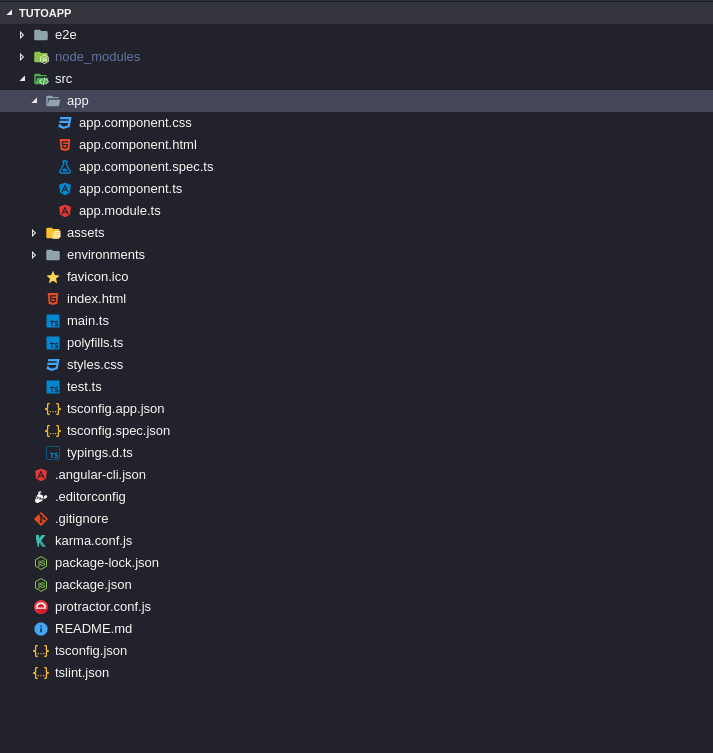
ANGULAR

**Estructura** de una app creada con Angular



📂 e2e: Esta carpeta por el momento no nos es útil, aquí se encuentra el código para escribir tests end to end que prueben la aplicación

📂 node\_modules: En esta carpeta se encuentran las librerías de angular y sus dependencias, cuando instalemos librerías se añadirán aquí. Generalmente no hay que tocar nada de esta carpeta.

📂 src: Aquí se encuentran los archivos que componen nuestra aplicación

* + 📂 app: Aquí se donde se van a encontrar los componentes, vistas, y servicios de la app. Por el momento hay un componente llamado app con sus respectivos archivos (css, html controlador, tests, etc)
    - app.module.ts: En este archivo se especifica los componentes que vamos a usar en la app web. Cuando creemos un componente tenemos que importarlo en este achivo.
  + favicon: El favicon de la web
  + index.html: Punto de entrada a nuestra web, este archivo se carga en todas las webs, por lo que puedes poner código para que se incluya en todas las vistas.
  + main.ts: Algunas configuraciones de Angular, de momento no nos hace falta tocarlo.
  + polyfills.ts: Configuraciones y código que se ejecutará antes de que se inicie la app. De momento tampoco nos hace falta tocarlo.
  + styles.css: Estilos css globales que se aplicarán en toda las vistas de la página.
  + test.ts: Configuración para los tests. No es útil de momento
  + tsconfig.app.json, tsconfig.spec.json y typings.d.ts: Lo mismo que el anterior.

{} .angular-cli.json: Archivo de configuración de la app.

.editorconfig: Configuraciones a la hora de desarrollar, por ejemplo, como van a ser las identaciones.

.gitignore: Archivo para que git ignore ciertas carpetas que no hace falta subir, como node\_modules (cuando te bajas el proyecto ejecutas npm install para que descargue las dependencias en node\_modules).

{} karma.conf.js: Más configuraciones para los tests, esta vez los de Karma.

{} package-lock.json: Árbol de dependencias que se crea automáticamente

{} package.json: Archivo con las dependencias instaladas y los comandos que se pueden ejecutar con npm

protractor.conf.js: Configuración para protractor, una herramienta para realizar tests en el navegador.

README.md Archivo readme con información de la aplicación.

tsconfig.json: Configuración para Typescript, el lenguaje de Angular.

tslint.json: Configración del linter de TypeScript (un linter sirve para hacer comprobaciones del estilo del código que escribimos).

## 📄 package.json:

* metadatos{nombre,vesión,como se ha hecho, descripción y datos}
* scripts
  + - star preconfigurada ng serve compila antes ejec y abre navegador.
    - build compila y prepara para distribuir
    - test para ejecutar test
    - lint comprobación estática de la calidad del código
    - e2e test de integración
* dependencies: dependencias de ejecución {las necesitará el navegador.
  + - Core es el núcleo principal.
    - common enriquece html.
    - Forms para formularios.
    - platform-browser para preparar que se pueda ejecutar en el navegador (también es posible ejecutar en servidor).
    - Router permite tener SPA gestionando rutas del lado del cliente sin necesidad de recargar la página.
    - http comunicación con los servidores y APIS
    - core gestiona zonas de la pantalla
    - rxjs librería enorme reactive extensions, programación reactiva mediante emision de estados…
    - zone gestiona zonas de la pantalla y gestiona canvios
* devdependencies: dependencias del desarrollador {todas las herramientas que necesitaremos nosotros como desarrolladores pero que no necesitará el usuario: el propio cli en forma local, compilador y servicio para el lenguaje (html y typesrcipt)

{} .angular-cli.json: Archivo de configuración de la app. Configura al propio CLI. Guía para conocer qué se ha creado.

* src (se puede cambiar) continen el código fuente.
* Dist: directorio de distribución . Si hacemos npm run build ejecutará a ng build el cual va a compilar el código y generarlo para su distribucióne en un servidor. Generará un html i múltiples JS, dependencias, todo... Crea una carpeta DIST que es la carpeta que hay que poner en un servidor de ejecución. La configuración de la carpeta de origen y de destino configurará muchas otras cosas.
* Prefix: por defecto es app. Se puede cambiar por un nombre que nos recuerde el proyecto propio..

Resultado de imagen de icono typescript typings.d.ts para incorporar librerías Js a TS. Tiene Intelligence para gestionar tipos, clases,etc para poder utilizar las libresías JS como si hubieran sido programadas en TS. Se va incrementando con tipos de las librerías.

* polyfills.ts tratar que todos los navegadores entiendan una base común de JS y, en caso de no poder, hacer algo al respecto:Útil cuando se programa para navegadores antiguos o problemáticos.
* index.html es el fichero índice de la aplicación que generes (aparece en el archivo .angular.cli.json seción de index ,desde donde es posible utilizar otro nombre si lo necesitáramos). index.html contiene etiqueta nueva <app-root>Loading...</app-root> que inyecta los scripts transpilados . Si abrimos el inspector del navegador veremos etiquetas script:
  + inline.bundle.js lleva código propio de WEBPACK. Es un cargador de módulos dinámico en el lado del navegador.
  + Polyfills: se genera aunque no esté descomentado el tesxto del navegador correspondiente:
  + Styles: compilación de todos los estilos CSS, SCSS, LESS o lo que fuera se acaban importando aquí como módulos de JS.
  + Vendor: todo el código de terceros (de pacage,json)
  + Main: todo el código que tu programes.

## 📄 tsconfig.app.json

especifica como hacer las traducciones de nuestro código fuente. No se suelen manipular.

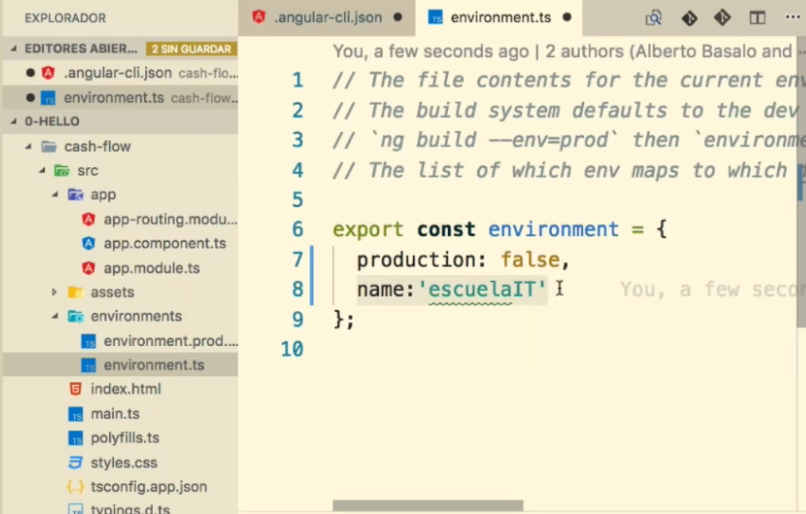
## 📄Main.ts

Es el fichero principal de la aplicación, en TS.

## 📂 Enviroments

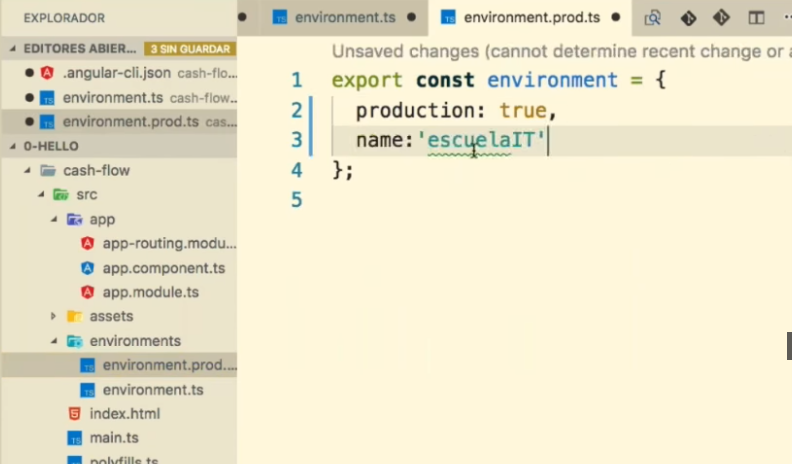
## 📄Enviroments.ts

Nos permite crear propiedades para configurar la aplicación.



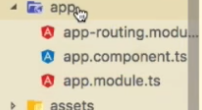
Cuando estemos en ejecución les podemos dadr un valos distinti desde

## 📄environment.pod.ts



El sistema será capaz de determinar cuando estamos en ejecución y cuando en producción al hacer ng build.

## 📂 app



Es la carpeta donde realmente vamos a programar

## 📄 app-module.ts

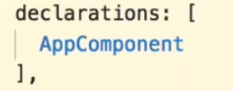
Todo lo realizaremos con clases. 

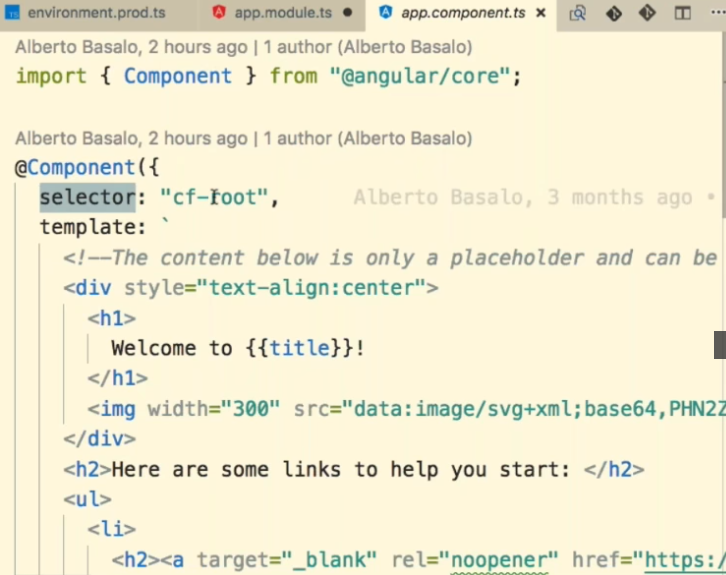
Posteriormente se decorará con una función

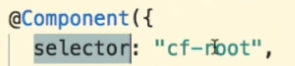


 invocamos a la función . La arroba aporta metadatos al lenguaje. Los decoradores aportan nuevas funcionalidades a algo que inicialmenta no lo tenían. Nos servirán para convertir cualquier clase en MÓDULOS , que son el armazón básico de las aplicaciones.

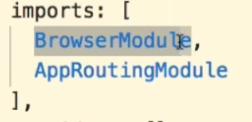
es una función que viene con el framework. Se importa de angular/core. Convierte cualquier función en un módulo. La configuración se hace a través de una serie de propiedades: Por defecto nos crea un patrón que deberemos ir modificando

nos dice de que está compuesto el módulo.AppComponent lo crea automáticamente el CLI es una clase con un decorador Component con su propio decorador.



el selector cf-root nos aparecerá en index.html .Crearemos un elemento html nuevo con ese selector. Cuando sea usado se aplicará la lógica de ese selector a partir de ese momento.

Un módulo se apoya normalmente en otros módulos, que importaremos



es el módulo dentro de angular que tiene toda la lógica para funcionar dentro de un Browser. Lo ha creado CLI.

Ahora si abres el archivo app.component.ts situado en la carpeta src/app y cambias el string de:

title = 'app';

por:

title = 'my wonderful app';

por poner un ejemplo, si ahora abres la página (si no tienes funcionando el comando ng serve, ejecútalo), verás que ahora en la página pone Welcome to my wonderful app!

Como ves, existe una variable llamada title (Typescript tiene inferencia de tipos y no hace falta que especifiques de que tipo es la variable) que automáticamente se pinta en el html, para ello si abres el archivo app.component.html verás que en la 4ª linea hay:

Welcome to {{ title }}!

Con {{ nombre\_variable }} puedes pintar variables creadas en el controlador (archivos ts) de su correspondiente componente (en este caso el componente es app).