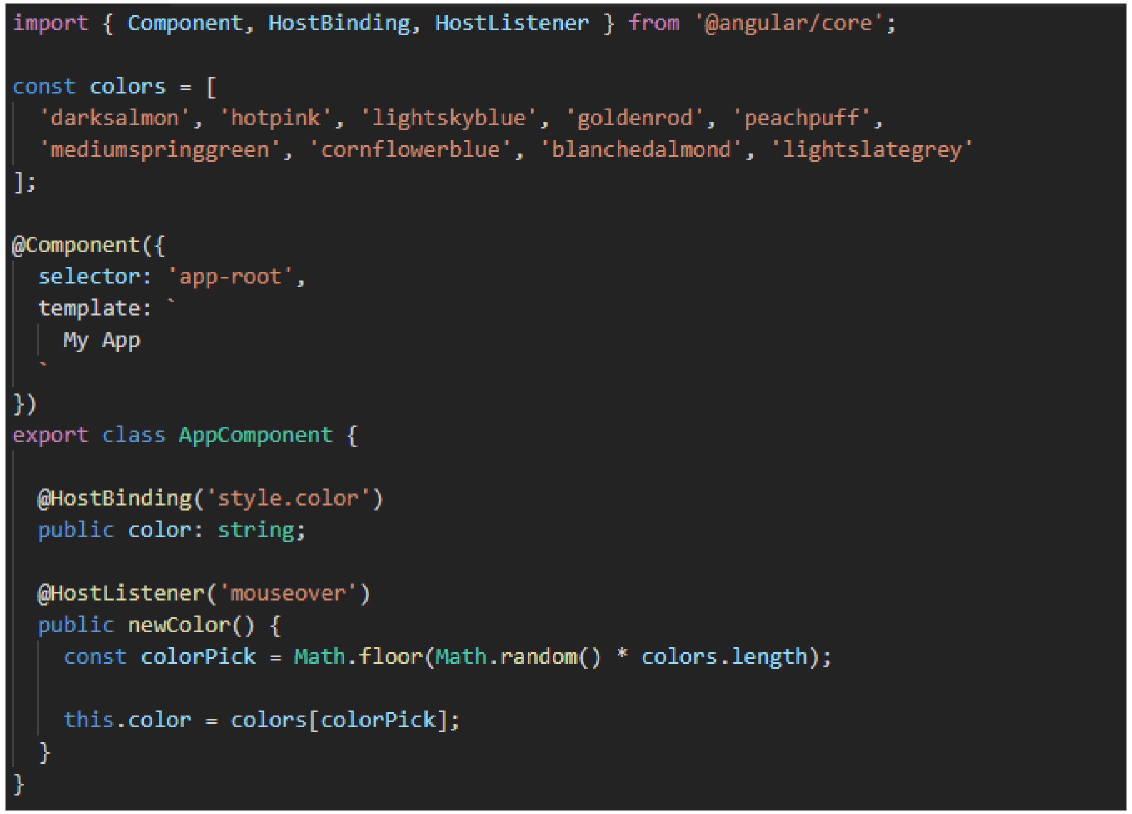
TEMPLATE REF

- Provee acceso en el propio template a los elementos nativos (en el caso de navegador, elementos del DOM)



```
@Component({
  selector: 'app-root',
  template: `
        <button (click)="clicked(email)">Click</button>
        <input type="email" class="input" #email>
        name: {{value}}
      `
})
export class AppComponent {

  public value = '';

  public clicked(email: HTMLInputElement): void {
    this.value = email.value;
  }
}
```

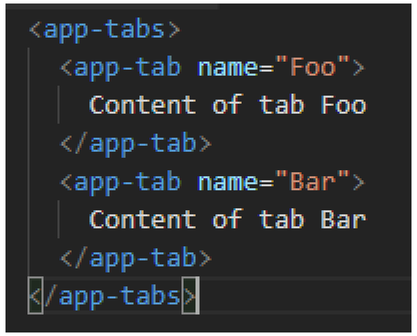
HOSTLISTENER, HOSTBINDING



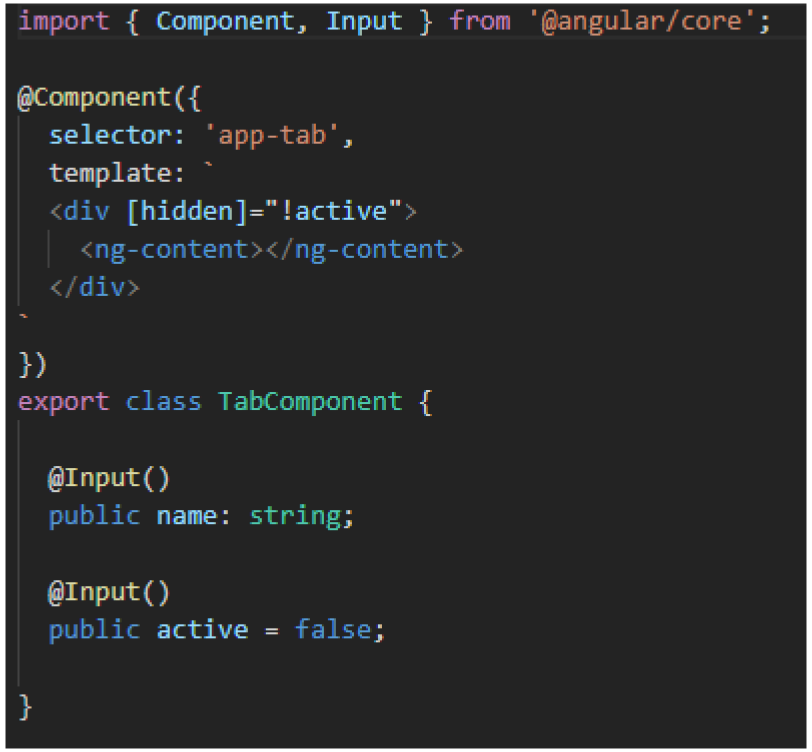
```
import { Component, HostBinding, HostListener } from '@angular/core';

const colors = [
  'darksalmon', 'hotpink', 'lightskyblue', 'goldenrod', 'peachpuff',
  'mediumspringgreen', 'cornflowerblue', 'blanchedalmond', 'lightslategrey'
];

@Component({
  selector: 'app-root',
  template: `
    My App
  `
})
export class AppComponent {

  @HostBinding('style.color')
  public color: string;

  @HostListener('mouseover')
  public newColor() {
    const colorPick = Math.floor(Math.random() * colors.length);

    this.color = colors[colorPick];
  }
}
```

NG-CONTENT

- Permite "proyectar" contenido.



```html
<app-tabs>
  <app-tab name="Foo">
    Content of tab Foo
  </app-tab>
  <app-tab name="Bar">
    Content of tab Bar
  </app-tab>
</app-tabs>
```

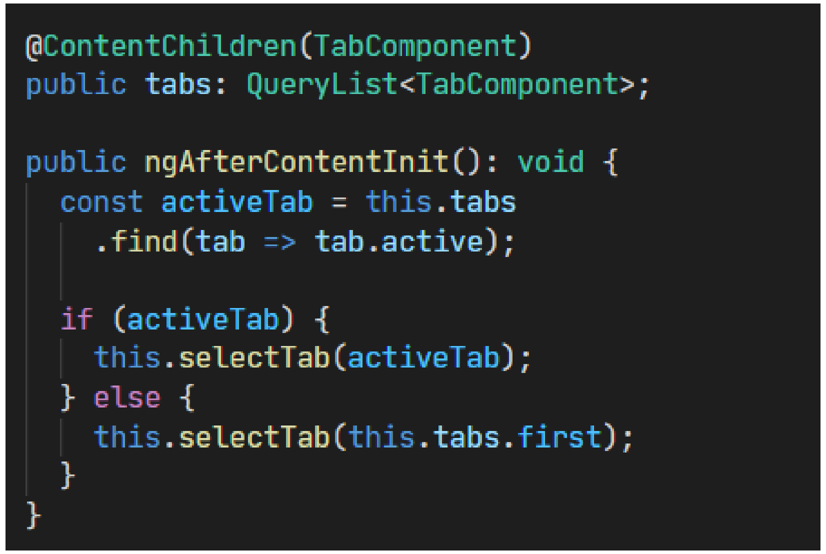


```typescript
import { Component, Input } from '@angular/core';

@Component({
  selector: 'app-tab',
  template: `
  <div [hidden]="!active">
    <ng-content></ng-content>
  </div>
`
})
export class TabComponent {

  @Input()
  public name: string;

  @Input()
  public active = false;

}
```

plain concepts

CONTENTCHILD, CONTENTCHILDREN

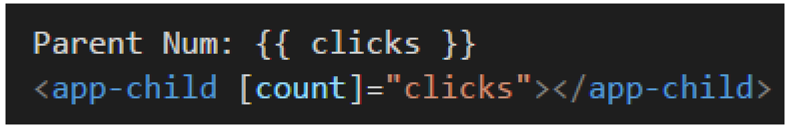
- Permite acceder a componente(s) proyectado(s) a través del nombre del componente o selectores.



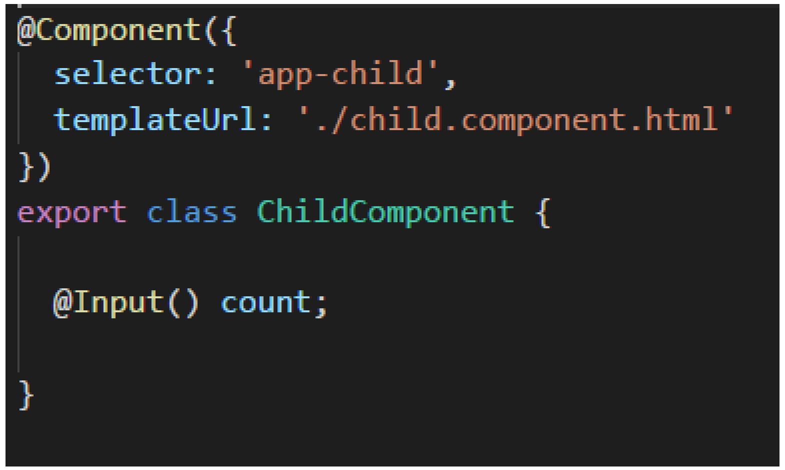
```
@ContentChildren(TabComponent)
public tabs: QueryList<TabComponent>;

public ngAfterContentInit(): void {
  const activeTab = this.tabs
    .find(tab => tab.active);

  if (activeTab) {
    this.selectTab(activeTab);
  } else {
    this.selectTab(this.tabs.first);
  }
}
```

PASANDO DATOS A UN COMPONENTE

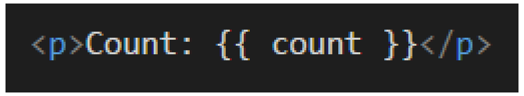
- A través del decorador @Input
- El flujo de detección de cambio es de padre a hijo.



```
Parent Num: {{ clicks }}
<app-child [count]="clicks"></app-child>
```



```
@Component({
  selector: 'app-child',
  templateUrl: './child.component.html'
})
export class ChildComponent {

  @Input() count;

}
```

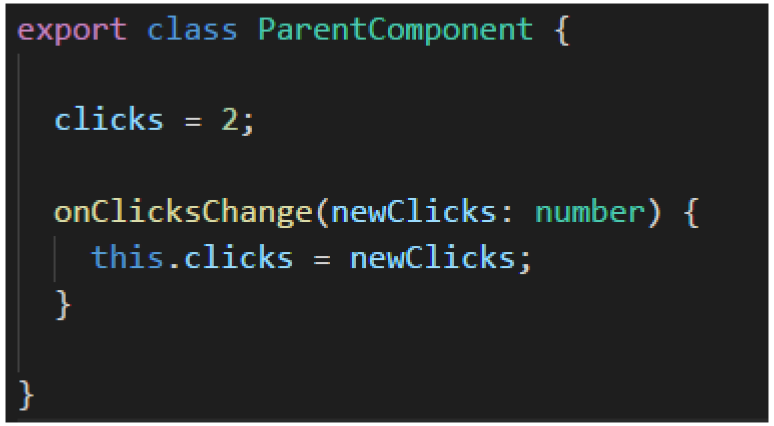


```
<p>Count: {{ count }}</p>
```

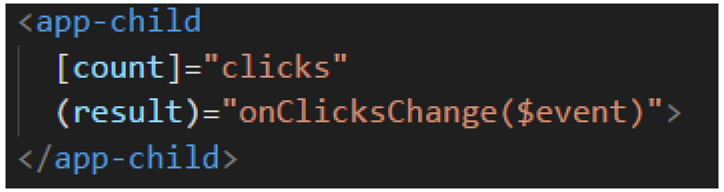
plain concepts

RESPONDIENDO A EVENTOS DE UN COMPONENTE

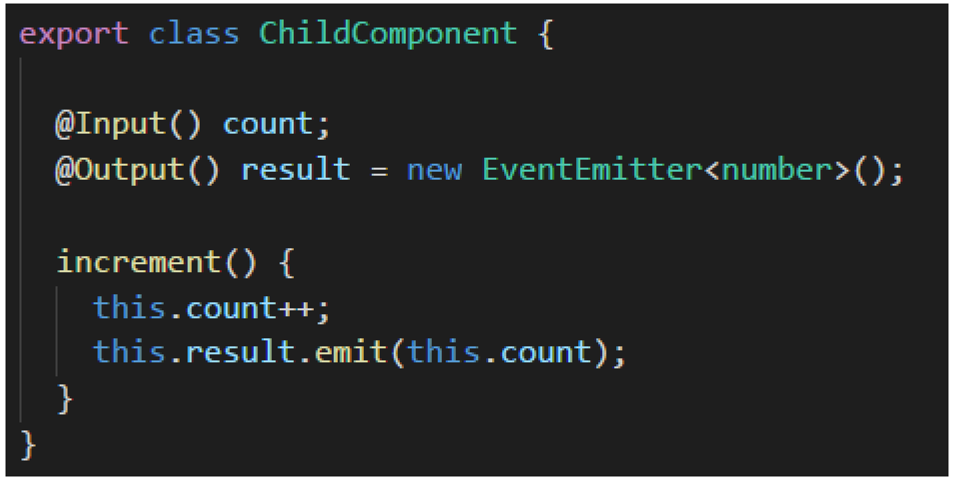
- A través del decorador @Output se define un EventEmitter
- Se define un handler en el padre



```
export class ParentComponent {

  clicks = 2;

  onClicksChange(newClicks: number) {
    this.clicks = newClicks;
  }

}
```



```
<app-child
  [count]="clicks"
  (result)="onClicksChange($event)">
</app-child>
```

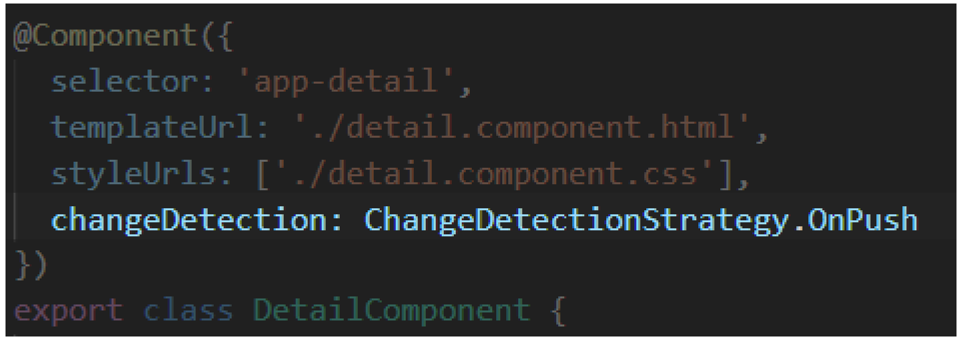


```
export class ChildComponent {

  @Input() count;
  @Output() result = new EventEmitter<number>();

  increment() {
    this.count++;
    this.result.emit(this.count);
  }
}
```

plain concepts

CHANGEDETECTIONSTRATEGY.ONPUSH

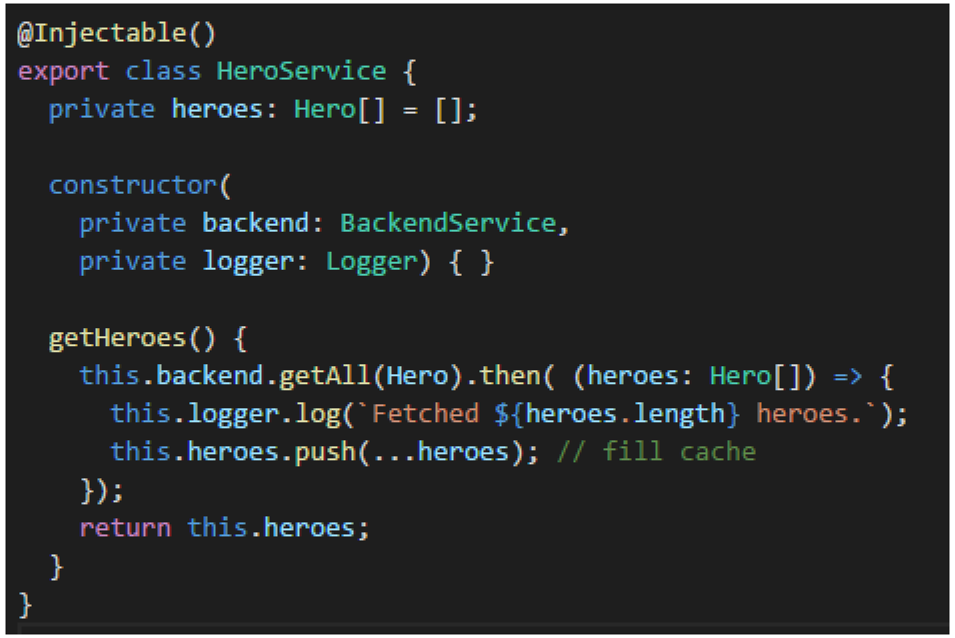
- Un template definido con esta estrategia solo se evaluará nuevamente cuando:
  - La referencia de uno de los inputs cambie.
  - Un eventhandler es emitido.
  - Programáticamente.



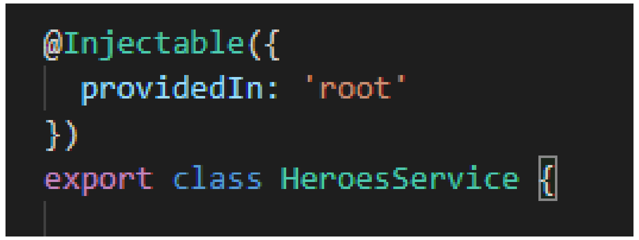
```
@Component({
  selector: 'app-detail',
  templateUrl: './detail.component.html',
  styleUrls: ['./detail.component.css'],
  changeDetection: ChangeDetectionStrategy.OnPush
})
export class DetailComponent {
```

SERVICIOS

- Para Angular los servicios son cualquier clase, valor, función, funcionalidad,… que necesita la aplicación.
- Angular no aporta nada en concreto para desarrollar un servicio.
- Los componentes son consumidores de servicios.



```
@Injectable()
export class HeroService {
  private heroes: Hero[] = [];

  constructor(
    private backend: BackendService,
    private logger: Logger) { }

  getHeroes() {
    this.backend.getAll(Hero).then( (heroes: Hero[]) => {
      this.logger.log(`Fetched ${heroes.length} heroes.`);
      this.heroes.push(...heroes); // fill cache
    });
    return this.heroes;
  }
}
```



```
@Injectable({
  providedIn: 'root'
})
export class HeroesService {
```

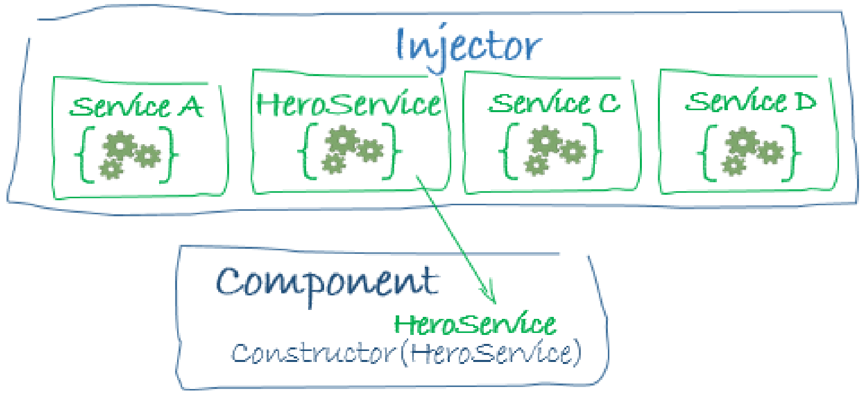
plain concepts

DEPENDENCY INJECTION

- Provee la instancia de una clase con todas las instancias que necesita.

- Para ello:
  - Se fija en los parámetros del constructor.
  - Busca en su contenedor de instancias ya creadas.
  - Si no existe, creará una nueva instancia.

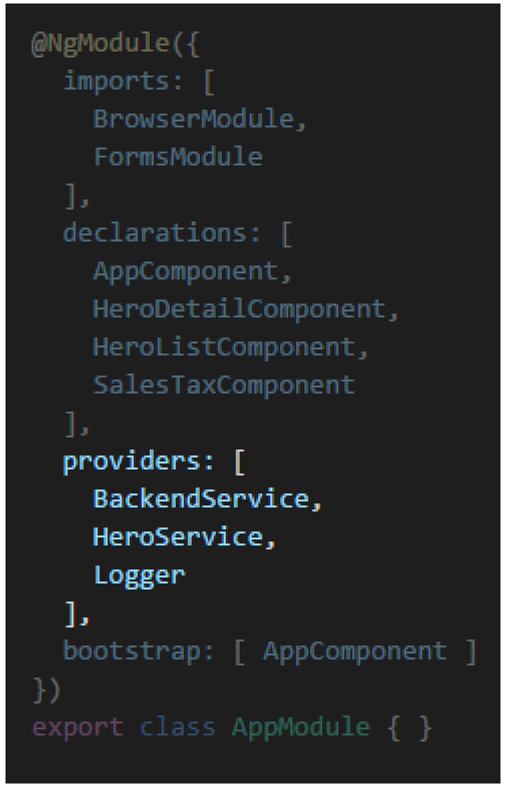
- Los servicios que tienen su propio constructor con dependencias, tienen que declarar el decorador *@Injectable()*



DEPENDENCY INJECTION - REGISTRO

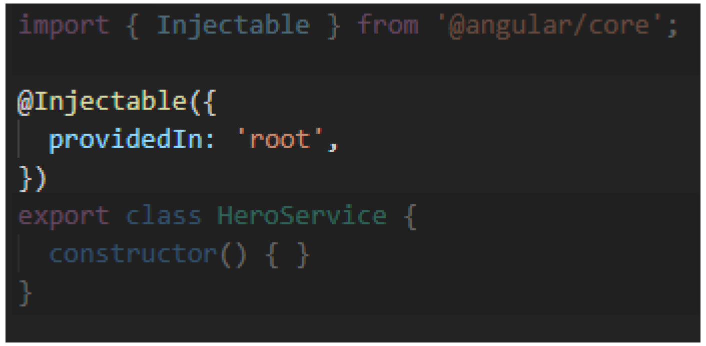
- Para que funcione la inyección de dependencias, hay que registrar todos los servicios. 3 posibilidades:

@NgModule-level



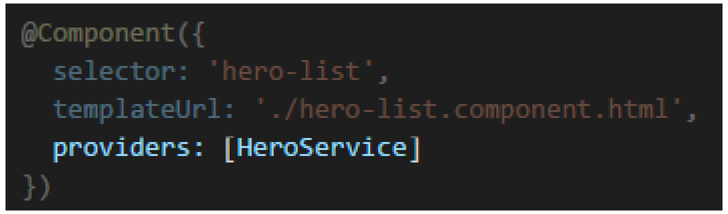
```
@NgModule({
  imports: [
    BrowserModule,
    FormsModule
  ],
  declarations: [
    AppComponent,
    HeroDetailComponent,
    HeroListComponent,
    SalesTaxComponent
  ],
  providers: [
    BackendService,
    HeroService,
    Logger
  ],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
```

@Injectable-level



```
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root',
})
export class HeroService {
  constructor() { }
}
```

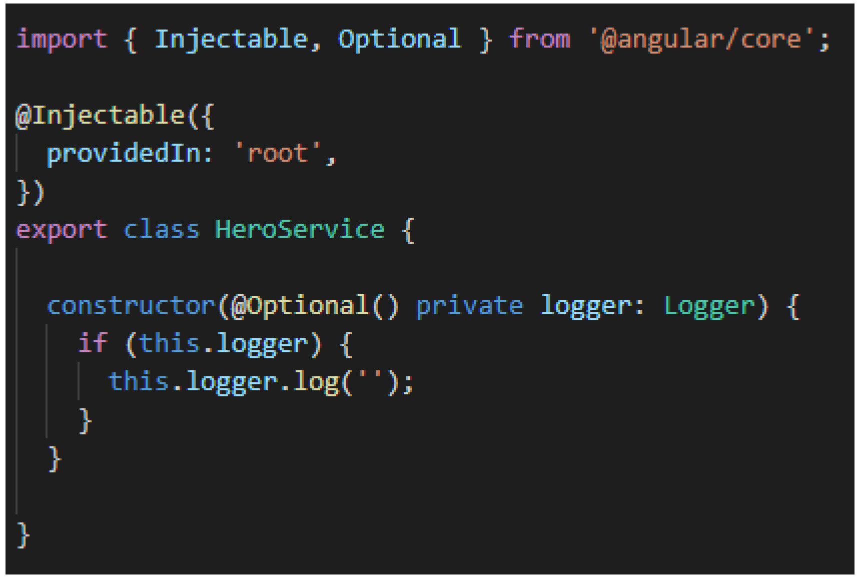
@Component-level



```
@Component({
  selector: 'hero-list',
  templateUrl: './hero-list.component.html',
  providers: [HeroService]
})
```

OPTIONAL DEPENDENCIES

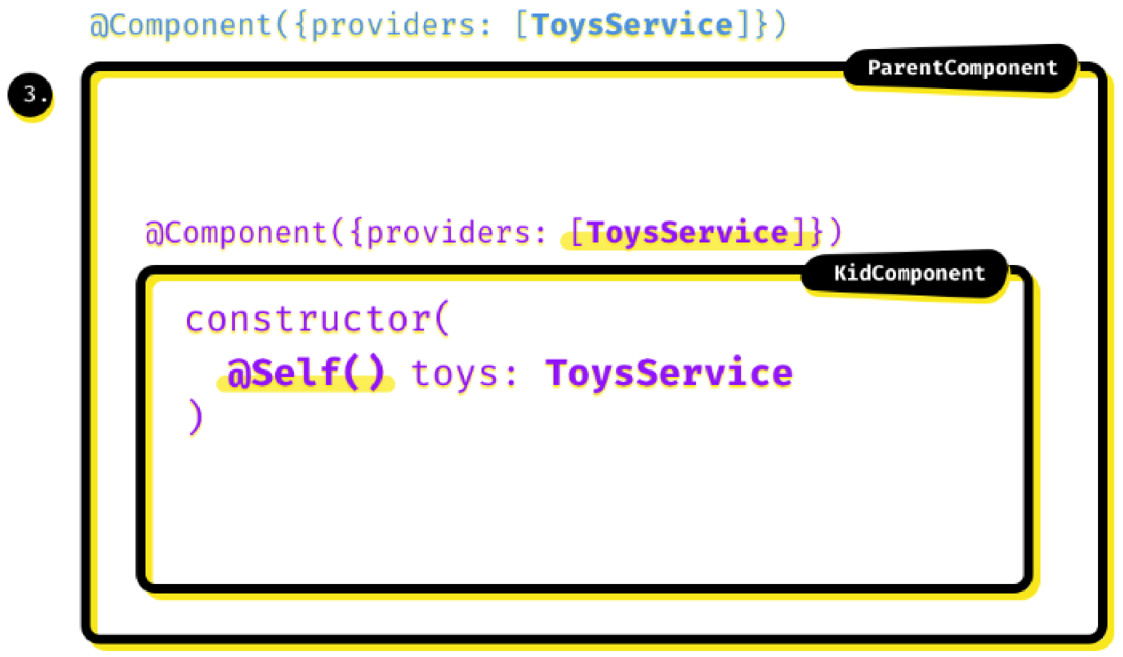
- La anotación *@Optional* nos permite definir dependencias opcionales.



```
import { Injectable, Optional } from '@angular/core';

@Injectable({
  providedIn: 'root',
})
export class HeroService {

  constructor(@Optional() private logger: Logger) {
    if (this.logger) {
      this.logger.log('');
    }
  }

}
```

plain concepts

@SELF



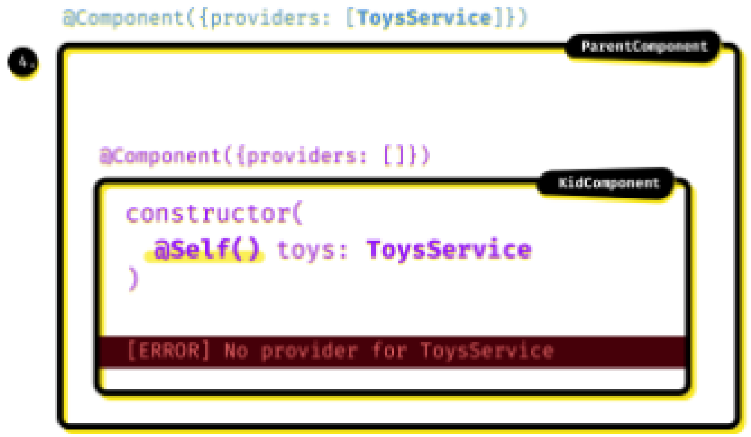
```
@Component({providers: [ToysService]})
```

```
@Component({providers: [ToysService]})
```

```
constructor(
  @Self() toys: ToysService
)
```

Fuente: medium.com

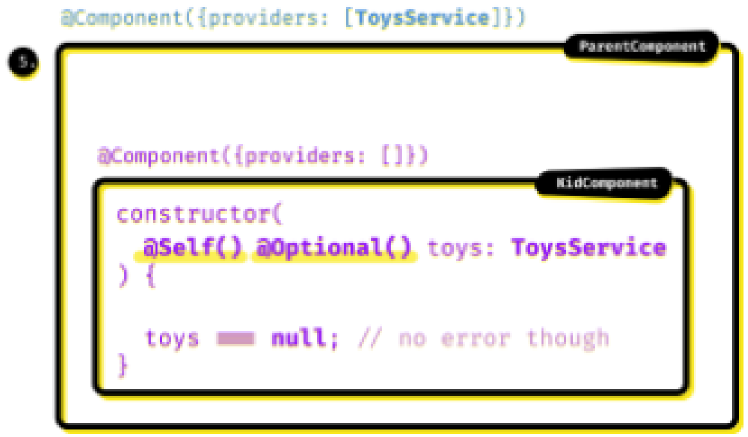
@OPTIONAL


4.

```
@Component({providers: [ToysService]})
```

```
@Component({providers: []})
```

```
constructor(
  @Self() toys: ToysService
)

[ERROR] No provider for ToysService
```


5.

```
@Component({providers: [ToysService]})
```

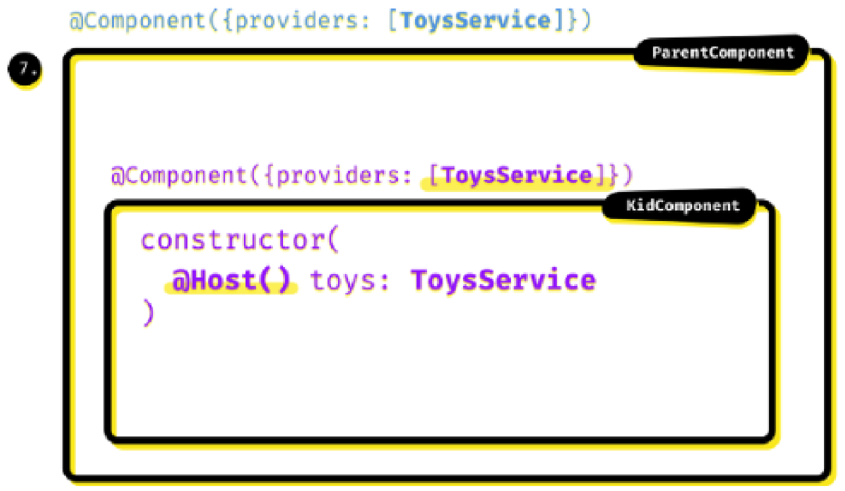
```
@Component({providers: []})
```

```
constructor(
  @Self() @Optional() toys: ToysService
) {

  toys === null; // no error though
}
```

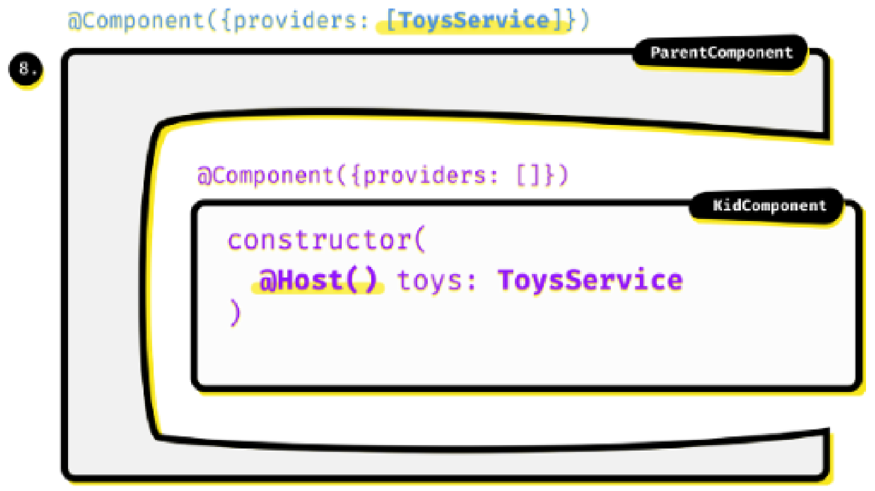
[Fuente: medium.com](medium.com)

plain concepts

@HOST


@Component({providers: [ToysService]})
ParentComponent
7.
@Component({providers: [ToysService]})
KidComponent

```
constructor(
  @Host() toys: ToysService
)
```


@Component({providers: [ToysService]})
ParentComponent
8.
@Component({providers: []})
KidComponent

```
constructor(
  @Host() toys: ToysService
)
```

Fuente: medium.com

TIPOS DE PROVEEDORES

- Normal

```
providers: [Logger],
```

```
providers: [{ provide: Logger, useClass: Logger }],
```

- Alternative class

```
class BetterLogger extends Logger {

}
```

```
providers: [{ provide: Logger, useClass: BetterLogger }],
```

- Aliased class

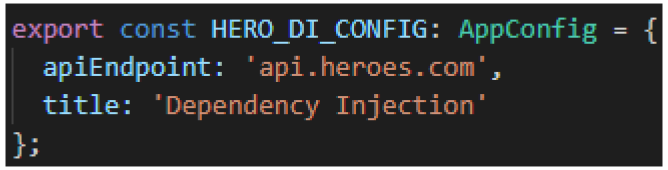
```
[NewLogger,
  // Not aliased! Creates two instances of `NewLogger`
  { provide: OldLogger, useClass: NewLogger }],
```

```
[NewLogger,
  // Alias OldLogger w/ reference to NewLogger
  { provide: OldLogger, useExisting: NewLogger }],
```

plain concepts

TIPOS DE PROVEEDORES

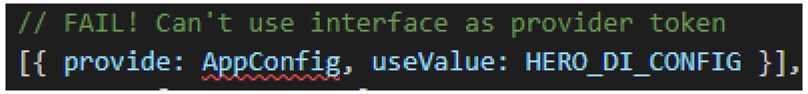
- Non-class:



```
export const HERO_DI_CONFIG: AppConfig = {
  apiEndpoint: 'api.heroes.com',
  title: 'Dependency Injection'
};
```

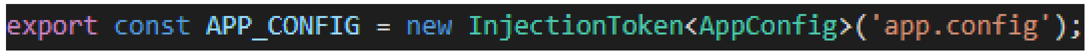
App.config.ts

Las interfaces en Typescript no valen para definir la dependencia



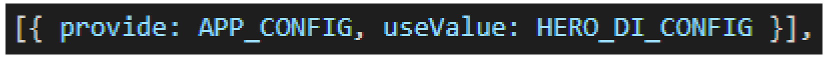
```
// FAIL! Can't use interface as provider token
[{ provide: AppConfig, useValue: HERO_DI_CONFIG }],
```

En su lugar hay que usar una instancia de la clase *InjectionToken*



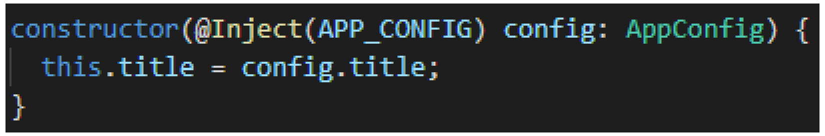
```
export const APP_CONFIG = new InjectionToken<AppConfig>('app.config');
```

App.config.ts



```
[{ provide: APP_CONFIG, useValue: HERO_DI_CONFIG }],
```

Providers.component.ts

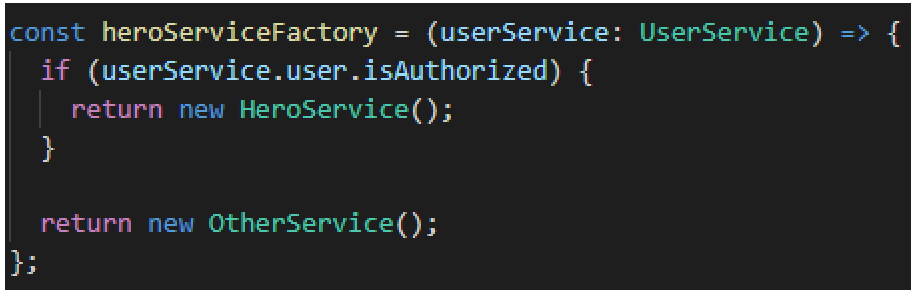


```
constructor(@Inject(APP_CONFIG) config: AppConfig) {
  this.title = config.title;
}
```

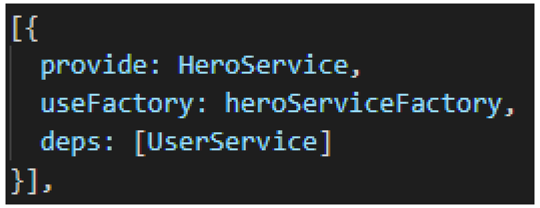
app.component.ts

TIPOS DE PROVEEDORES

- Factory: Si queremos definir dependencias en tiempo de ejecución



```
const heroServiceFactory = (userService: UserService) => {
  if (userService.user.isAuthorized) {
    return new HeroService();
  }

  return new OtherService();
};
```

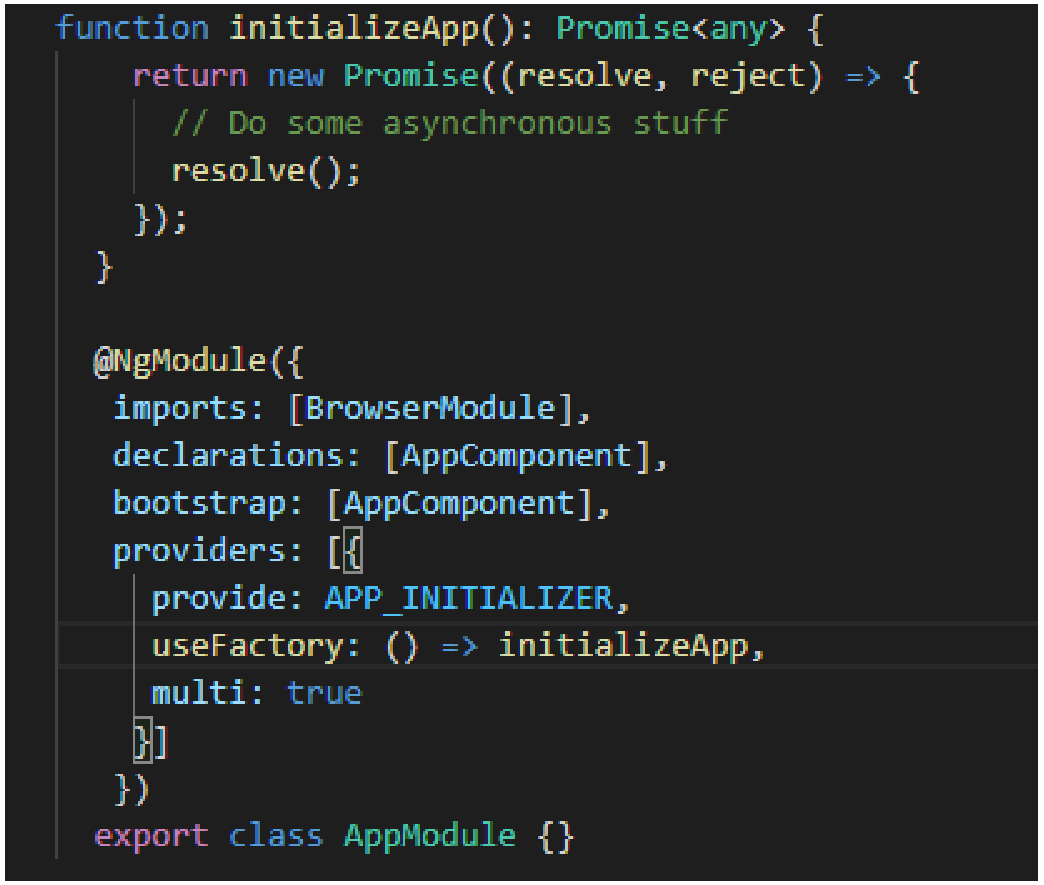


```
[{
  provide: HeroService,
  useFactory: heroServiceFactory,
  deps: [UserService]
}],
```

TOKENS PREDEFINIDOS

- PLATFORM_INITIALIZER: Se invoca el callback cuando una Plataforma se inicia.
- APP_BOOTSTRAP_LISTENER: El callback es invocado para cada componente que se inicia. La función de handler recibe la instancia ComponentRef del componente iniciado.
- **APP_INITIALIZER**: Se invoca el callback antes de que la aplicación se inicie. Todos los inicializadores registrados pueden devolver una Promesa y todas las promesas deben ser resueltas antes de que la aplicación se inicie. Si uno falla, la aplicación no se inicia.

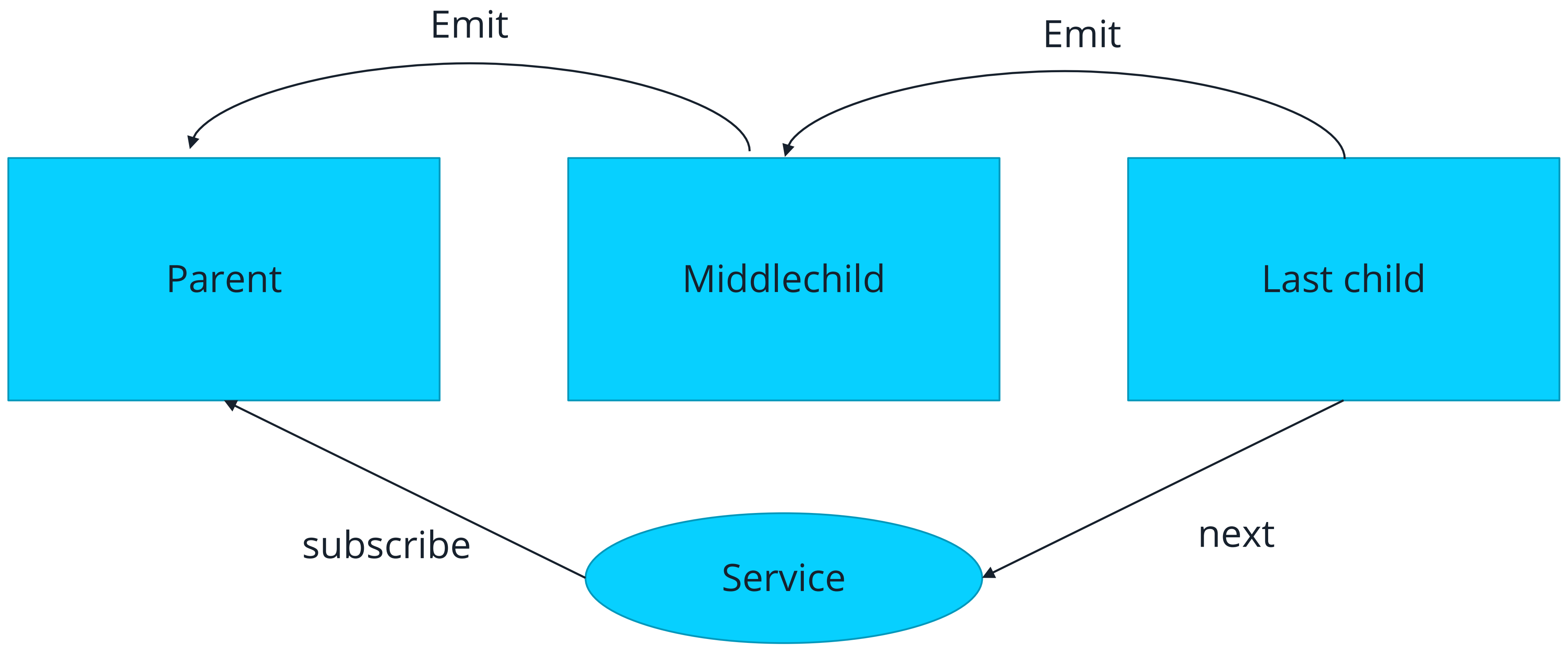
APP_INITIALIZER



```
function initializeApp(): Promise<any> {
    return new Promise((resolve, reject) => {
      // Do some asynchronous stuff
      resolve();
    });
  }

  @NgModule({
   imports: [BrowserModule],
   declarations: [AppComponent],
   bootstrap: [AppComponent],
   providers: [{
     provide: APP_INITIALIZER,
     useFactory: () => initializeApp,
     multi: true
    }]
   })
  export class AppModule {}
```

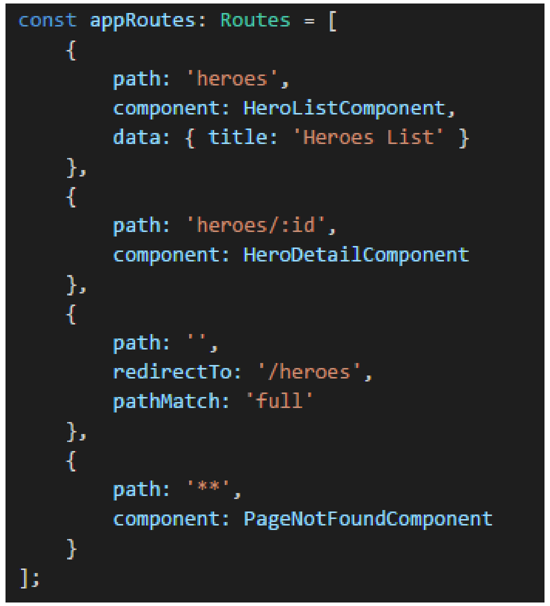
plain concepts

BROADCAST SERVICE

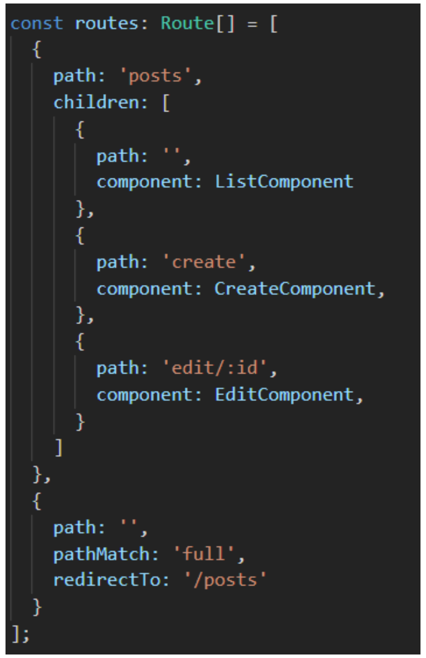


ROUTER - CONFIGURACIÓN

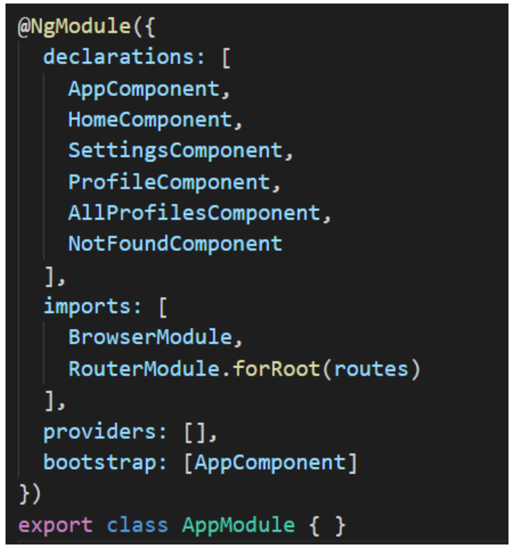
- Cada ruta se mapea a un componente.
- Se pueden definir parámetros en la ruta (ej.: heroes/**:id**)
- Se puede pasar datos arbitrarios a través del objeto **data**
- Se puede crear una jerarquía de rutas



```
const appRoutes: Routes = [
    {
        path: 'heroes',
        component: HeroListComponent,
        data: { title: 'Heroes List' }
    },
    {
        path: 'heroes/:id',
        component: HeroDetailComponent
    },
    {
        path: '',
        redirectTo: '/heroes',
        pathMatch: 'full'
    },
    {
        path: '**',
        component: PageNotFoundComponent
    }
];
```

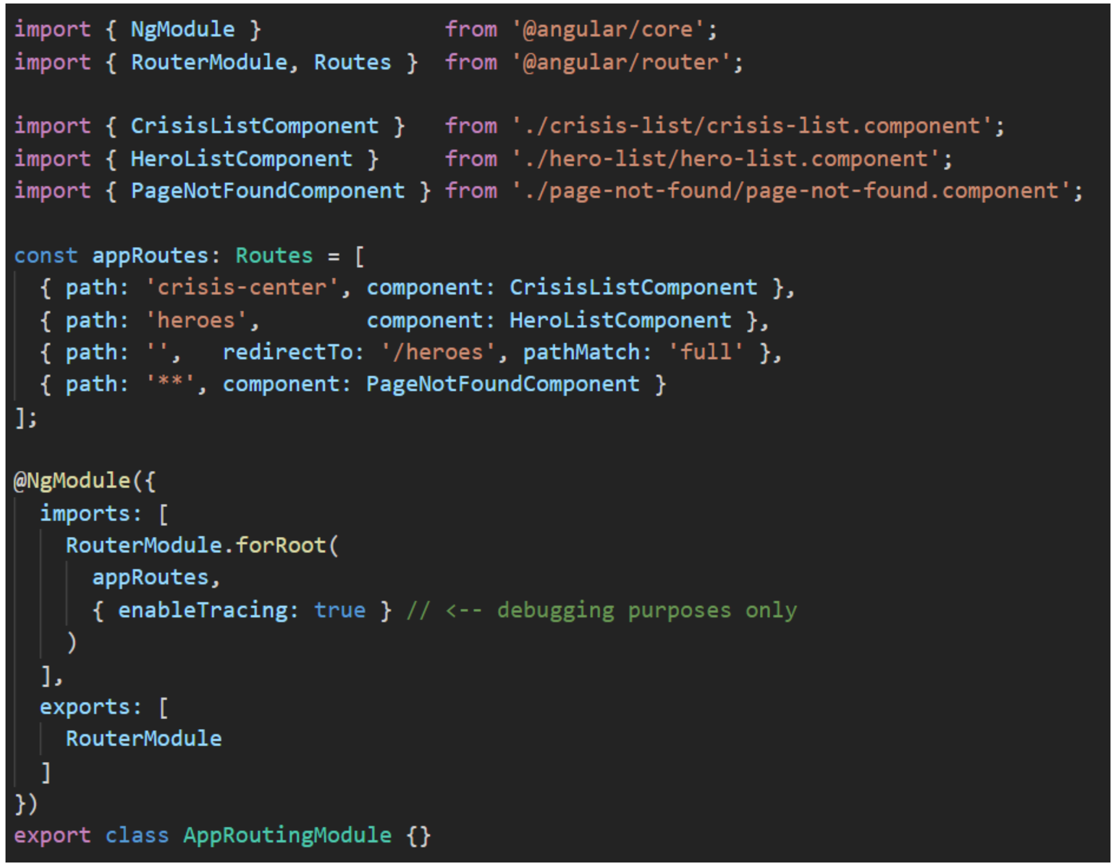


```
const routes: Route[] = [
  {
    path: 'posts',
    children: [
      {
        path: '',
        component: ListComponent
      },
      {
        path: 'create',
        component: CreateComponent,
      },
      {
        path: 'edit/:id',
        component: EditComponent,
      }
    ]
  },
  {
    path: '',
    pathMatch: 'full',
    redirectTo: '/posts'
  }
];
```

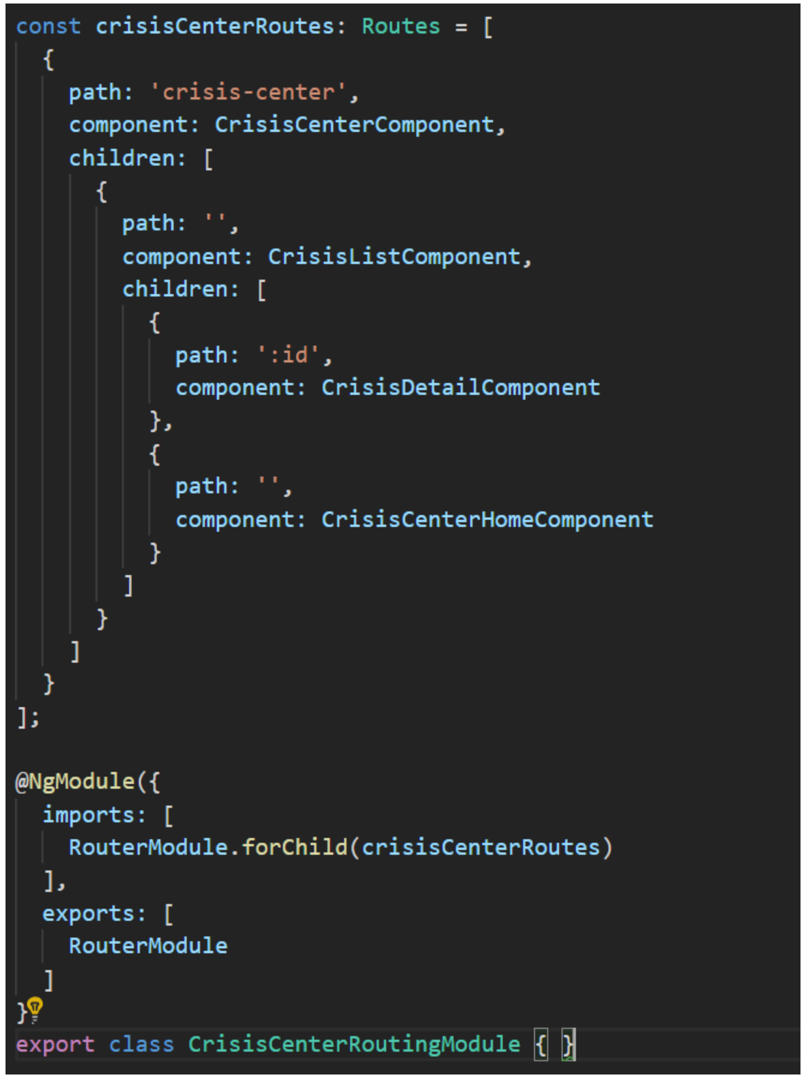


```
@NgModule({
  declarations: [
    AppComponent,
    HomeComponent,
    SettingsComponent,
    ProfileComponent,
    AllProfilesComponent,
    NotFoundComponent
  ],
  imports: [
    BrowserModule,
    RouterModule.forRoot(routes)
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

ROUTER MODULE



```
import { NgModule }              from '@angular/core';
import { RouterModule, Routes }  from '@angular/router';

import { CrisisListComponent }   from './crisis-list/crisis-list.component';
import { HeroListComponent }     from './hero-list/hero-list.component';
import { PageNotFoundComponent } from './page-not-found/page-not-found.component';

const appRoutes: Routes = [
  { path: 'crisis-center', component: CrisisListComponent },
  { path: 'heroes',        component: HeroListComponent },
  { path: '',   redirectTo: '/heroes', pathMatch: 'full' },
  { path: '**', component: PageNotFoundComponent }
];

@NgModule({
  imports: [
    RouterModule.forRoot(
      appRoutes,
      { enableTracing: true } // <-- debugging purposes only
    )
  ],
  exports: [
    RouterModule
  ]
})
export class AppRoutingModule {}
```

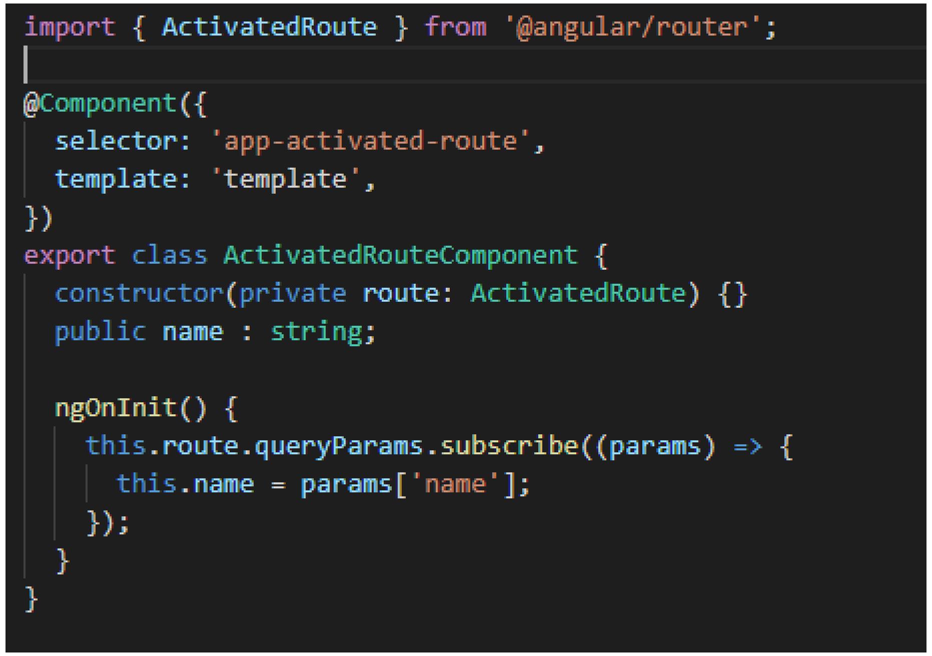


```
const crisisCenterRoutes: Routes = [
  {
    path: 'crisis-center',
    component: CrisisCenterComponent,
    children: [
      {
        path: '',
        component: CrisisListComponent,
        children: [
          {
            path: ':id',
            component: CrisisDetailComponent
          },
          {
            path: '',
            component: CrisisCenterHomeComponent
          }
        ]
      }
    ]
  }
];

@NgModule({
  imports: [
    RouterModule.forChild(crisisCenterRoutes)
  ],
  exports: [
    RouterModule
  ]
})
export class CrisisCenterRoutingModule { }
```

plain concepts

ACTIVATED ROUTE

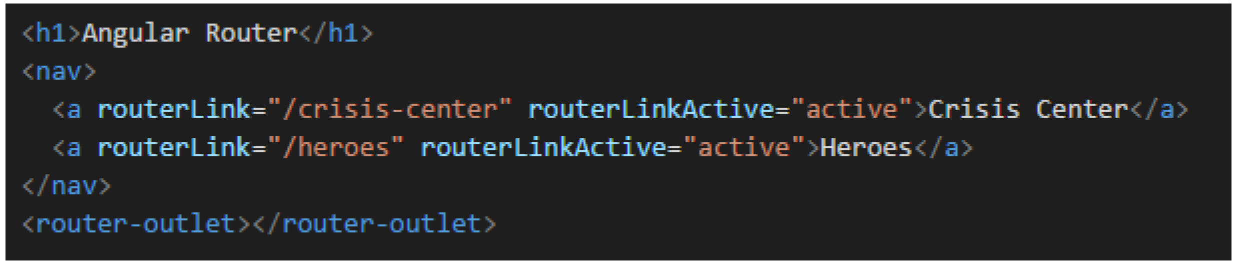
- Permite acceder a información sobre la ruta asociada al componente que está cargado.



```
import { ActivatedRoute } from '@angular/router';

@Component({
  selector: 'app-activated-route',
  template: 'template',
})
export class ActivatedRouteComponent {
  constructor(private route: ActivatedRoute) {}
  public name : string;

  ngOnInit() {
    this.route.queryParams.subscribe((params) => {
      this.name = params['name'];
    });
  }
}
```

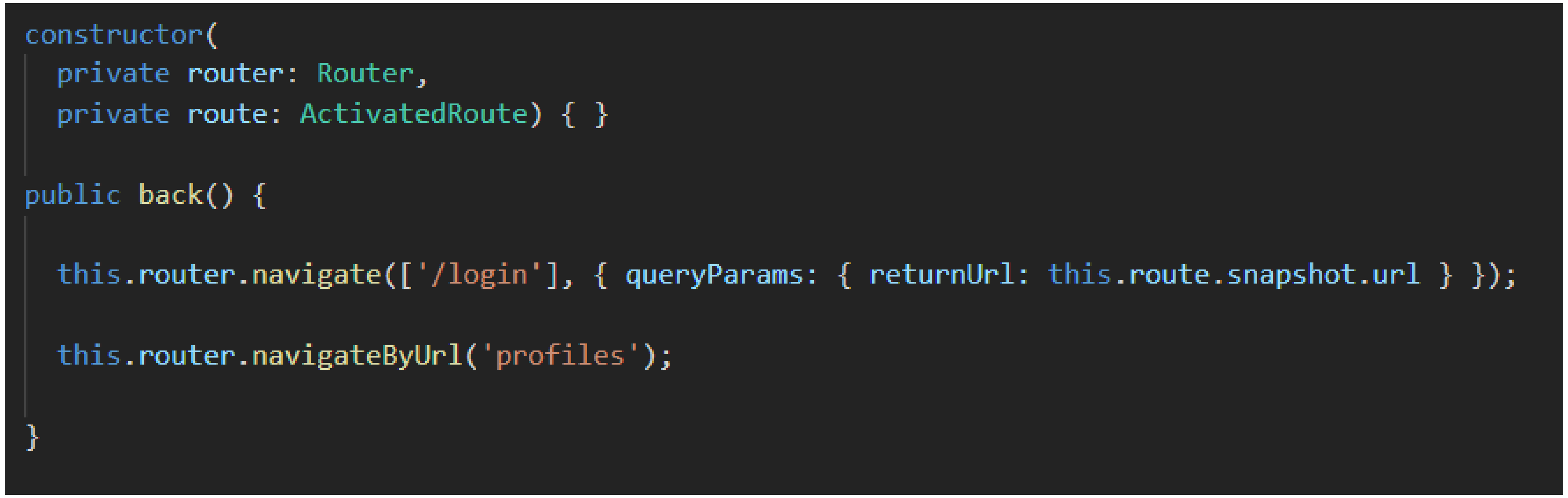
ROUTER – LINKS & OUTLET

- Se puede navegar a componentes desde los templates mediante **routerLinks**.
- Los templates de los componentes resueltos por el router se mostrarán a partir de la directiva **router-outlet.** Puede haber tantos como jerarquías definidos.
- Con la directiva **routerLinkActive** se puede asociar una clase css a la ruta activa.



```html
<h1>Angular Router</h1>
<nav>
  <a routerLink="/crisis-center" routerLinkActive="active">Crisis Center</a>
  <a routerLink="/heroes" routerLinkActive="active">Heroes</a>
</nav>
<router-outlet></router-outlet>
```

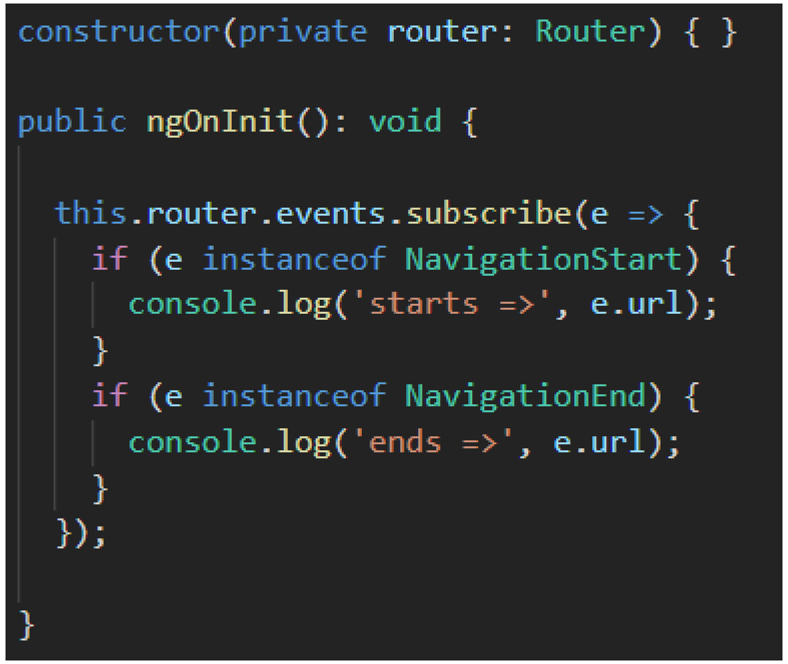
ROUTER - NAVIGATING PROGRAMMATICALLY



```
constructor(
  private router: Router,
  private route: ActivatedRoute) { }

public back() {

  this.router.navigate(['/login'], { queryParams: { returnUrl: this.route.snapshot.url } });

  this.router.navigateByUrl('profiles');

}
```

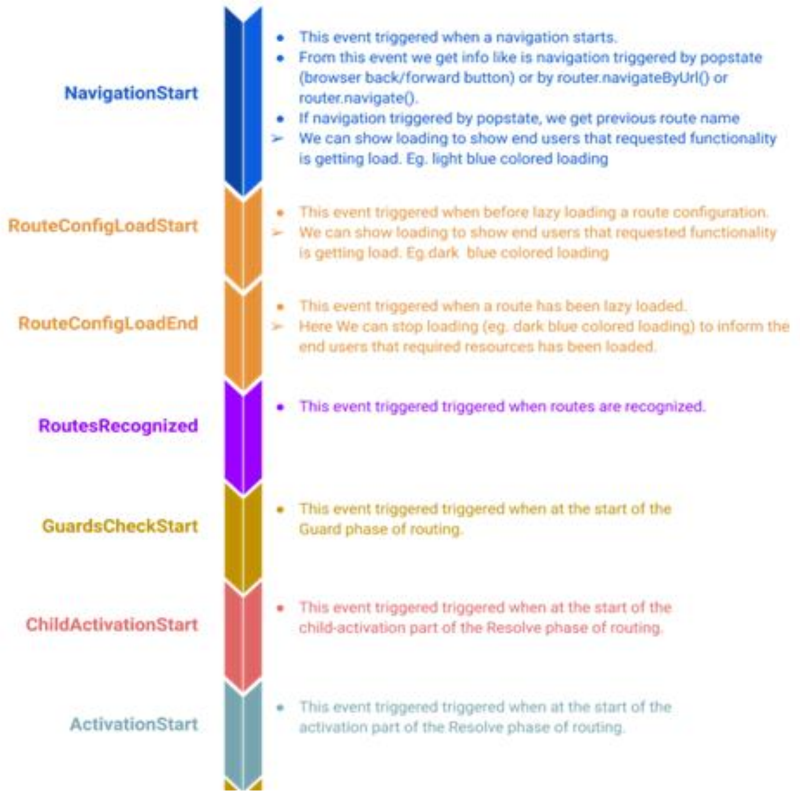
ROUTER - EVENTOS

- NavigationStart,
- RouteConfigLoadStart,
- RouteConfigLoadEnd,
- RoutesRecognized,
- GuardsCheckStart,
- ChildActivationStart,
- ActivationStart,
- GuardsCheckEnd,
- ResolveStart,
- ResolveEnd,
- ActivationEnd
- ChildActivationEnd
- NavigationEnd,
- NavigationCancel,
- NavigationError
- Scroll



```
constructor(private router: Router) { }

public ngOnInit(): void {

  this.router.events.subscribe(e => {
    if (e instanceof NavigationStart) {
      console.log('starts =>', e.url);
    }
    if (e instanceof NavigationEnd) {
      console.log('ends =>', e.url);
    }
  });

}
```

ROUTER - EVENTOS



NavigationStart
- This event triggered when a navigation starts.
- From this event we get info like is navigation triggered by popstate (browser back/forward button) or by router.navigateByUrl() or router.navigate().
- If navigation triggered by popstate, we get previous route name
- ➢ We can show loading to show end users that requested functionality is getting load. Eg. light blue colored loading

RouteConfigLoadStart
- This event triggered when before lazy loading a route configuration.
- ➢ We can show loading to show end users that requested functionality is getting load. Eg.dark blue colored loading

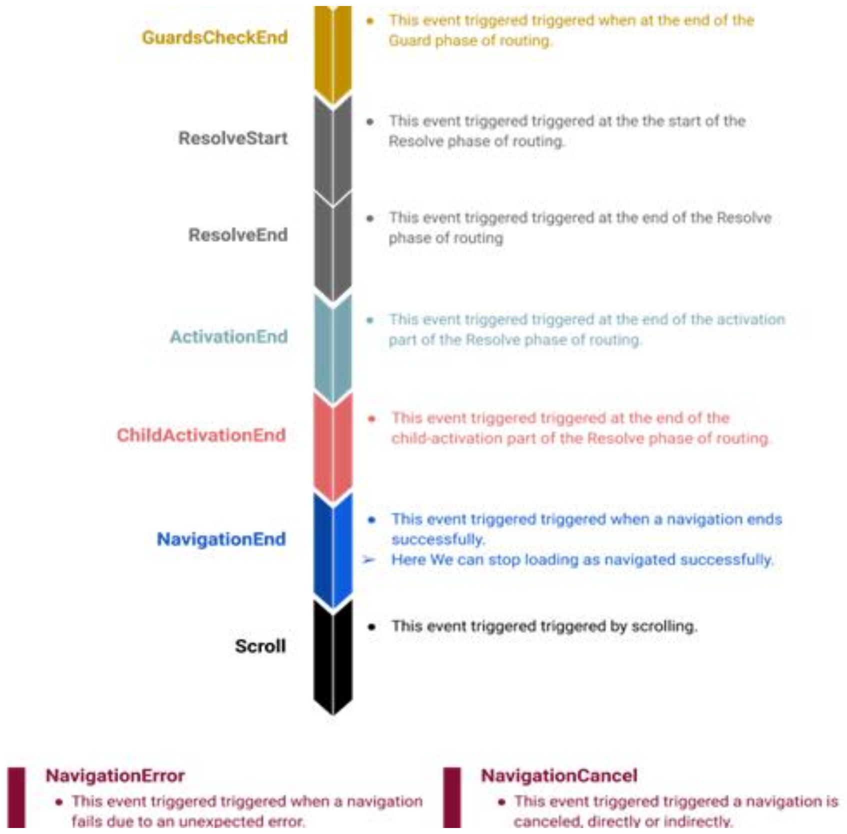
RouteConfigLoadEnd
- This event triggered when a route has been lazy loaded.
- ➢ Here We can stop loading (eg. dark blue colored loading) to inform the end users that required resources has been loaded.

RoutesRecognized
- This event triggered triggered when routes are recognized.

GuardsCheckStart
- This event triggered triggered when at the start of the Guard phase of routing.

ChildActivationStart
- This event triggered triggered when at the start of the child-activation part of the Resolve phase of routing.

ActivationStart
- This event triggered triggered when at the start of the activation part of the Resolve phase of routing.



GuardsCheckEnd
- This event triggered triggered when at the end of the Guard phase of routing.

ResolveStart
- This event triggered triggered at the the start of the Resolve phase of routing.

ResolveEnd
- This event triggered triggered at the end of the Resolve phase of routing

ActivationEnd
- This event triggered triggered at the end of the activation part of the Resolve phase of routing.

ChildActivationEnd
- This event triggered triggered at the end of the child-activation part of the Resolve phase of routing.

NavigationEnd
- This event triggered triggered when a navigation ends successfully.
- ➢ Here We can stop loading as navigated successfully.

Scroll
- This event triggered triggered by scrolling.

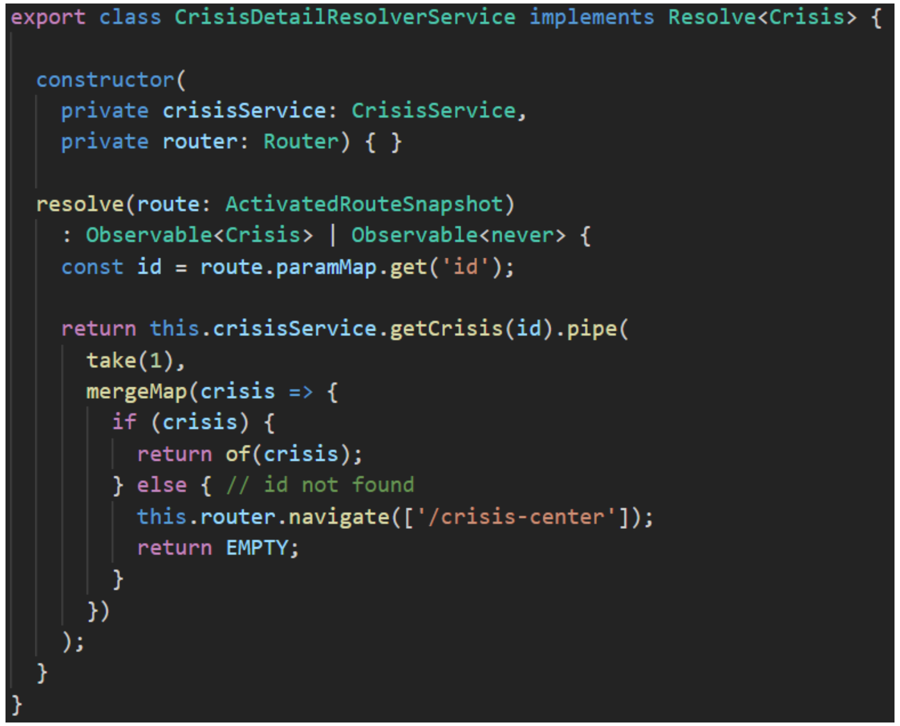
NavigationError
- This event triggered triggered when a navigation fails due to an unexpected error.

NavigationCancel
- This event triggered triggered a navigation is canceled, directly or indirectly.

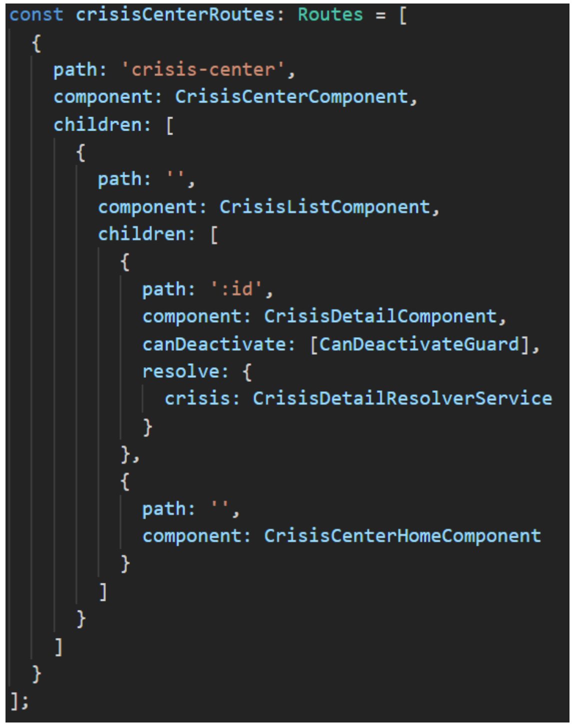
GUARDS

- Permite controlar si el usuario puede navegar a determinado componente
- Si devuelve *true*, se permite la navegación.
- Tipos:
  - *CanActivate*: navegar a una ruta.
  - *CanActivateChild*: navegar a las rutas hijas.
  - *CanDeactivate*: salir de la ruta activa.
  - *Resolve:* permite retrasar la renderización del componente solicitado hasta que se hayan recuperado todos los datos necesarios.
  - …

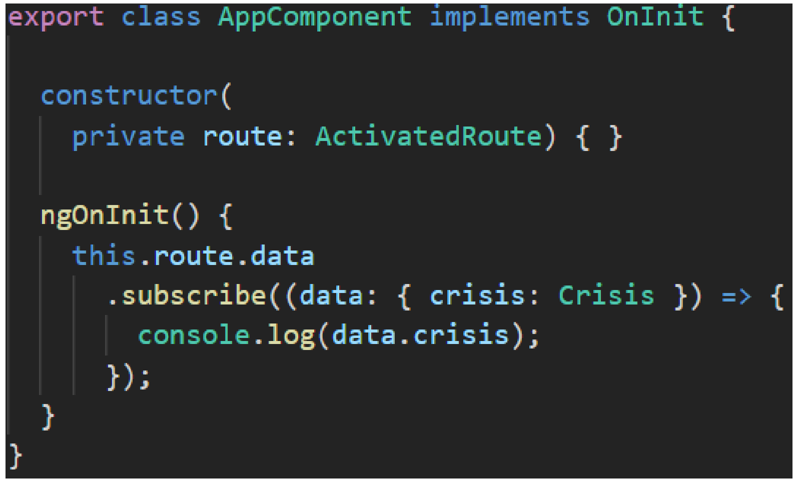
GUARDS - RESOLVE



```typescript
export class CrisisDetailResolverService implements Resolve<Crisis> {

  constructor(
    private crisisService: CrisisService,
    private router: Router) { }

  resolve(route: ActivatedRouteSnapshot)
    : Observable<Crisis> | Observable<never> {
    const id = route.paramMap.get('id');

    return this.crisisService.getCrisis(id).pipe(
      take(1),
      mergeMap(crisis => {
        if (crisis) {
          return of(crisis);
        } else { // id not found
          this.router.navigate(['/crisis-center']);
          return EMPTY;
        }
      })
    );
  }
}
```



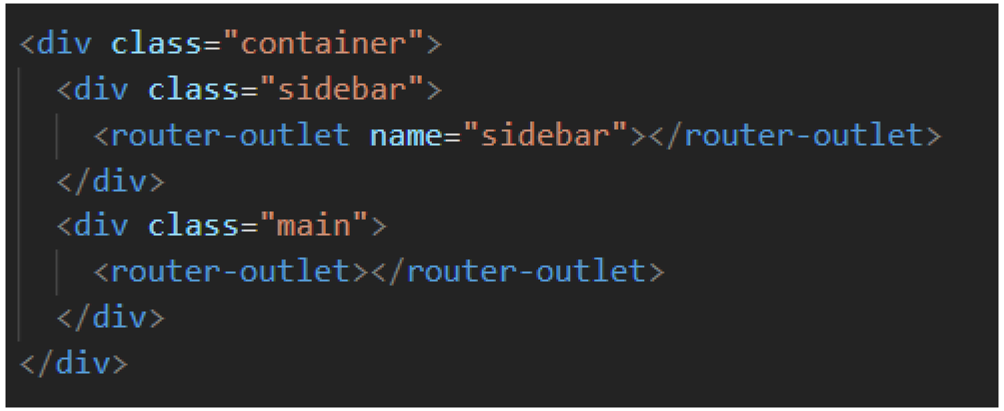
```typescript
const crisisCenterRoutes: Routes = [
  {
    path: 'crisis-center',
    component: CrisisCenterComponent,
    children: [
      {
        path: '',
        component: CrisisListComponent,
        children: [
          {
            path: ':id',
            component: CrisisDetailComponent,
            canDeactivate: [CanDeactivateGuard],
            resolve: {
              crisis: CrisisDetailResolverService
            }
          },
          {
            path: '',
            component: CrisisCenterHomeComponent
          }
        ]
      }
    ]
  }
];
```



```typescript
export class AppComponent implements OnInit {

  constructor(
    private route: ActivatedRoute) { }

  ngOnInit() {
    this.route.data
      .subscribe((data: { crisis: Crisis }) => {
        console.log(data.crisis);
      });
  }
}
```

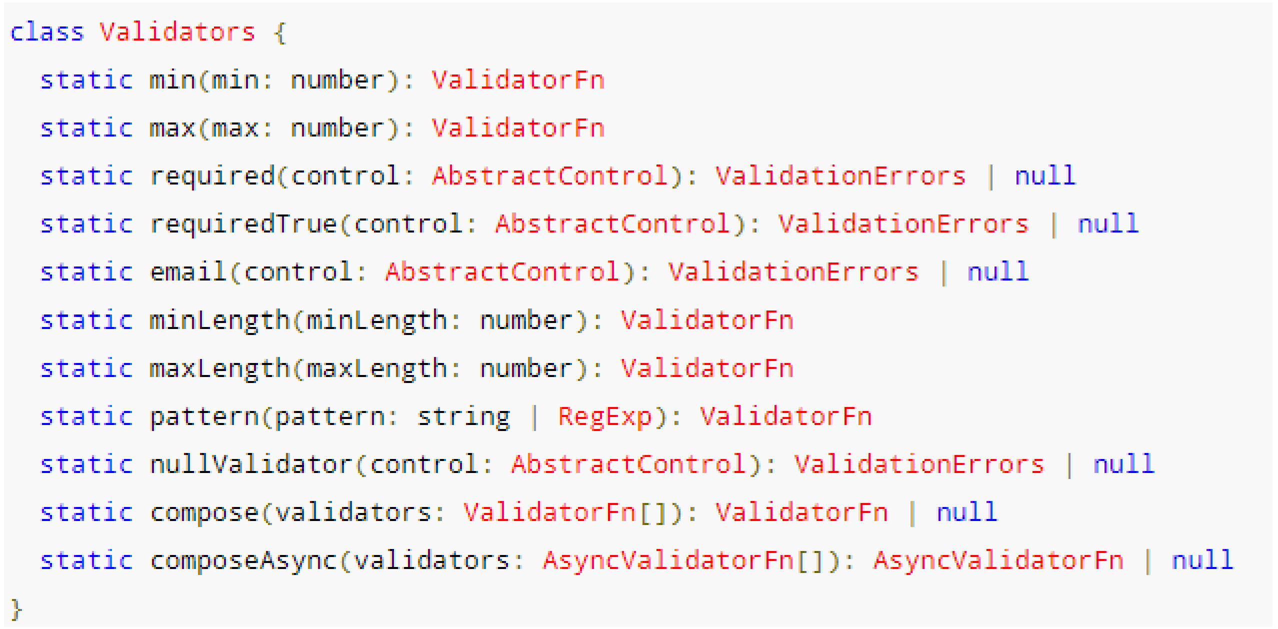
plain concepts

ROUTER – MULTIPLE NAMED OUTLETS



```html
<div class="container">
  <div class="sidebar">
    <router-outlet name="sidebar"></router-outlet>
  </div>
  <div class="main">
    <router-outlet></router-outlet>
  </div>
</div>
```

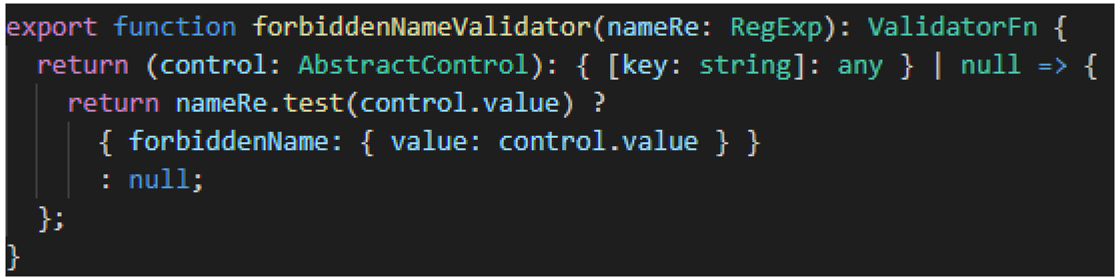
FORMULARIO VALIDACIONES – BUILT-IN



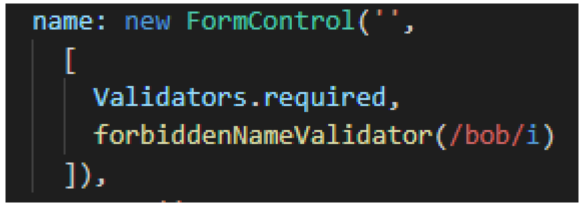
```
class Validators {
  static min(min: number): ValidatorFn
  static max(max: number): ValidatorFn
  static required(control: AbstractControl): ValidationErrors | null
  static requiredTrue(control: AbstractControl): ValidationErrors | null
  static email(control: AbstractControl): ValidationErrors | null
  static minLength(minLength: number): ValidatorFn
  static maxLength(maxLength: number): ValidatorFn
  static pattern(pattern: string | RegExp): ValidatorFn
  static nullValidator(control: AbstractControl): ValidationErrors | null
  static compose(validators: ValidatorFn[]): ValidatorFn | null
  static composeAsync(validators: AsyncValidatorFn[]): AsyncValidatorFn | null
}
```

https://angular.io/api/forms/Validators

FORMULARIO VALIDACIONES - CUSTOM



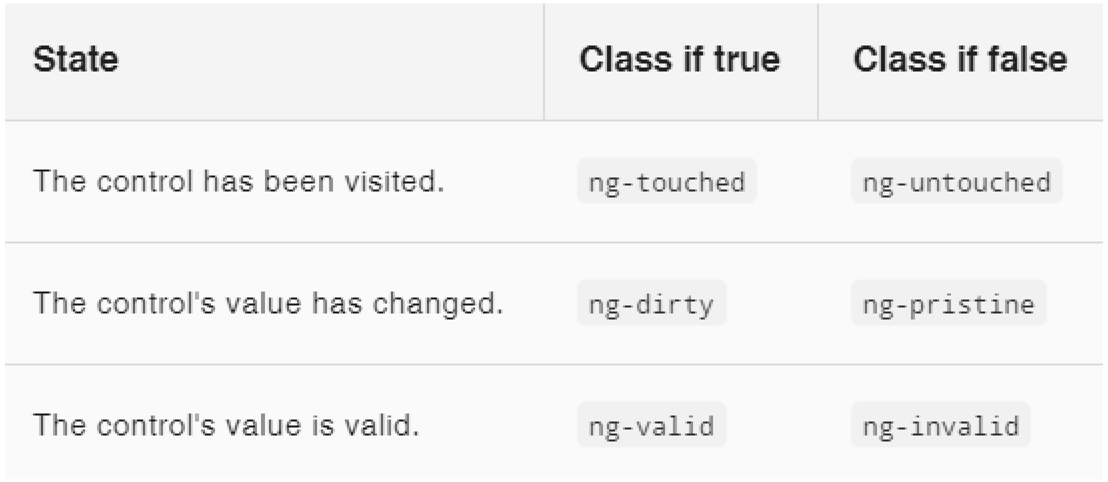
```
export function forbiddenNameValidator(nameRe: RegExp): ValidatorFn {
  return (control: AbstractControl): { [key: string]: any } | null => {
    return nameRe.test(control.value) ?
      { forbiddenName: { value: control.value } }
      : null;
  };
}
```



```
name: new FormControl('',
  [
    Validators.required,
    forbiddenNameValidator(/bob/i)
  ]),
```

CLASES CSS EN BASE A ESTADOS

- Angular añade/elimina clases css a los elementos del formulario y al propio formulario en base a su estado.

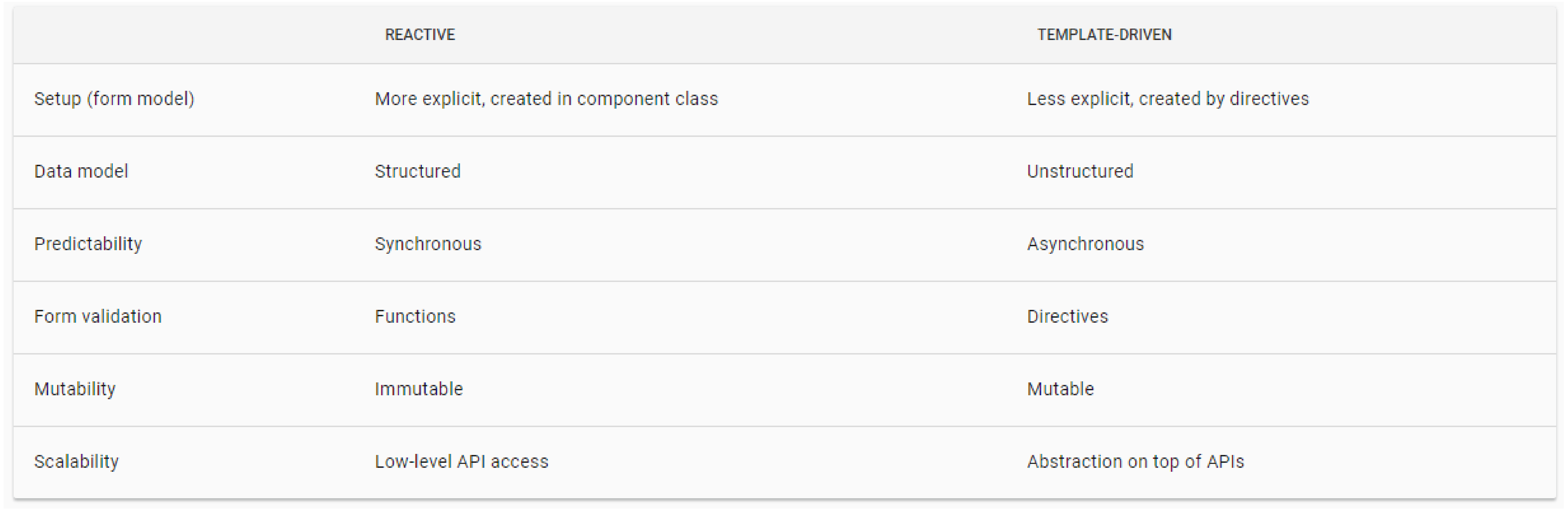


State	Class if true	Class if false
The control has been visited.	`ng-touched`	`ng-untouched`
The control's value has changed.	`ng-dirty`	`ng-pristine`
The control's value is valid.	`ng-valid`	`ng-invalid`

https://angular.io/guide/forms

plain concepts

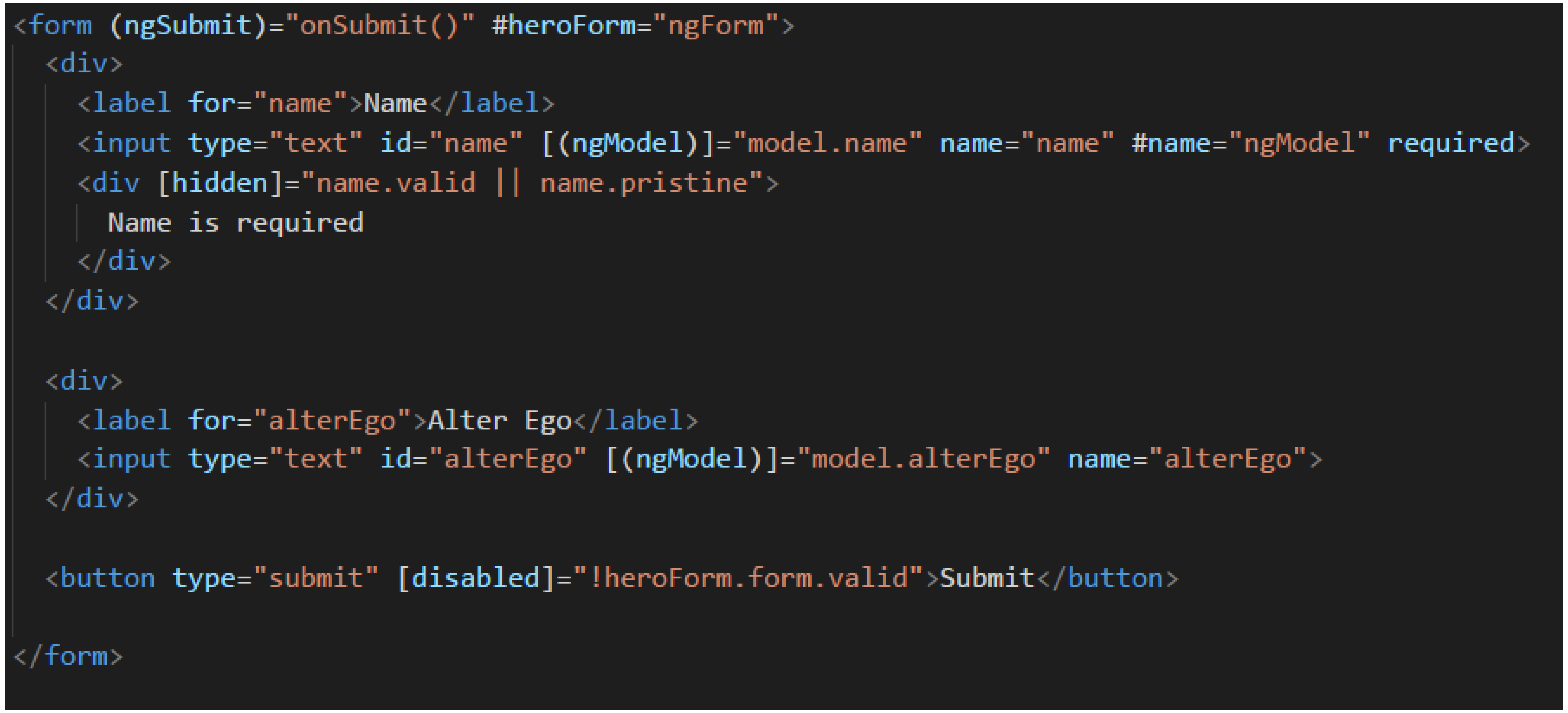
FORMULARIOS



	REACTIVE	TEMPLATE-DRIVEN
Setup (form model)	More explicit, created in component class	Less explicit, created by directives
Data model	Structured	Unstructured
Predictability	Synchronous	Asynchronous
Form validation	Functions	Directives
Mutability	Immutable	Mutable
Scalability	Low-level API access	Abstraction on top of APIs

https://angular.io/guide/forms-overview

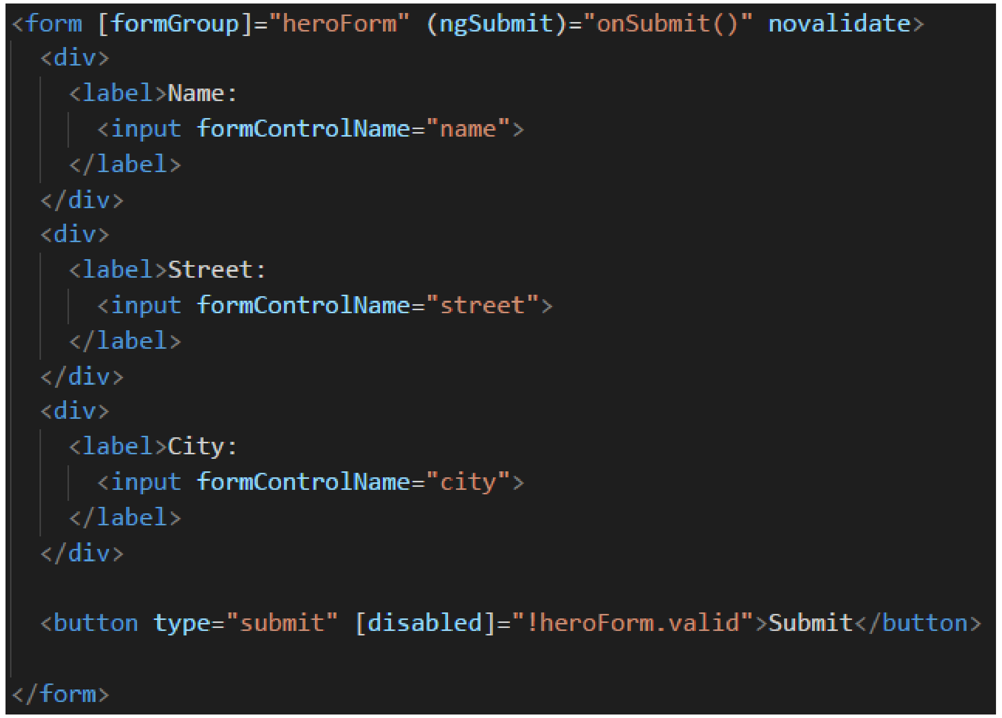
FORMS - TEMPLATE-DRIVEN FORMS



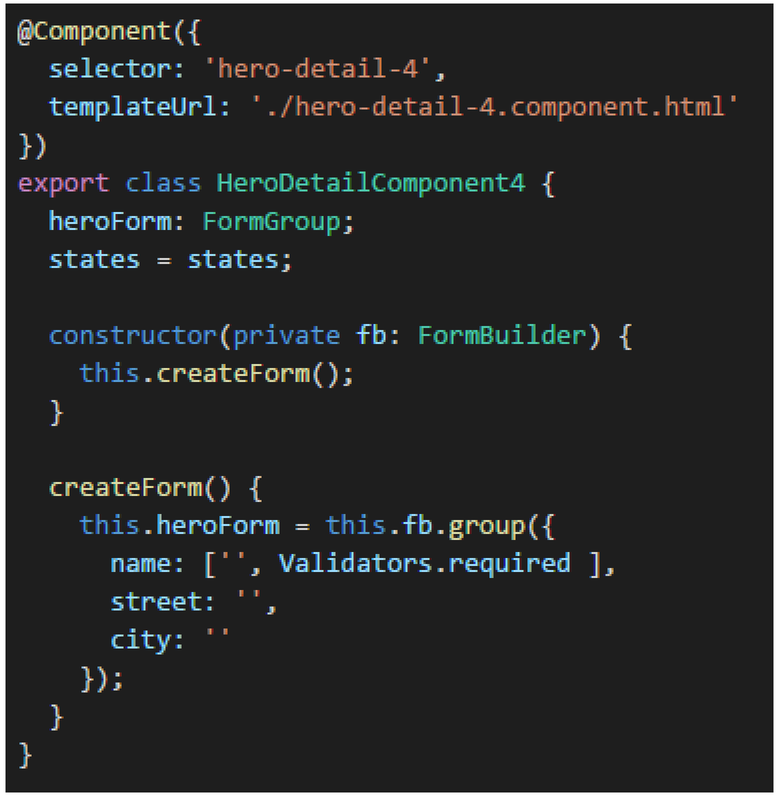
```
<form (ngSubmit)="onSubmit()" #heroForm="ngForm">
  <div>
    <label for="name">Name</label>
    <input type="text" id="name" [(ngModel)]="model.name" name="name" #name="ngModel" required>
    <div [hidden]="name.valid || name.pristine">
      Name is required
    </div>
  </div>

  <div>
    <label for="alterEgo">Alter Ego</label>
    <input type="text" id="alterEgo" [(ngModel)]="model.alterEgo" name="alterEgo">
  </div>

  <button type="submit" [disabled]="!heroForm.form.valid">Submit</button>

</form>
```

plain concepts

FORMS - REACTIVE FORMS



```html
<form [formGroup]="heroForm" (ngSubmit)="onSubmit()" novalidate>
  <div>
    <label>Name:
      <input formControlName="name">
    </label>
  </div>
  <div>
    <label>Street:
      <input formControlName="street">
    </label>
  </div>
  <div>
    <label>City:
      <input formControlName="city">
    </label>
  </div>

  <button type="submit" [disabled]="!heroForm.valid">Submit</button>

</form>
```

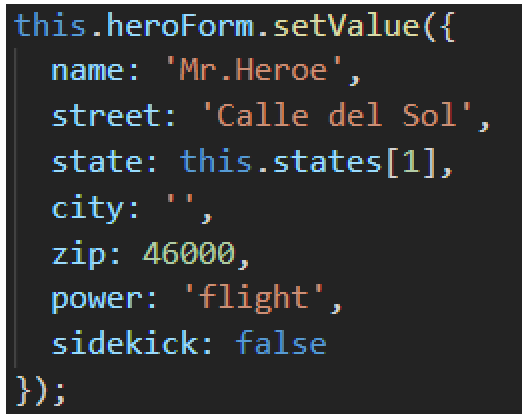


```typescript
@Component({
  selector: 'hero-detail-4',
  templateUrl: './hero-detail-4.component.html'
})
export class HeroDetailComponent4 {
  heroForm: FormGroup;
  states = states;

  constructor(private fb: FormBuilder) {
    this.createForm();
  }

  createForm() {
    this.heroForm = this.fb.group({
      name: ['', Validators.required ],
      street: '',
      city: ''
    });
  }
}
```

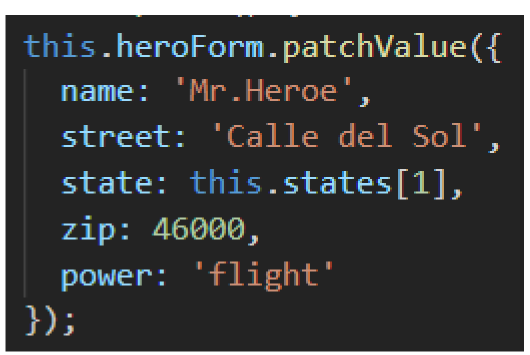
plain concepts

FORMS - REACTIVE FORMS - UPDATE

- Dos maneras para actualizar el modelo
  - setValue: Hay que definir todos los campos, si no, lanza error.
  - patchValue: Actualiza lo que se le define.



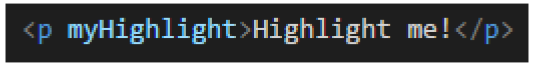
```
this.heroForm.setValue({
  name: 'Mr.Heroe',
  street: 'Calle del Sol',
  state: this.states[1],
  city: '',
  zip: 46000,
  power: 'flight',
  sidekick: false
});
```



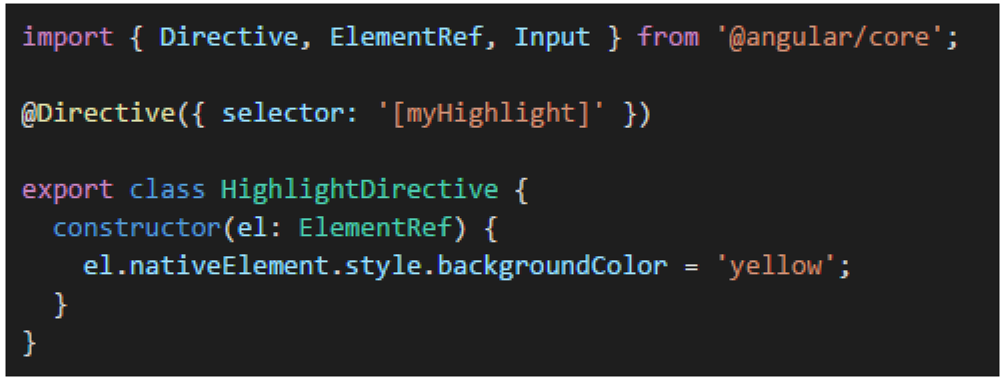
```
this.heroForm.patchValue({
  name: 'Mr.Heroe',
  street: 'Calle del Sol',
  state: this.states[1],
  zip: 46000,
  power: 'flight'
});
```

DIRECTIVES

- Components: Directivas con template.
- Structural directives: Cambian layout del DOM añadiendo o eliminando elementos de éste.
  - ngFor, ngIf, …
- Attribute directives: Cambian apariencia o comportamiento de un elemento.
  - ngStyle, …



```html
<p myHighlight>Highlight me!</p>
```



```typescript
import { Directive, ElementRef, Input } from '@angular/core';

@Directive({ selector: '[myHighlight]' })

export class HighlightDirective {
  constructor(el: ElementRef) {
    el.nativeElement.style.backgroundColor = 'yellow';
  }
}
```

PIPES

- Transforman valores en un template. Hay que importarlos en el modulo!
- Built-in:
  - DatePipe, UpperCasePipe, LowerCasePipe, CurrencyPipe, NumberPipe, PercentPipe, JsonPipe.

```
<p>The hero's birthday is {{ birthday | date }}</p>
```

```
<p>The chained hero's birthday is {{  birthday | date:'fullDate' | uppercase}}</p>
```

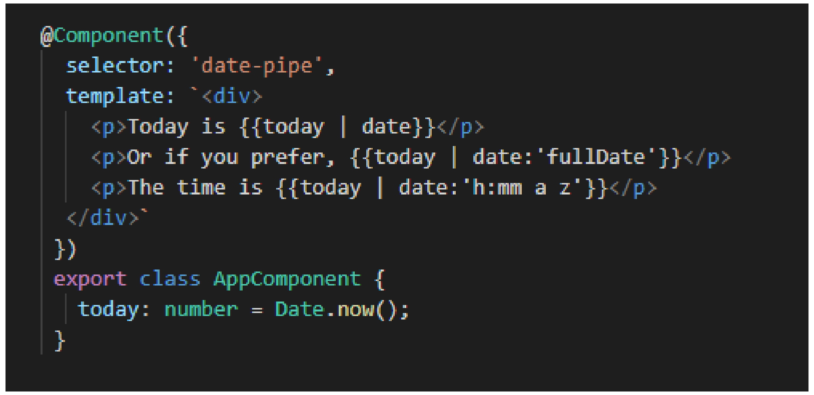
- Custom:

```
<p>Super power boost: {{2 | exponentialStrength: 10}}</p>
```

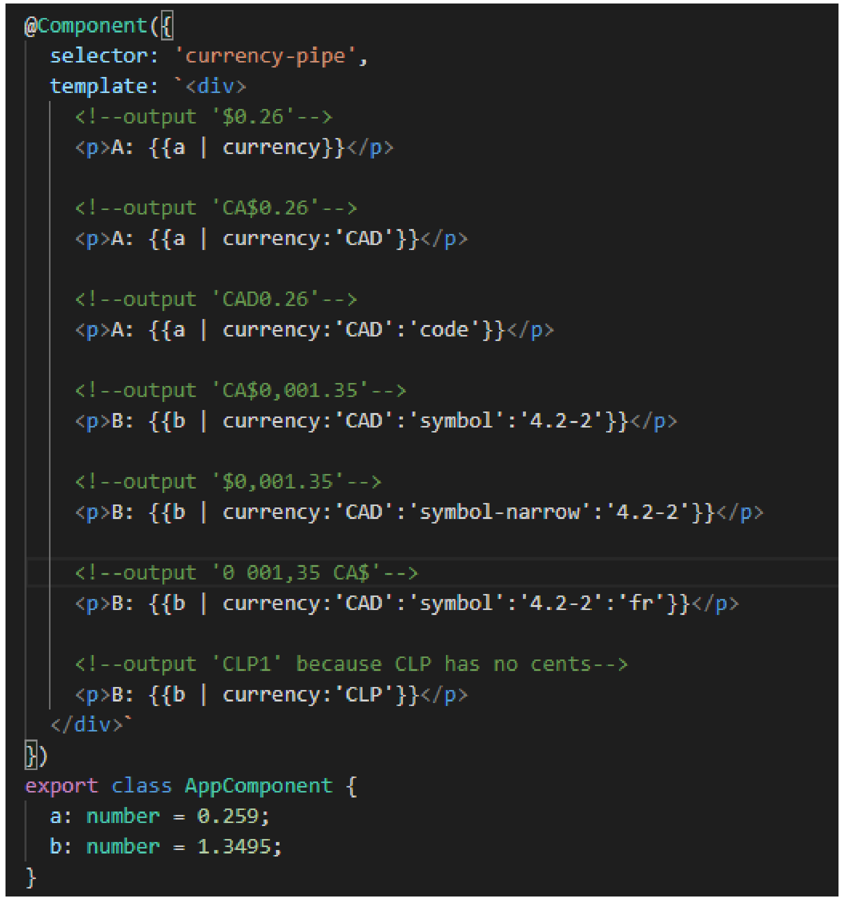
```
@Pipe({ name: 'exponentialStrength' })
export class ExponentialStrengthPipe implements PipeTransform {
  transform(value: number, exponent: string): number {
    let exp = parseFloat(exponent);
    return Math.pow(value, isNaN(exp) ? 1 : exp);
  }
}
```

plain concepts

PIPES



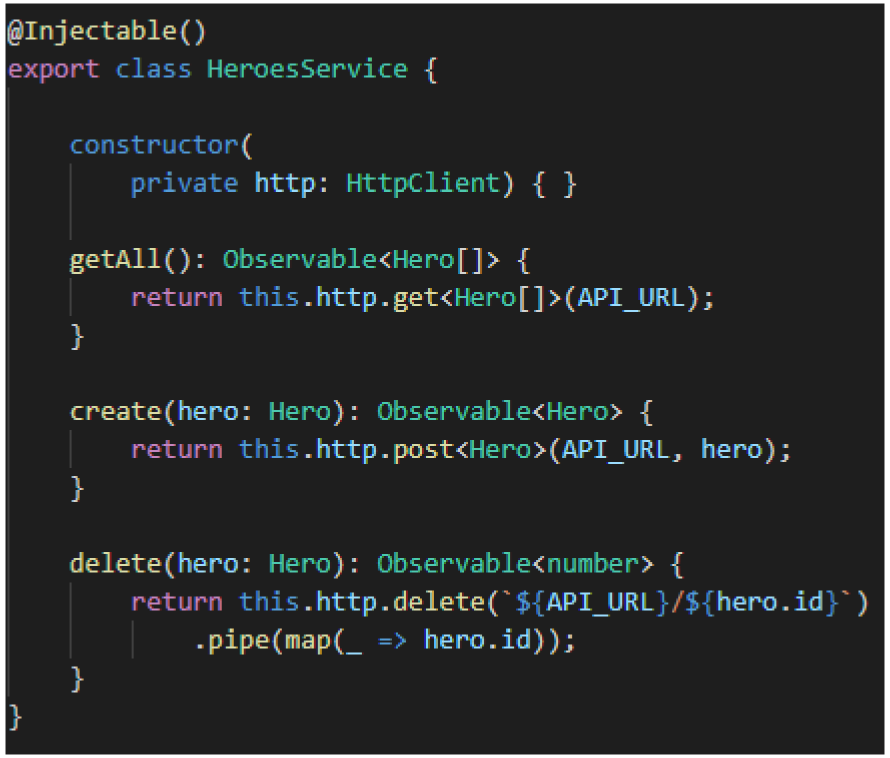
```
@Component({
  selector: 'date-pipe',
  template: `<div>
    <p>Today is {{today | date}}</p>
    <p>Or if you prefer, {{today | date:'fullDate'}}</p>
    <p>The time is {{today | date:'h:mm a z'}}</p>
  </div>`
})
export class AppComponent {
  today: number = Date.now();
}
```



```
@Component({
  selector: 'currency-pipe',
  template: `<div>
    <!--output '$0.26'-->
    <p>A: {{a | currency}}</p>

    <!--output 'CA$0.26'-->
    <p>A: {{a | currency:'CAD'}}</p>

    <!--output 'CAD0.26'-->
    <p>A: {{a | currency:'CAD':'code'}}</p>

    <!--output 'CA$0,001.35'-->
    <p>B: {{b | currency:'CAD':'symbol':'4.2-2'}}</p>

    <!--output '$0,001.35'-->
    <p>B: {{b | currency:'CAD':'symbol-narrow':'4.2-2'}}</p>

    <!--output '0 001,35 CA$'-->
    <p>B: {{b | currency:'CAD':'symbol':'4.2-2':'fr'}}</p>

    <!--output 'CLP1' because CLP has no cents-->
    <p>B: {{b | currency:'CLP'}}</p>
  </div>`
})
export class AppComponent {
  a: number = 0.259;
  b: number = 1.3495;
}
```

plain concepts

HTTP

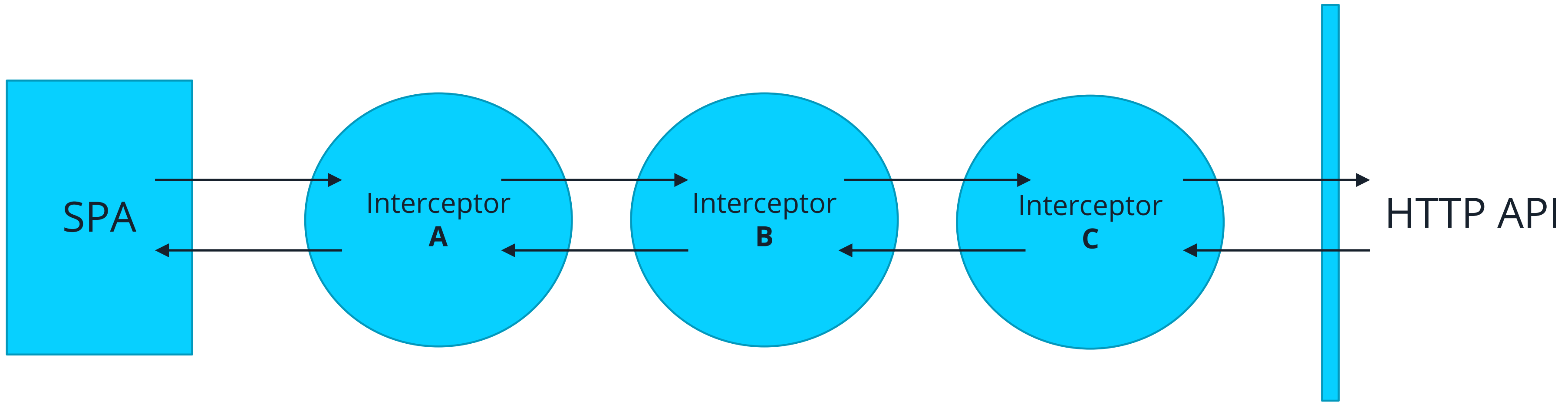


```
@Injectable()
export class HeroesService {

    constructor(
        private http: HttpClient) { }

    getAll(): Observable<Hero[]> {
        return this.http.get<Hero[]>(API_URL);
    }

    create(hero: Hero): Observable<Hero> {
        return this.http.post<Hero>(API_URL, hero);
    }

    delete(hero: Hero): Observable<number> {
        return this.http.delete(`${API_URL}/${hero.id}`)
            .pipe(map(_ => hero.id));
    }
}
```

plain concepts

HTTP INTERCEPTORS

- Permiten interceptar peticiones y respuestas HTTP para hacer algo con ellas o modificarlas.
- Las instancias de HttpRequest y HttpResponse son de solo lectura. Tener en cuenta si se quieren modificar.
- Se puede aplicar varios interceptores, que se procesaran en orden:

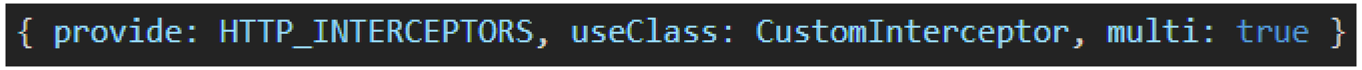


plain concepts

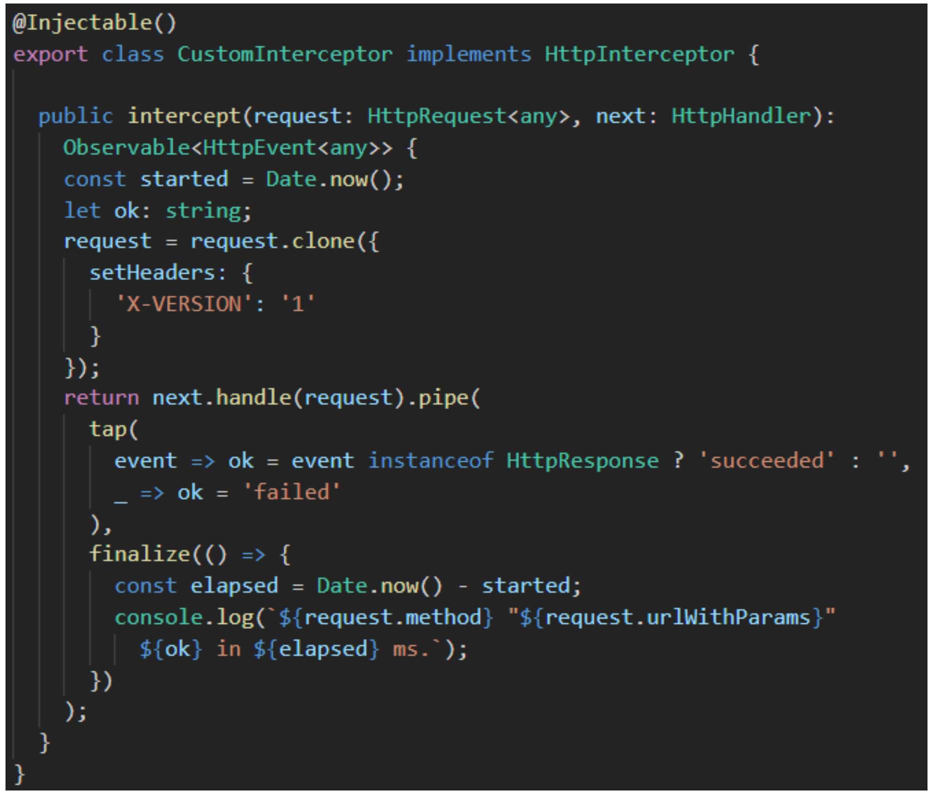
HTTP INTERCEPTORS - USOS

- Cambiar la url de la petición (ej: http=>https)
- Mostrar feedback mientras se resuelve la petición (ej: spinner)
- Añadir cabeceras
- Mostrar feedback del resultado de las peticiones (ej: toaster de exito o error)
- Reintentar peticiones si fallan (ej: conexión movil mala)
- Profiling
- Fake backend (ej: tests, demos)
- Autenticación

HTTP INTERCEPTORS



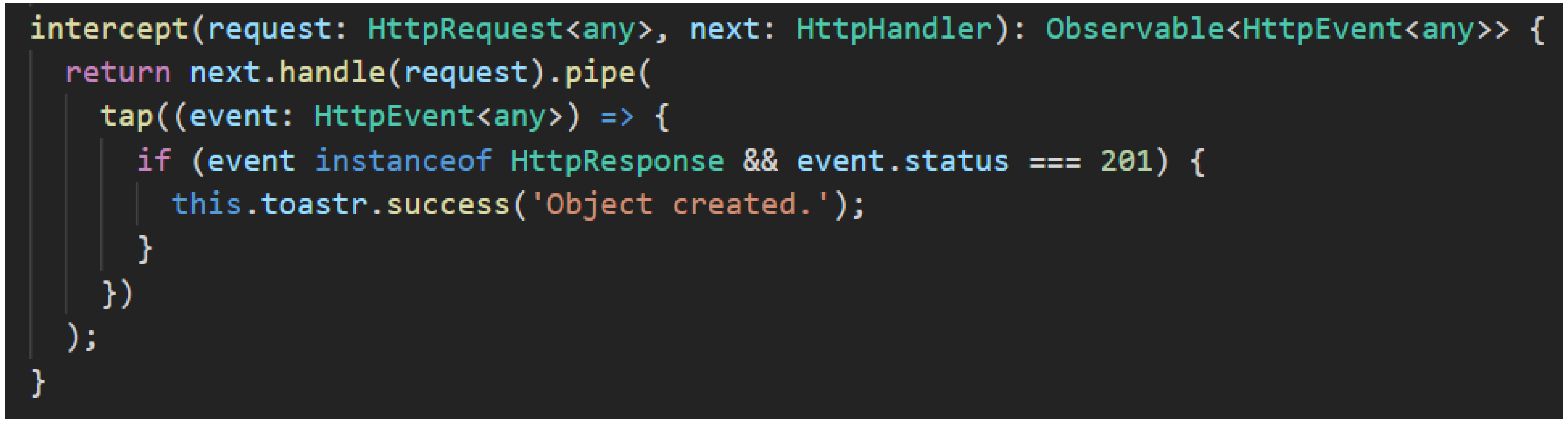
```
{ provide: HTTP_INTERCEPTORS, useClass: CustomInterceptor, multi: true }
```



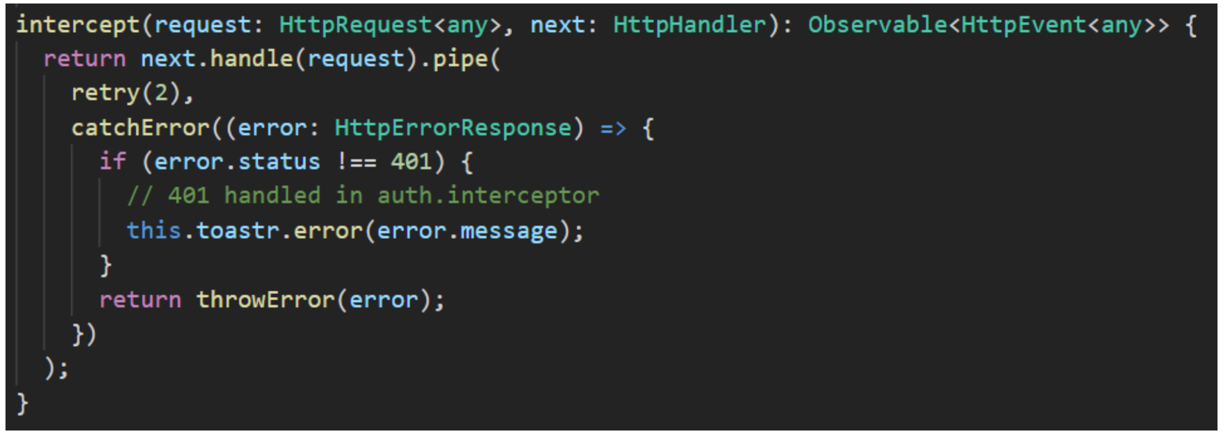
```
@Injectable()
export class CustomInterceptor implements HttpInterceptor {

  public intercept(request: HttpRequest<any>, next: HttpHandler):
    Observable<HttpEvent<any>> {
    const started = Date.now();
    let ok: string;
    request = request.clone({
      setHeaders: {
        'X-VERSION': '1'
      }
    });
    return next.handle(request).pipe(
      tap(
        event => ok = event instanceof HttpResponse ? 'succeeded' : '',
        _ => ok = 'failed'
      ),
      finalize(() => {
        const elapsed = Date.now() - started;
        console.log(`${request.method} "${request.urlWithParams}"
          ${ok} in ${elapsed} ms.`);
      })
    );
  }
}
```

plain concepts

HTTP INTERCEPTORS



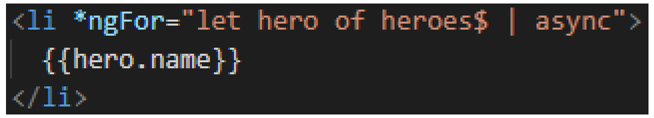
```
intercept(request: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
  return next.handle(request).pipe(
    tap((event: HttpEvent<any>) => {
      if (event instanceof HttpResponse && event.status === 201) {
        this.toastr.success('Object created.');
      }
    })
  );
}
```



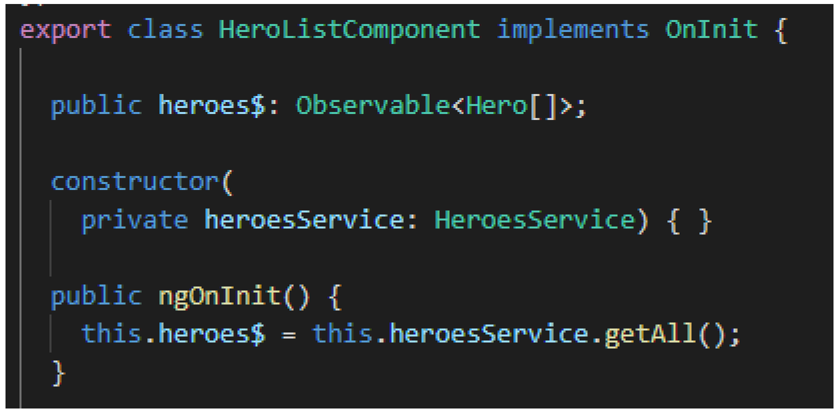
```
intercept(request: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
  return next.handle(request).pipe(
    retry(2),
    catchError((error: HttpErrorResponse) => {
      if (error.status !== 401) {
        // 401 handled in auth.interceptor
        this.toastr.error(error.message);
      }
      return throwError(error);
    })
  );
}
```

plain concepts

ASYNCPIPE



```
<li *ngFor="let hero of heroes$ | async">
  {{hero.name}}
</li>
```



```
export class HeroListComponent implements OnInit {

  public heroes$: Observable<Hero[]>;

  constructor(
    private heroesService: HeroesService) { }

  public ngOnInit() {
    this.heroes$ = this.heroesService.getAll();
  }
```

JIT VS AOT

- *Just-in-time* (JIT)
  - You can compile the app in the browser, at runtime, as the application loads, using the *just-in-time* (JIT) compiler
  - JIT compilation incurs a runtime performance penalty. Views take longer to render because of the in-browser compilation step. The application is bigger because it includes the Angular compiler and a lot of library code that the application won't actually need. Bigger apps take longer to transmit and are slower to load.
  - Compilation can uncover many component-template binding errors. JIT compilation discovers them at runtime, which is late in the process.

- *Ahead-of-time* (AOT)
  - The *ahead-of-time* (AOT) compiler can catch template errors early and improve performance by compiling at build time.

https://angular.io/guide/aot-compiler

AOT

- *Faster rendering*
  - With AOT, the browser downloads a pre-compiled version of the application. The browser loads executable code so it can render the application immediately, without waiting to compile the app first.
- *Fewer asynchronous requests*
  - The compiler *inlines* external HTML templates and CSS style sheets within the application JavaScript, eliminating separate ajax requests for those source files.
- *Smaller Angular framework download size*
  - There's no need to download the Angular compiler if the app is already compiled. The compiler is roughly half of Angular itself, so omitting it dramatically reduces the application payload.
- *Detect template errors earlier*
  - The AOT compiler detects and reports template binding errors during the build step before users can see them.
- *Better security*
  - AOT compiles HTML templates and components into JavaScript files long before they are served to the client. With no templates to read and no risky client-side HTML or JavaScript evaluation, there are fewer opportunities for injection attacks.

https://angular.io/guide/aot-compiler

INTERNACIONALIZACIÓN VS LOCALIZACIÓN

- *Internacionalizacion (i18n)*
  - *Proceso de preparación y diseño de una aplicación para permitir su uso en diferentes idiomas*
  - *Separamos el contenido que va a ser traducido*
  - *Permitir texto bidireccional (izq a derecha y derecha a izquierda)*

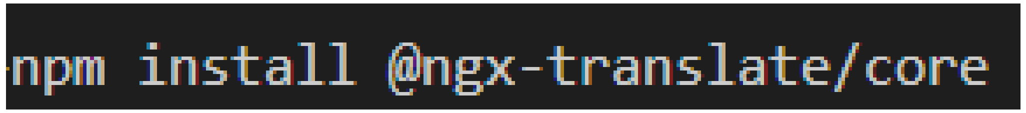
- *Localización*
  - *Crear versions de un proyecto para diferentes idiomas o variantes de un idioma*
    - *Formateo de fechas*
    - *Formateo de moneda*
    - *Traducción de nombres como países*
  - *Formato: **{ID_idioma}_{extension_local} : en-US***

INTERNACIONALIZACIÓN: NGX-TRANSLATE VS I18N

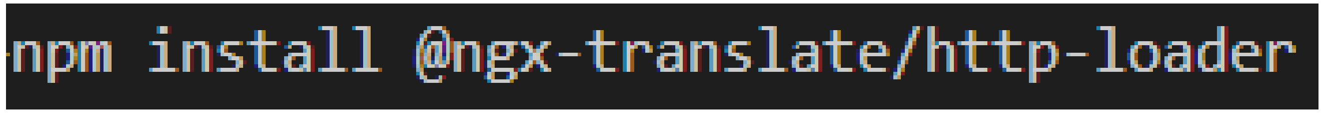
- *Angular i18n*
  - *Oficial de Angular*
  - *Herramientas para extraer strings en ficheros de traducción*
  - *SEO*
- *Ngx-translate*
  - *Muy sencillo*
  - *Podemos cambiar el lenguaje de la aplicación en tiempo real sin recargarla*
  - *Usa archivos JSON*
  - *Muy extendido*
  - *Similar a AngularJS*
  - *Interpolación*

NGX-TRANSLATE

- Instalar paquetes



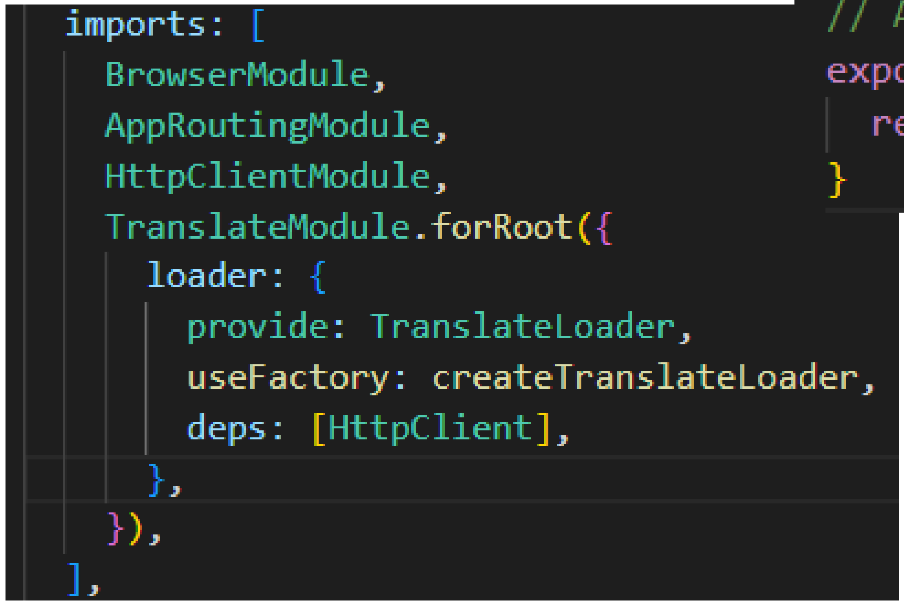
```
npm install @ngx-translate/core
```



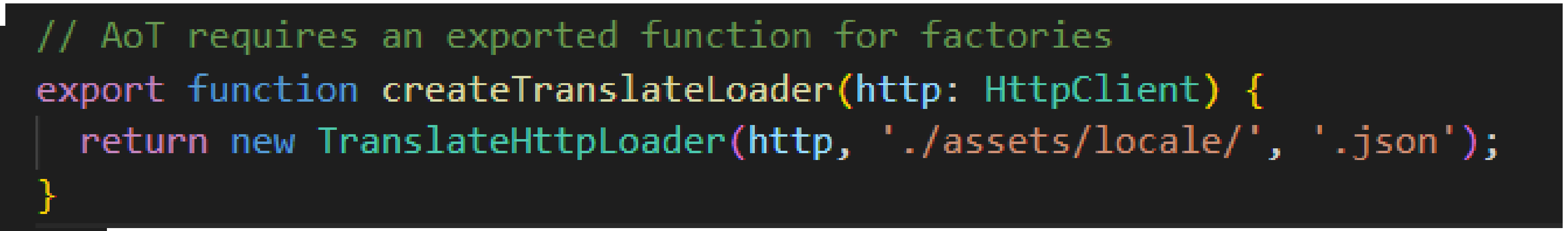
```
npm install @ngx-translate/http-loader
```

NGX-TRANSLATE

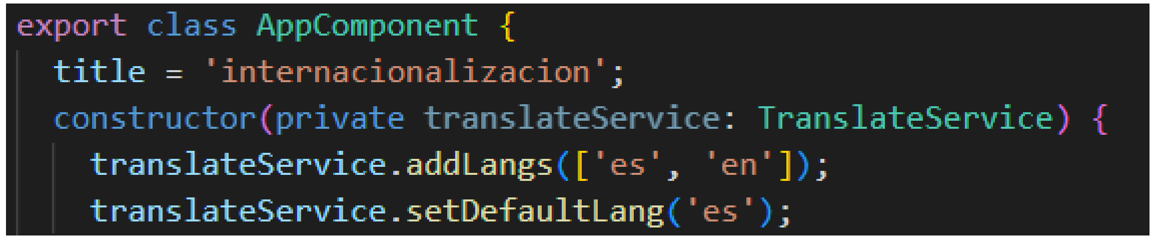
- Configuración



```
imports: [
  BrowserModule,
  AppRoutingModule,
  HttpClientModule,
  TranslateModule.forRoot({
    loader: {
      provide: TranslateLoader,
      useFactory: createTranslateLoader,
      deps: [HttpClient],
    },
  }),
],
```



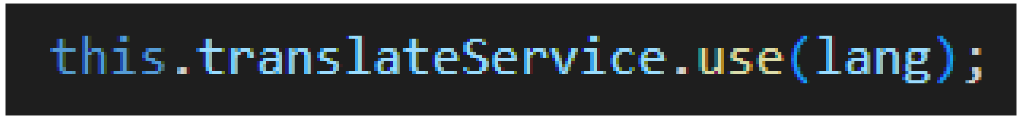
```
// AoT requires an exported function for factories
export function createTranslateLoader(http: HttpClient) {
  return new TranslateHttpLoader(http, './assets/locale/', '.json');
}
```



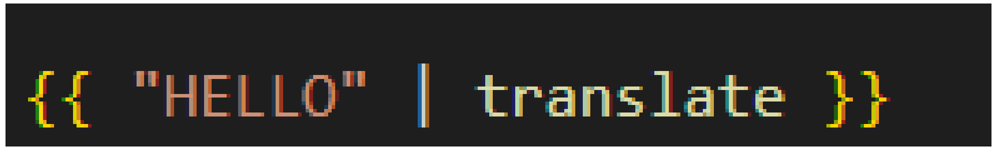
```
export class AppComponent {
  title = 'internacionalizacion';
  constructor(private translateService: TranslateService) {
    translateService.addLangs(['es', 'en']);
    translateService.setDefaultLang('es');
```

NGX-TRANSLATE

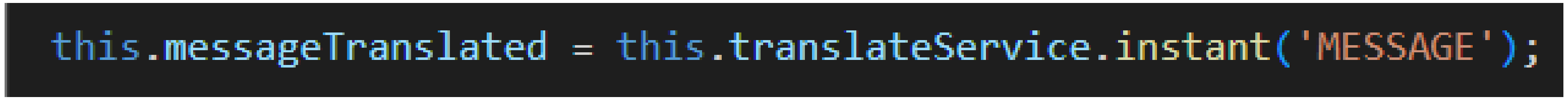
- USO



```
this.translateService.use(lang);
```



```
{{ "HELLO" | translate }}
```

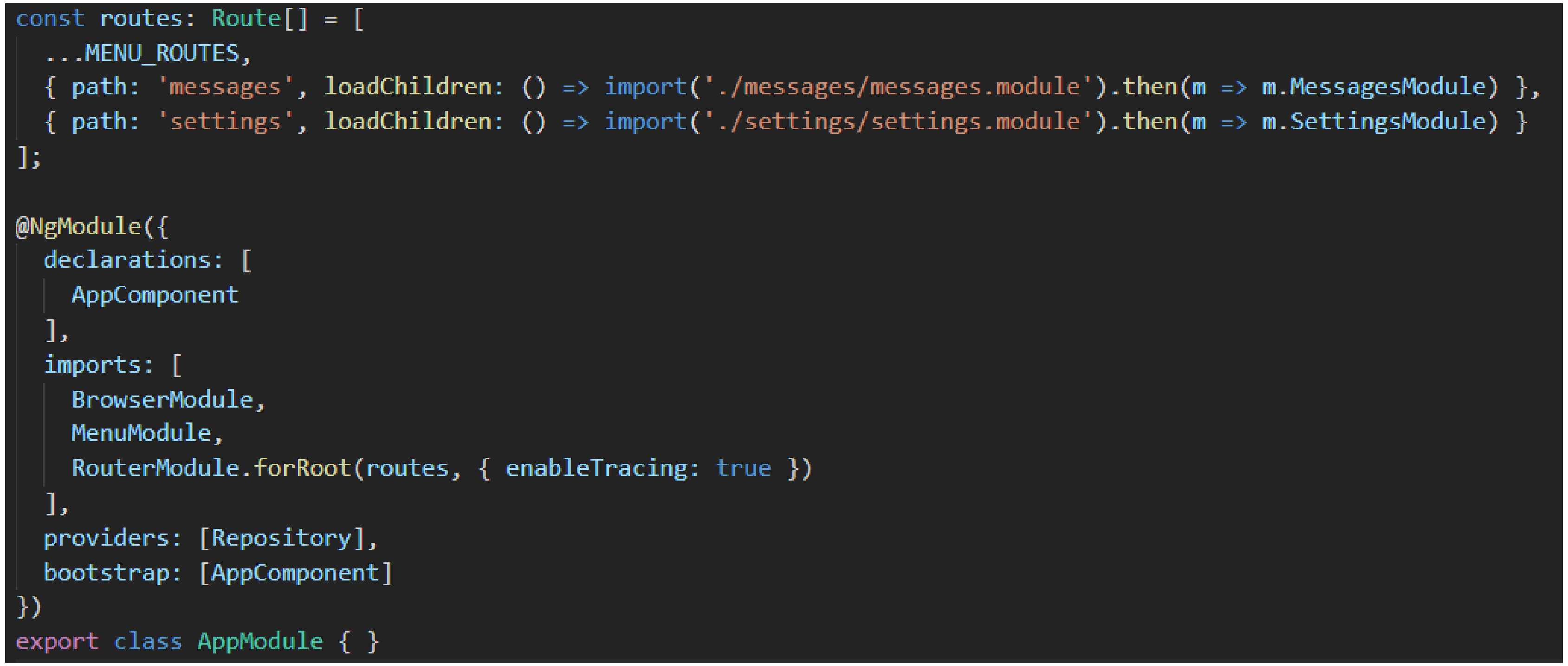


```
this.messageTranslated = this.translateService.instant('MESSAGE');
```

plain concepts

LAZY LOADING DE MÓDULOS

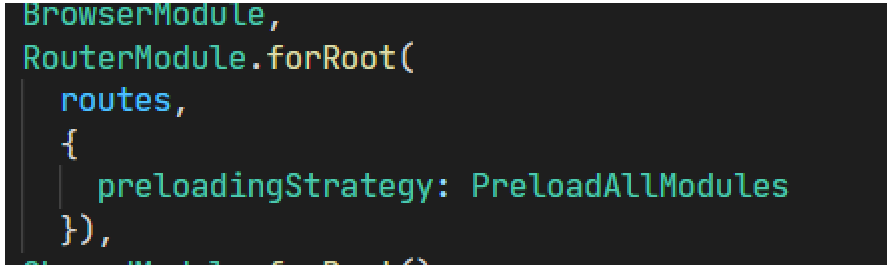
- Permite cargar dinámicamente módulos bajo demanda.



```
const routes: Route[] = [
  ...MENU_ROUTES,
  { path: 'messages', loadChildren: () => import('./messages/messages.module').then(m => m.MessagesModule) },
  { path: 'settings', loadChildren: () => import('./settings/settings.module').then(m => m.SettingsModule) }
];

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    MenuModule,
    RouterModule.forRoot(routes, { enableTracing: true })
  ],
  providers: [Repository],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

plain concepts

PRELOAD MODULES

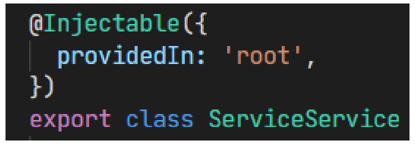


```
BrowserModule,
RouterModule.forRoot(
  routes,
  {
    preloadingStrategy: PreloadAllModules
  }),
```

plain concepts

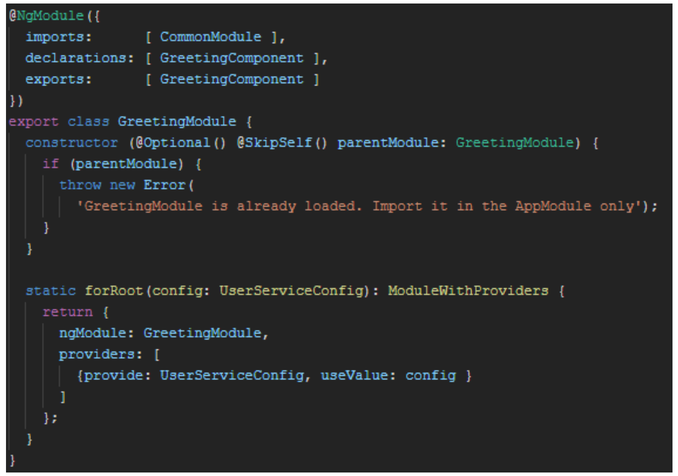
FOR ROOT

- Si un módulo *Shared* define servicios, la carga de este módulo en otros módulos, duplicaría las instancias de estos servicios y estos no se comportarían como singleton cuando el módulo se carga asíncronamente
- Para evitarlo:
  - Usar

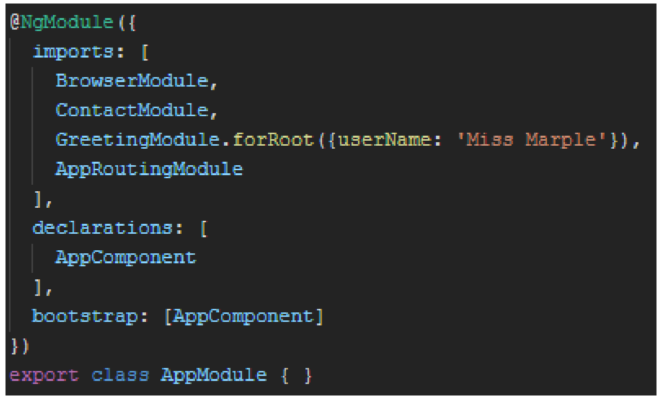


```
@Injectable({
  providedIn: 'root',
})
export class ServiceService
```

  - Definir método forRoot() en el módulo para garantizar que se cargue desde la raíz.



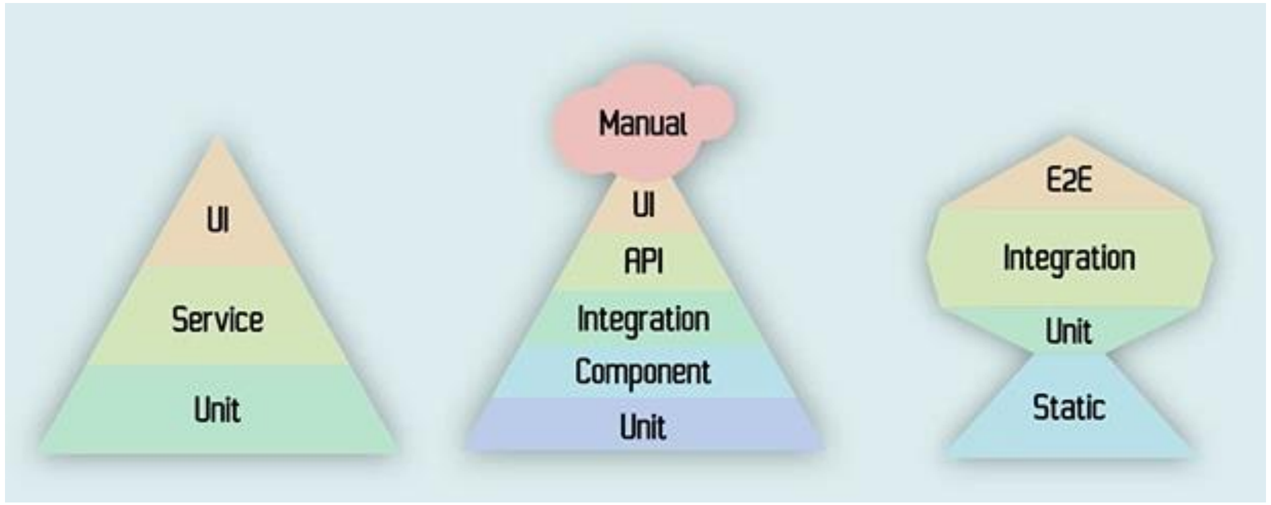
```
@NgModule({
  imports:      [ CommonModule ],
  declarations: [ GreetingComponent ],
  exports:      [ GreetingComponent ]
})
export class GreetingModule {
  constructor (@Optional() @SkipSelf() parentModule: GreetingModule) {
    if (parentModule) {
      throw new Error(
        'GreetingModule is already loaded. Import it in the AppModule only');
    }
  }

  static forRoot(config: UserServiceConfig): ModuleWithProviders {
    return {
      ngModule: GreetingModule,
      providers: [
        {provide: UserServiceConfig, useValue: config }
      ]
    };
  }
}
```



```
@NgModule({
  imports: [
    BrowserModule,
    ContactModule,
    GreetingModule.forRoot({userName: 'Miss Marple'}),
    AppRoutingModule
  ],
  declarations: [
    AppComponent
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

TESTING

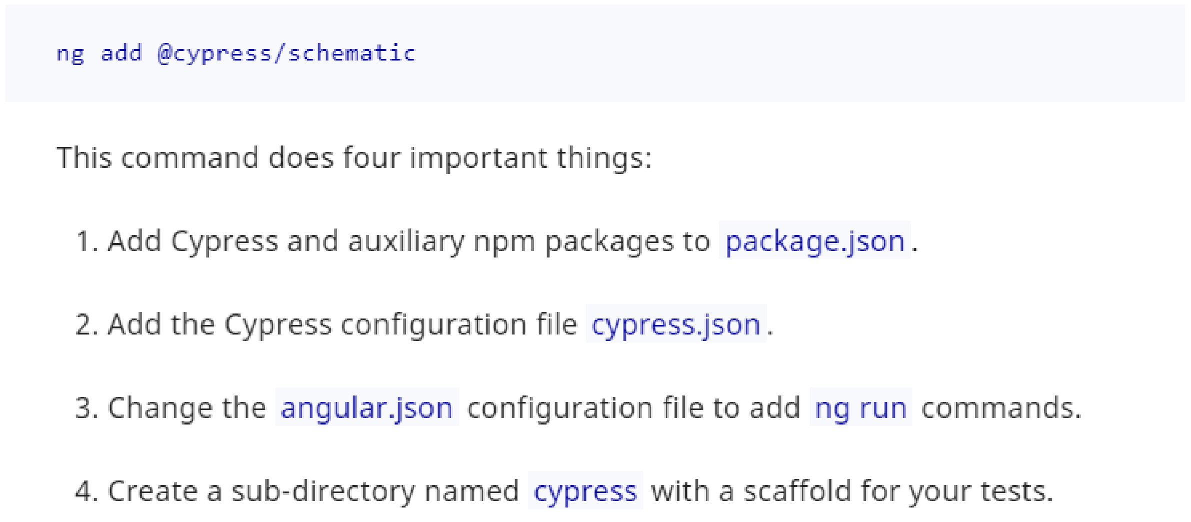
- Tests Unitarios (Jasmine & Karma)
- Tests Integración (testing-library)
- Tests e2e (cypress)
- Tests performance
- Tests Accesibilidad



E2E : CYPRESS VS PROTRACTOR

cypress.io

INSTALAR CYPRESS

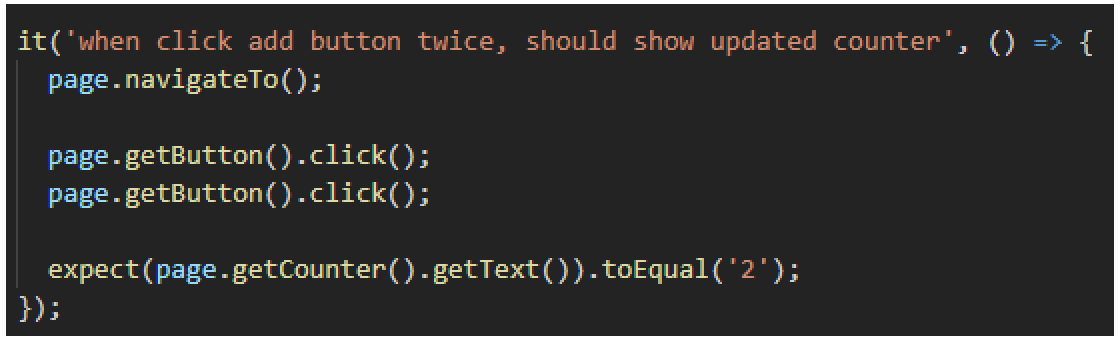


```
ng add @cypress/schematic
```

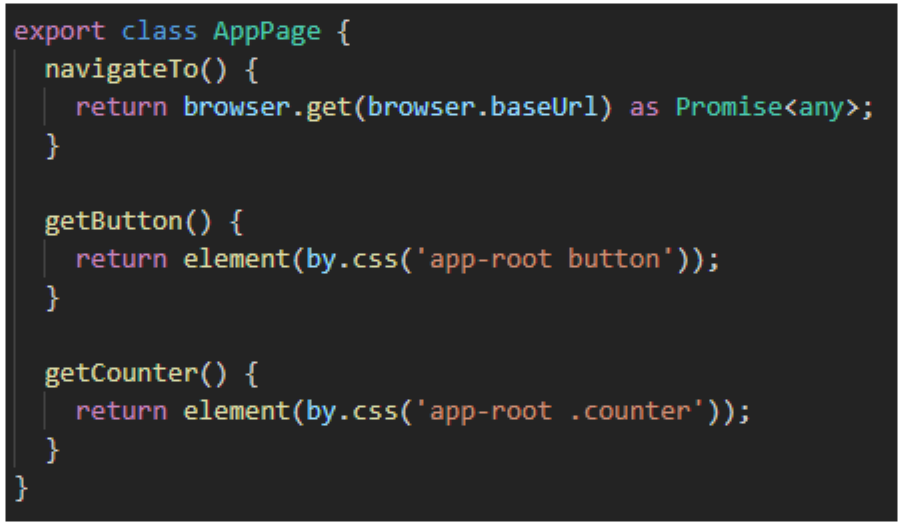
This command does four important things:

1. Add Cypress and auxiliary npm packages to `package.json`.
2. Add the Cypress configuration file `cypress.json`.
3. Change the `angular.json` configuration file to add `ng run` commands.
4. Create a sub-directory named `cypress` with a scaffold for your tests.

E2E TESTING



```
it('when click add button twice, should show updated counter', () => {
  page.navigateTo();

  page.getButton().click();
  page.getButton().click();

  expect(page.getCounter().getText()).toEqual('2');
});
```



```
export class AppPage {
  navigateTo() {
    return browser.get(browser.baseUrl) as Promise<any>;
  }

  getButton() {
    return element(by.css('app-root button'));
  }

  getCounter() {
    return element(by.css('app-root .counter'));
  }
}
```

plain concepts

¡GRACIAS!

plain concepts

Barcelona

Bilbao

Madrid

Sevilla

Dubai

London

Seattle

plain concepts Bilbao

afole@plainconcepts.com

Ledesma 10 bis, 2ª planta

48001 Bilbao, España

+34 94 6008 168