Para empezar con el curso debemos tener instalado Angular CLI.

Instalación en línea de comandos

npm install -g @angular/cli

Para crear una aplicación de ejemplo con Angular CLI y no generar los archivos

ng new myapp --dry-run

ng new myapp --prefix test --dry-run

ng new myapp --skip-test --prefix test --dry-run

ng new myapp --skip-test --prefix test --style scss

ng new myapp --skip-test --prefix test --style scss –routing

**Tslint** es una extensión estática de análisis de Typescript la cual checkea la mantenibilidad y legibilidad de nuestro código.

ng lint

ng lint --format stylish

ng lint –fix

Ejemplo inicial en la carpeta aweather-refactor-inicial [**Github**](https://github.com/OpenWebinarsNet/Angular-buenas-practicas)

Componentes:

La descripción siempre separada por “-“

“descripción”.component.ts  
“descripcion”.component.html  
“descripcion”.component.css

Módulos:

La descripción siempre separada por “-“

“descripción”.module.ts

Directivas:

La descripción siempre separada por “-“

“descripción”.directive.ts

Pipes:

La descripción siempre separada por “-“

“descripción”.pipe.ts

Routing

La descripción siempre separada por “-“

“descripción”-routing.module.ts

Modelos:

La descripción siempre separada por “-“

“descripción”.model.ts

El Principio de responsabilidad única es el primero de los cinco que componen SOLID.

El principio de Responsabilidad Única nos viene a decir que un objeto debe realizar una única cosa. Es muy habitual, si no prestamos atención a esto, que acabemos teniendo clases que tienen varias responsabilidades lógicas a la vez.

Refactorización a funciones pequeñas aislando funcionalidad y pensando en el principio de responsabilidad única.

Orden de los ficheros:

---- imports

...................

...................

---- decorador

...................

...................

---- clase

\*\*\*\* declaración de variables

\*\*\*\* contructor

\*\*\*\* implementación de interfaces (ngOnInit, ngOnChanges,...)

\*\*\*\* métodos publicos

\*\*\*\* métodos privados

Cuatro carpetas principales para estructurar nuestro código:

....components

....login

....components

....layout

....weather

...components

....layout

....core

....authorization

....cache

....functions

....localization

....services

....weather-service

....ui-controls

....dialogs

....directives

....form-controls

....material

....pipes

Lazy loading es la carga de módulos de forma asíncrona según el usuario lo vaya necesitando.

Optimizado para dispositivos móviles ya que no se tiene que bajar toda la aplicación si sólo van a ver dos páginas, el usuario de nuestra web sólo se descargará lo necesario.

Cuidado con la ruta “\*\*”, como Angular realmente no sabe las rutas que tenemos dentro de cada módulo. Angular se cree que esa ruta no existe.

<https://github.com/OpenWebinarsNet/Angular-buenas-practicas>

Para la optimización utilizaremos Source map explorer

npm install -g source-map-explorer

Configuramos el package.json para que nos funcione “npm run sme”

"sme": "./node\_modules/.bin/source-map.explorer",

ng build

Coger un fichero de la ruta de salida, nuestro “outputPath” del fichero angular.json.

npm run sme "ruta fichero"