ANGULAR

NODE CHEAT SHEET

COMANDO	DESCRIPCION
`node -v`	Indica la versión actual de node
`npm -v`	Indica la versión actual de npm
`nvm list`	Proporciona una lista con todas las versiones de node
`nvm use [version]`	Cambia la versión de node a la indicada por parámetros

ANGULAR CLI CHEAT SHEET

COMANDO	DESCRIPCION
`ng new [nombre]`	Crea un nuevo proyecto Angular dentro del directorio actual
`ng serve -o`	Inicia el servidor de desarrollo de Angular y el parámetro `--open` indica que se debe abrir automáticamente en el navegador
`ng generate component [nombre]`	Crea un nuevo componente con el nombre especificado

ANGULAR DIRECTIVAS CHEAT SHEET

DIRECTIVA	DESCRIPCION
`<router-outlet></router-outlet>`	Se utiliza para la navegación entre componentes, esta etiqueta se sustituirá por la vista del componente activo actual
`{{nombre-variable}}`	Se trata de interpolación, nos permite introducir el valor de una variable de nuestro controlador dentro de nuestra vista
`<a [routerLink="['ruta']"]>LINK</a>`	Se utiliza como atributo dentro de la etiqueta `<a></a>` y permite vincularlo con la ruta de otro componente. Debe usarse en combinación con `<router-outlet></router-outlet>`
`(click)= "nombre-funcion()"`	Permite vincular un elemento con una función de nuestro controlador que se ejecutará al hacer click

ANGULAR TYPESCRIPT CHEAT SHEET

CLASE	METODO	ESTATICO	DESCRIPCION
Router	`navigate(['ruta'])`	NO	Navega a la ruta que se le pasa por parámetro. Su función es la misma que la directiva `routerLink`

INTRODUCCION

QUE ES ANGULAR

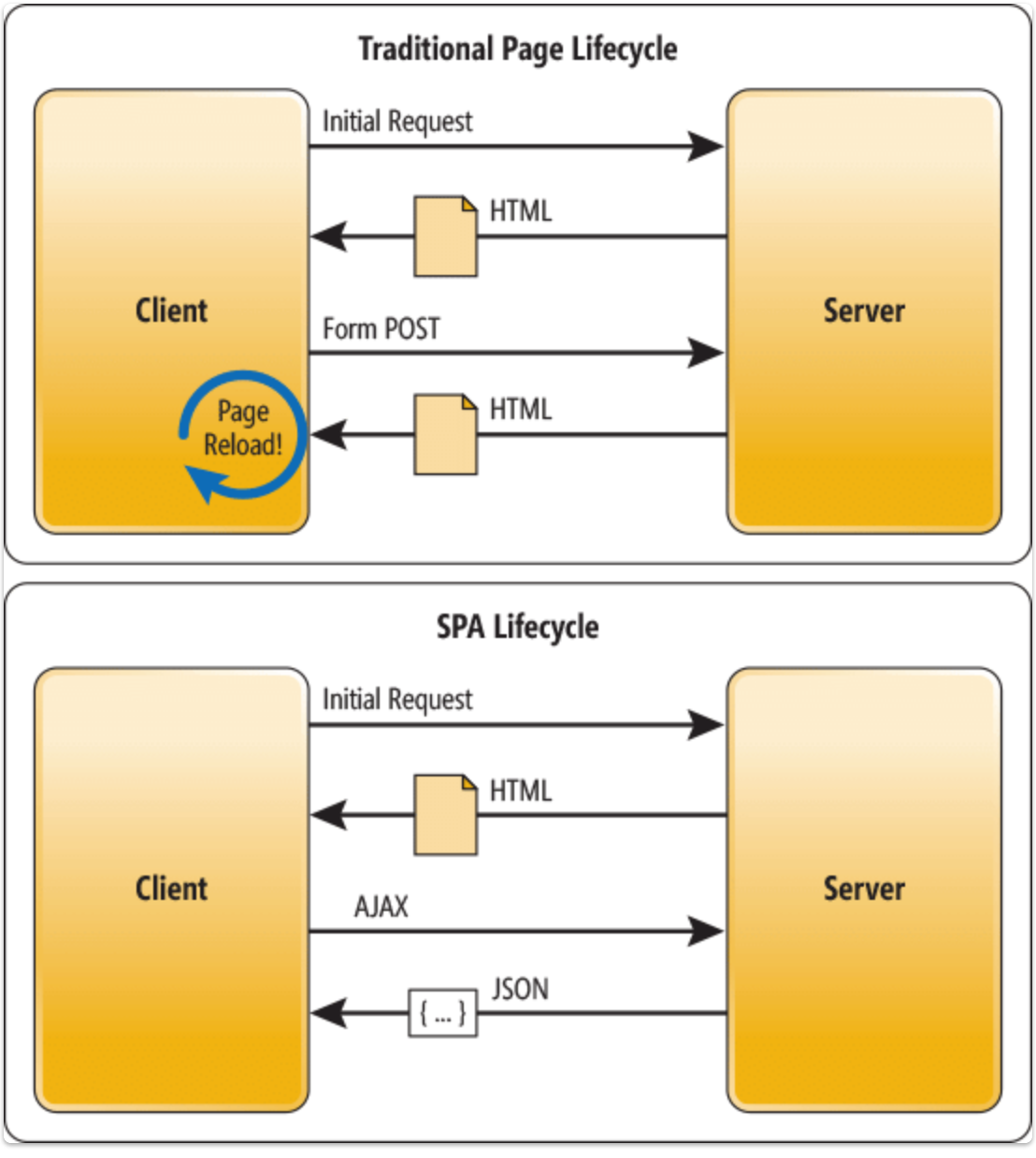
- **Angular es un *Framework* de *JavaScript/TypeScript*** que nos permite **desarrollar aplicaciones *front-end***

> *Importante: **TypeScript** es un **Framework** que nos permite desarrollar aplicaciones **JavaScript** más fácilmente*

> *Importante: Los navegadores solo son capaces de entender: **HTML/CSS/JavaScript***

- **Angular** nos permite desarrollar *Single Page Applicatione (SPA)*

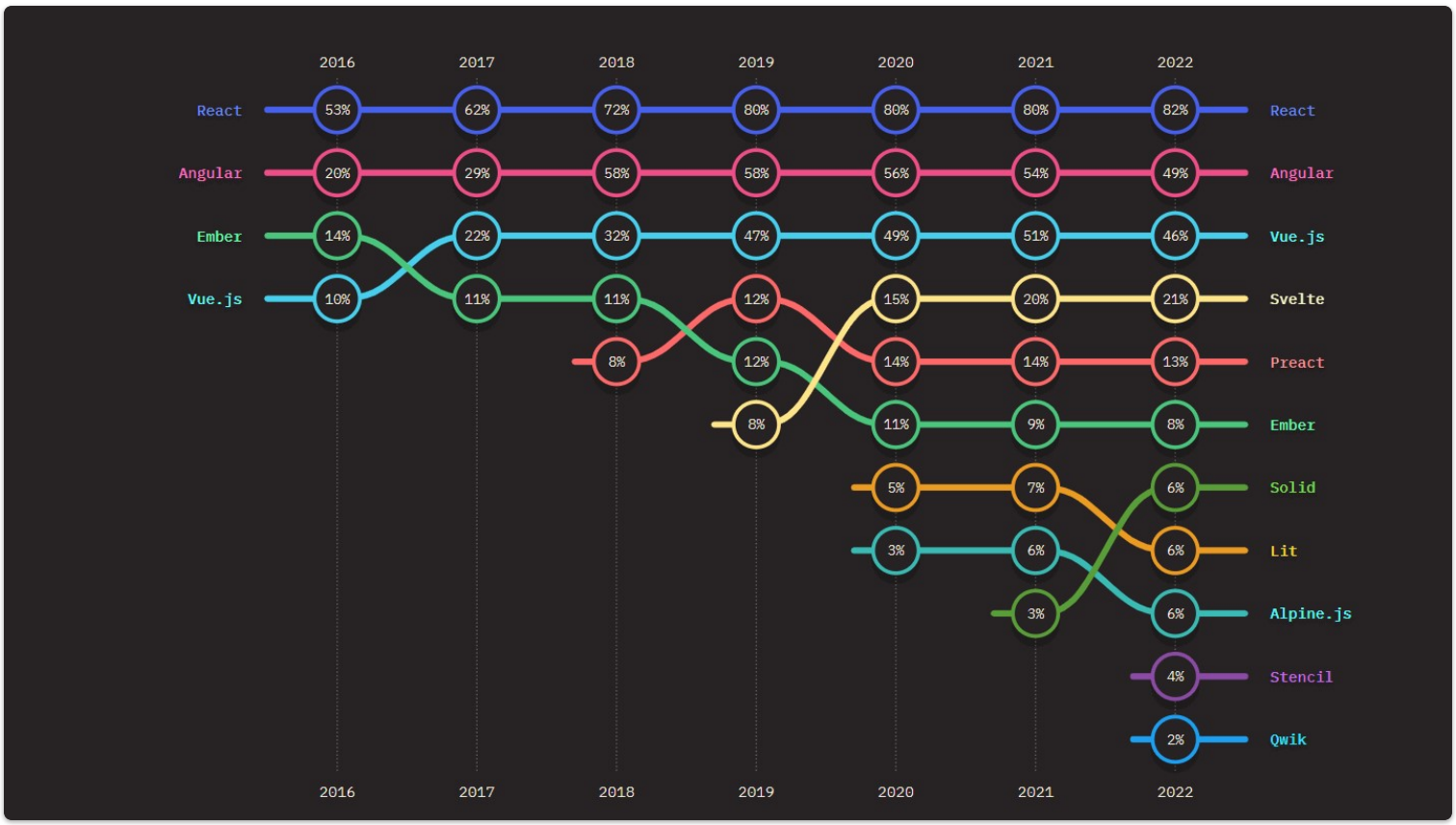
> *Importante: **SPA** permite **simular** la navegación entre recursos de una página web sin tener que descargar y cargar dichos recursos, esto, entre otras cosas **mejora enormemente el rendimiento y desarrollo** de nuestra aplicación web. Esto se consigue mediante **TypeScript***



- **Angular** dispone de una guía de estilo a la hora de desarrollar nuestra aplicación, esto hace que el **código sea más limpio, fácil y ágil de escribir**
- Se trata de un *Framework* muy extendido y con una gran comunidad detrás, facilitando así la obtención de ayuda e información en internet

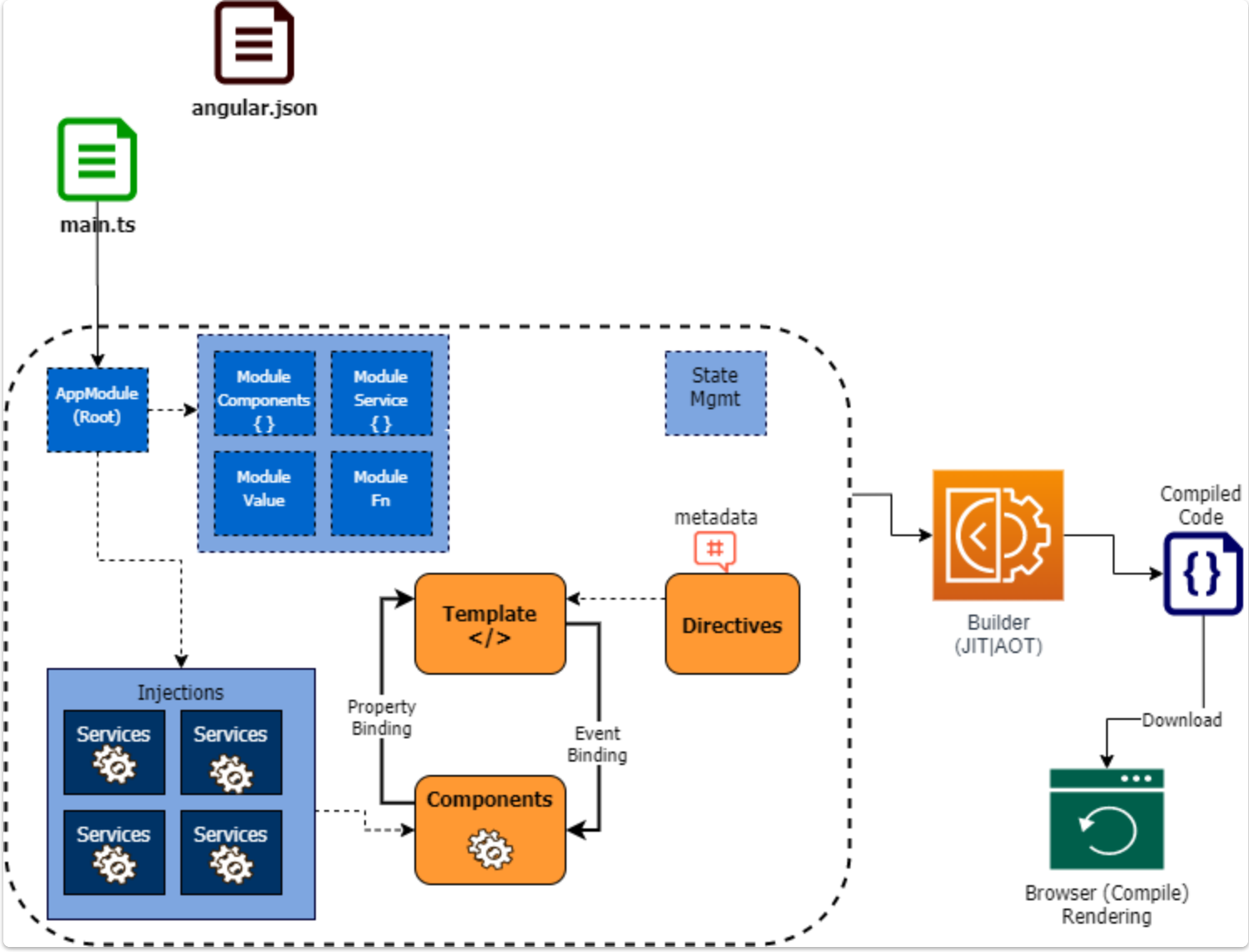
OTROS FRAMEWORKS DE JAVASCRIPT

Angular no es el único *Framework* de JavaScript front-end ni el más popular, existen muchos otros:



ARQUITECTURA

En este apartado se introduce la **arquitectura de una aplicación Angular**, con un enfoque en conocer sus piezas fundamentales para construir una aplicación web de front-end basada en Angular.



COMPONENTES

Es la unidad básica de construcción en un proyecto Angular, representa una página visual y su lógica.

Los **componentes** pueden estar acoplados y compartir código como parte de una visualización más grande.

> *Importante: Esto quiere decir que un componente puede estar formado a su vez por otro componente*

ESTRUCTURA DE UN COMPONENTE

Un componente se puede entender como la suma de los siguientes elementos:

ELEMENTO	DESCRIPCION
Template	Es la **parte visual** de nuestro componente: • Recursos **HTML** • Etiquetas de Angular (**directivas**)
Controlador	**Clases *TypeScript*** formadas por propiedades y métodos. Estas clases se encargan de

ELEMENTO	DESCRIPCION
	rellenar nuestra parte visual con información
Metadatos	Son propiedades que se definen en la clase del componente y que le permiten a Angular conocer y procesar información (selector, plantilla, estilos, servicios y otros). Los metadatos se definen mediante un decorador @Component y se utilizan para configurar y personalizar el comportamiento del componente en la aplicación Angular

SERVICIOS

Son clases cuya función es realizar **operaciones de lógica pesada**. Los servicios **se inyectan** en el constructor de los **controladores de un companente** que necesiten usar dicho servicio.

> *Importante: La función de un **controlador** es representar información y no de obtenerla o realizar cualquier otra función. Este trabajo siempre debe ser relegado a un **servicio***

DIRECTIVAS

Las directivas son un concepto del *Framework* de **Angular** que se utliza para **extender la funcionalidad** de los elementos HTML existentes o para **crear elementos personalizados** con comportamientos específicos.

> *Importante: Una directiva es realmente una clase de TypeScript decorada con alguna de las siguientes directivas:* `@Directive @Component @Pipe`

PIPES

Una **pipe** es una clase *TypeScript* decorada con la directiva `@Pipe`. La función de estas clases es **recibir datos, transformarlos y devolverlos al código llamador**. Al igual que ocurre con las directivas, podemos crear nuestos propios **pipes personalizados**.

MODULOS

Un módulo es una agrupación de un conjunto de componentes, servicios, directivas, pipes y otros módulos que proporciona un contexto de ejecución para estos elementos.

Un módulo está representado en Angular por una clase *TypeScript* decorada con la directiva `@NgModule`

La **unidad mínima** para una aplicación Angular debe contener por lo menos un módulo raíz llamado `app.module.ts` y un componente.

> *Importante: Una proyecto Angular **se agrupa en módulos según la funcionalidad**, esto permite tener nuestra aplicación **mejor estructurada, mejora la carga y permite la reutilización de los módulos en otros proyectos***

ENRUTAMIENTO

Es la capacidad que tiene una aplicación de Angular para **mostrar diferentes vistas según el contenido de la URL** actual del navegador. Esta funcionalidad es esencial a la hora de crear una *SPA*

INSTALACION DE ANGULAR

1. Instalar la versión LTS Node.js
2. Verificar instalación con los siguientes comandos:

```bash
node -v
npm -v
```

   Si todo ha ido bien, estos comandos deberían mostrar la versión de Node.js y npm
3. Instalar Angular CLI:

```bash
npm install -g @angular/cli
```

> *Importante: el parámetro -g indica que instalaremos Angular de manera global en nuestro sistema*

4. Verificamos nuestra instalación con el siguiente comando:

```bash
ng v
```

ESTRUCTURA DE UN PROYECTO ANGULAR

ELEMENTO	DESCRIPCION
node_modules/	Este directorio contiene todas las dependencias del proyecto, incluidas las bibliotecas de Angular y otras bibliotecas de terceros
src/	Este directorio contiene todo el código fuente de la aplicación
src/app/	Este directorio contiene todos los componentes, tuberías y otros elmentos que se crean
src/assets/	Este directorio contiene todos los recursos estáticos de nuestra aplicación, como las imágenes y archivos CSS
src/index.html	Este archivo es la página principal de nuestra aplicación
src/main.ts	Punto de entrada de la aplicación, encargado de cargar el módulo raíz. En este fichero podemos establecer el módulo que se cargará al arrancar la aplicación. Dentro de dicho módulo también se puede establecer el componente que se cargará
src/styles.css	Contiene los estilos CSS globales de la aplicación
angular.json	Configuración de Angular CLI para este proyecto
package.json	Contiene información sobre las dependencias y configuración del proyecto
tsconfig.json	Contiene la configuración del compilador *TypeScript* para el proyecto

ELEMENTO	DESCRIPCION
karma.conf.js	Karma es un *Framework* de pruebas que nos ayuda a probar nuestra aplicación, este fichero contiene su configuración

CONFIGURACION PARA MAXIMA COMPATIBILIDAD

Para hacer que nuestro proyecto sea compatible con la mayoría de navegadores debemos editar el fichero de configuración **tsconfig.json**:

```language-json
{
  "compilerOptions": {
    "target": "ES6",
    "module": "ES6",
    "lib": [
      "ES6",
      "DOM",
      "DOM.Iterable"
    ]
  }
```

COMPONENTES

En Angular CLI podemos crear un nuevo componente con el comando `ng generate component [nombre]`. Al ejecutarlo, Angular nos creará un directorio en la ruta **src/app** con el nombre de nuestro componente. Además de esto, dentro del directorio nos generará los siguientes archivos:

NOMBRE	DESCRIPCION
`[nombre].component.scss`	Aquí definiremos nuestra hoja de estilos para el componente
`[nombre].component.html`	Aquí definiremos nuestra estructura HTML
`[nombre].component.spec.ts`	Este fichero de *TypeScript* nos permite realizar pruebas
`[nombre].component.ts`	Este fichero será el controlador de nuestro componente

> *Importante: También se actualiza automáticamente el fichero TypeScript de nuestro módulo raíz, añadiendo una importación del componente creado*

CONTROLADOR

Este es el aspecto de un fichero `[nombre-componente].component.ts` al generar nuestro componente:

```
import { Component, OnInit } from '@angular/core';
// Aquí introducimos los imports...

@Component({ // Al igual que todos los componentes, debe tener la directiva @Component
        selector: 'app-heading', // Este es el nombre que tendrá nuestro elemento para poder referenciarlo en otros ficheros HTML
        templateUrl: './app-heading.component.html', // Ruta de nuestro fichero HTML que
```

```
pertenece al componente
        styleUrls: { './appheading.component.scss'  // Array que incluye las rutas de todos
nuestras hojas de estilo
})

// Los componentes implementas la clase OnInit
// El orden de ejecución es -> constructor -> ngOnInit
export class AppHeadingComponent implements OnInit {
        constructor () { } // En el costructor inyectaremos las clases que necesitemos

        ngOnInit() : void {

        }
}
```

ESTABLECER COMPONENTE DE INICIO

Para hacer que un componente sea el primero en cargar al abrir la aplicación debemos:

1. Establecer como módulo de inicio aquel que contiene el componente que queremos cargar dentro del fichero `src/main.ts`:

```language-typescript
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

import { AppModule } from './app/app.module';


platformBrowserDynamic().bootstrapModule(AppModule) // Como parámetro, introduciremos el
nombre de nuestro módulo en este método
  .catch(err => console.error(err));
```

2. Establecer el componente de inicio en el fichero `src/app/[nombre-módulo]/[nombre-módulo].module.ts`:

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

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';

@NgModule({
        declarations: [
                AppComponent
        ],
        imports: [
                BrowserModule,
                AppRoutingModule
        ],
        providers: [],
```

```
        bootstrap: [AppComponent] // Aquí introduciremos el componente que queramos cargar
})

export class AppModule { }
```

3. Para cargar el componente dentro de nuestra página, usamos la etiquete perteneciente en nuestro `src/index.html`:

```language-html
<!doctype html>
<html lang="en">
<head>
        <meta charset="utf-8">
        <title>Test</title>
        <base href="/">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
        <body>
          <app-heading></app-heading> <!-- Aquí introducimos nuestra etiqueta, en nuestro caso se llamaba app-heading -->
        </body>
</html>
```

INTERPOLACION

Podemos cargar propiedades de nuestro `[nombre-componente].component.ts` dentro de nuestro fichero HTML mediante interpolación, esto se hace mediante el uso de `{{nombre-variable}}` / `{{nombre-función}}`:

```language-html
<h1> Bienvenido a {{nombre-variable}} </h1>
```

ENRUTAMIENTO

BASES

El enrutamiento es lo que nos permite simular la navegación entre los diferentes recursos de nuestra página web y es la esencia de una SPA.

Cuando creamos por primera vez nuestro proyecto con Angular CLI, si establecemos que queremos usar enrutamiento, se nos creará automáticamente el archivo `src/app/app-routing.module.ts` que es esencial para establecer el enrutamiento en nuestra aplicación:

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

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

```
// Dentro de esta constanta es donde tenemos que añadir los diferentes componentes entre los que queremos navegar
const routes: Routes = [
        {path: '', component: HomeComponent},
        {path: 'about', component: AboutComponent}
];



@NgModule({

  imports: [RouterModule.forRoot(routes)],

  exports: [RouterModule]

})

export class AppRoutingModule { }
```

> *Importante: Dentro del array de rutas añadimos las rutas de la siguiente manera:*
> `{ path: [ruta en la URL], component: [nombre de la clase de nuestro componente] }`

Una vez establecidas las diferentes rutas, para poder hacer uso de la navegación debemos usar la **etiqueta reservada de Angular** `<router-outlet></router-outlet>`, esta etiqueta **se sustituira automáticamente** por el HTML de nuestro **componente activo**.

NAVEGACION CON ENLACES

En el apartado anterior hemos visto **cómo navegar entre componentes** mediante enrutamiento, sin embargo, esto solo nos permite cambiar de vista cambiando manualmente la ruta de nuestra **URL**.

En este apartado vamos a ver cómo podemos **navegar de manera dínamica mediante enlaces**.

Pongamos que en la vista de nuestro módulo raíz tenemos lo siguiente:

```
<ul>                                                                    language-html
        <li>
                <a>HOME</a>
        </li>
        <li>
                <a>ABOUT</a>
        </li>
</ul>
<router-outlet></router-outlet>
```

Lo que queremos conseguir es que al pinchar en los diferentes enlaces, el contenido de *router-outlet* se sustituya por el componente que nos interese.

Para conseguir esto debemos usar una **directiva** de Angular llamada *routerLink*:

```html
<ul>
        <li>
                <a [routerLink="['']"]>HOME</a>
        </li>
        <li>
                <a [routerLink="['about']"]>ABOUT</a>
        </li>
</ul>
<router-outlet></router-outlet>
```

> *Importante: el valor de **routerLink** debe ser igual al establecido en nuestro fichero de enrutamiento* `src/app/app-routing.module.ts`

> *Importante: Al igual que en el apartado anterior, debemos establecer nuestras rutas dentro del fichero* `src/app/app-routing.module.ts`

NAVEGACION CON CODIGO

Esta es otra forma de forzar la navegación sin el uso de enlaces `<a></a>`, por ejemplo, mediante botones.

Para conseguir esto, debemos **enlazar el elemento** con el que va a interactuar el usuario **con una función** de nuestro controlador *TypeScript*.

Dada la siguiente vista:

```html
<button>HOME</button>
<button>ABOUT</button>
<router-outlet></router-outlet>
```

Digamos que queremos navegar entre componentes igual que en el apartado anterior. Para ello tenemos que usar otra directiva llamada *click*:

```html
<button (click)= "navigateToHome()">HOME</button>
<button (click)= "navigateToAbout()">ABOUT</button>
<router-outlet></router-outlet>
```

A continuación que hacer lo siguiente en nuestro controlador:

- Para poder trabajar con **rutas** en nuestro controlador debemos importar la clase `Router` , crear una propiedad e inicializarla **inyectándola en el constructor**
- Debemos **crear los métodos definidos** anteriormente
- Dentro de nuestro métodos, usamos el método de la clase `Router`: `navigate()`

```typescript
import { Component } from '@angular/core';
import { Router } from '@angular/router'; // Importamos Router


@Component({
```

```
  selector: 'app-root',

  templateUrl: './app.component.html',

  styleUrls: ['./app.component.scss']

})

export class AppComponent {
        private router : Router // Creamos la propiedad router
        constructor(router: Router) {
                this.router = router;
        } // Inyectamos Router
        title = 'test';

        // Existe otra manera de inyectar una clase en TypeScript de la siguiente manera:
        // constructor(private router: Router){}

        // Definimos los métodos con el método de instancia de la clase Router
        navigateToHome(){
                this.router.navigate(['']);
        }

        navigateToAbout(){
                this.router.navigate(['about']);
        }
}
```