



Angular

Componentes (vistas)

Repasando ¿Qué es Angular?

Para debatir en clase!!

Repasemos, ¿Qué es angular?

- Es una plataforma y un **framework** para desarrollar aplicaciones web en TypeScript/JavaScript.
- De **código abierto**, mantenido por Google,
- Se utiliza para **crear y mantener** aplicaciones web de **una sola página (SPA)**.
- Su **objetivo** es facilitar el desarrollo de aplicaciones basadas en navegador
- Con **capacidad de implementar** el patrón de diseño Modelo Vista Controlador (**MVC**), en un esfuerzo para hacer que el desarrollo y las pruebas sean más fáciles.
- Angular **se basa** en clases tipo "**Componentes**", cuyas propiedades son las usadas para hacer el binding de los datos.
- En dichas clases tenemos propiedades (variables) y métodos (funciones a llamar) haciéndolo orientado a objetos (**P OO**).

Estructura y archivos de un Proyecto en Angular

- **package.json**: es el descriptor de dependencias npm.
- **.gitignore**: son los archivos y carpetas que git debería ignorar de este proyecto cuando se añada al repositorio.
- **index.html** que debe servir como página de inicio.

No obstante, no es exactamente el directorio raíz de publicación, porque al desplegar el proyecto los resultados de compilar todos los archivos se llevarán a la carpeta "dist".ts, que son código fuente TypeScript. Como quizás sepas, los archivos .ts solo existen en la etapa de desarrollo, es decir, en el proyecto que el navegador debe consumir no encontrarás archivos .ts

- **angular.json**, es el corazón de AngularCLI y contiene las configuraciones de Angular y opciones necesarias para trabajar con artefactos de angular tales como por ej. para generar componentes, pipes, service provider, clases, directivas etc. Define el convenio a seguir (pero es posible modificarlo)

Archivos y carpetas de un Proyecto en Angular

Carpetas

- **dist (directorio de distribución).** Se genera al compilar y es la versión de tu aplicación que subirás al servidor web para hacer público el proyecto. En dist aparecerán todos los archivos que el navegador va a necesitar y nunca código fuente en lenguajes no interpretables por él.
- **src:** carpeta que contiene el fuente de nuestra app.
- **e2e:** esta carpeta, denominada “end to end”, engloba una serie de ficheros cuya función es la ejecución de test automáticos. Se ejecuta con el comando **ng e2e**.
- **node_modules:** esta carpeta contiene todas las dependencias de nuestro proyecto.

Carpeta src (archivos)

- **styles.css:** define los estilos globales de la aplicación.
- **polyfills.ts:** permite que ciertos navegadores se comporten correctamente ante los CSS y los JavaScripts que se generan con Angular.
- **main.ts:** punto de partida de la app. Importante para el webpack dado que, en base a este archivo transpila los archivos TypeScript en Javascript y crea “chunks” (fragmentos) que representan parte de archivos o archivos completos según la configuración de compilación.

Carpeta src (subcarpetas y archivos)

- **enviroments:** permite configurar propiedades o configuraciones según el entorno (producción o desarrollo) y pueden ser consumidas por el código.
- **app.** carpeta dónde vamos a programar.
 - **app.modules.ts.** es el código que lanza la aplicación importando y exportando lo necesario para crear la aplicación. Los módulos son contenedores para almacenar componentes y servicios.
 - **archivos de componente.** es un bloque básico de construcción de una página web. Contiene una parte visual (`app.component.html` y `app.component.css`) y una funcional (`app.component.ts`)
- **assets:** contiene todos los archivos adicionales.

¿Qué es una web SPA?

Para trabajar en grupos!!

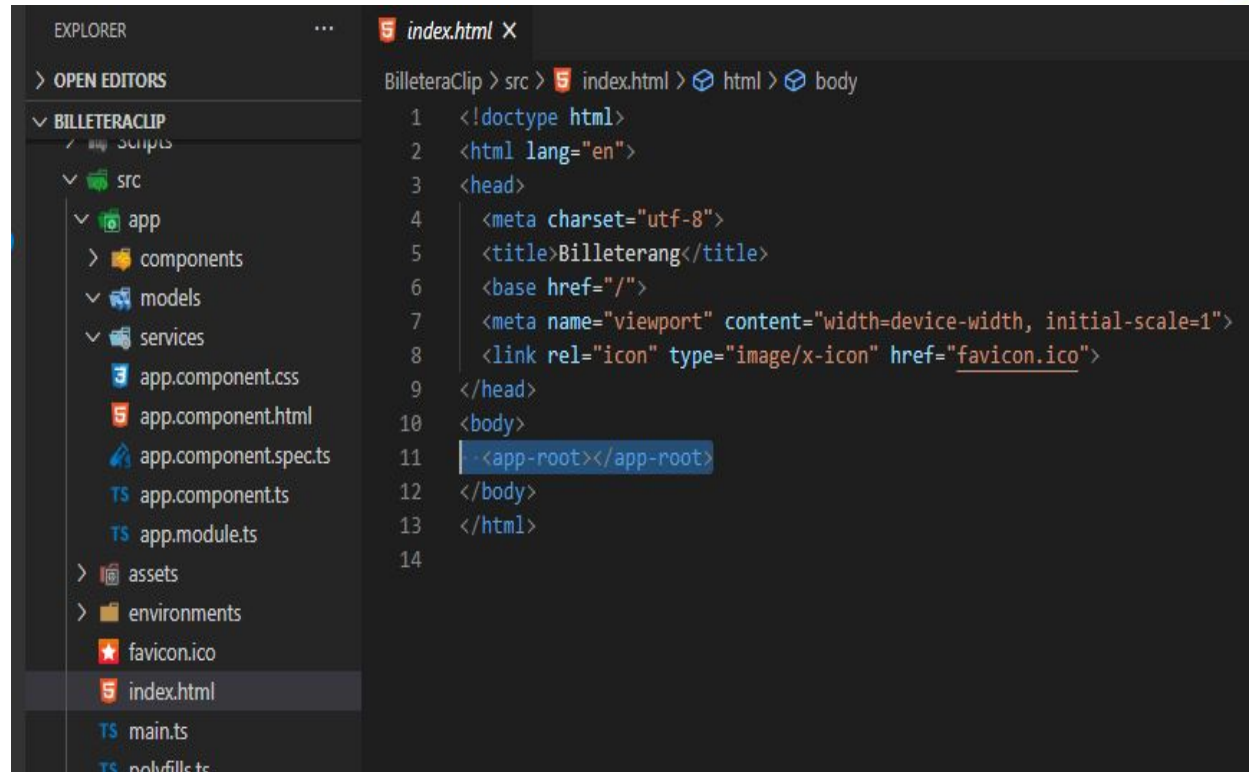


SPA

VS

MPA

Archivo Index.html
Con Angular se construyen aplicaciones SPA (Single Page Application) por lo que éste archivo es importante.



```
EXPLORER
...
BILLETERACLIP
  src
    app
      components
      models
      services
      app.component.css
      app.component.html
      app.component.spec.ts
      app.component.ts
      app.module.ts
    assets
    environments
    favicon.ico
    index.html
    main.ts
    polyfills.ts

index.html X
BilleteraClip > src > index.html > html > body
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8">
5   <title>Billeterang</title>
6   <base href="/">
7   <meta name="viewport" content="width=device-width, initial-scale=1">
8   <link rel="icon" type="image/x-icon" href="favicon.ico">
9 </head>
10 <body>
11   <app-root></app-root>
12 </body>
13 </html>
14
```

Una SPA se caracteriza por tener un único punto de entrada, generalmente el archivo index.html donde toda la acción se produce dentro del mismo index.html.

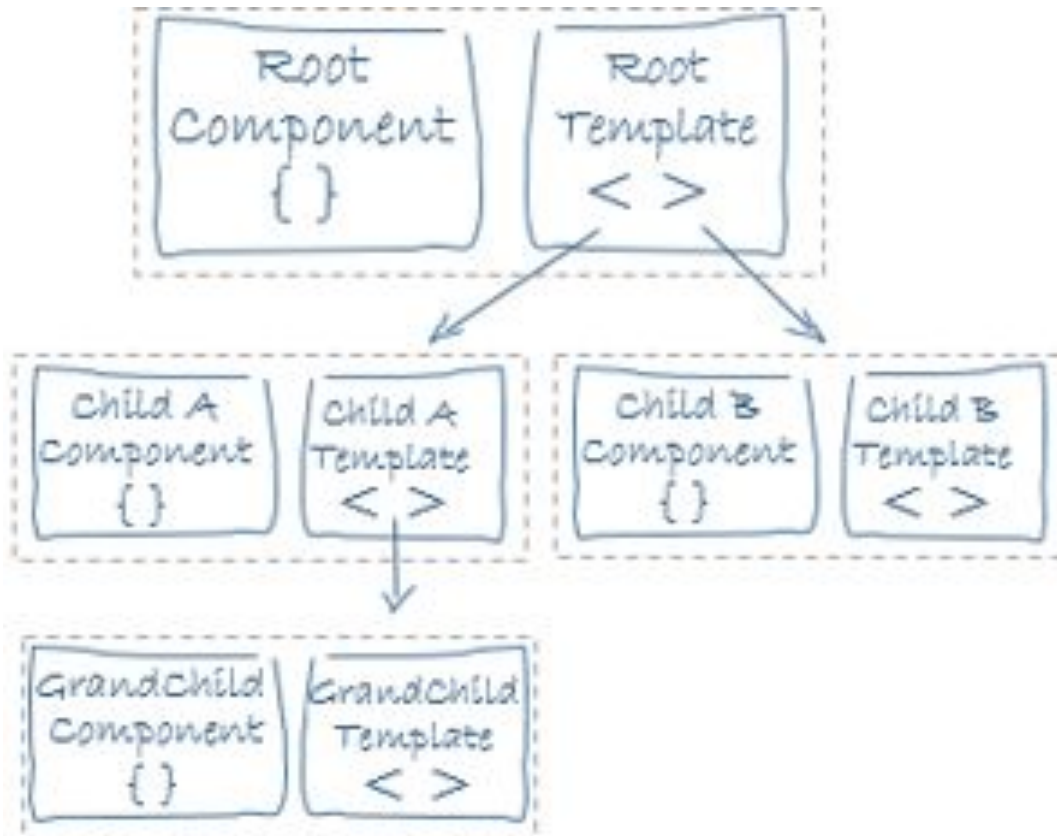
Funcionalmente, podríamos describir características de una SPA:

- **Punto de entrada central:** Un punto de entrada único, un que se va transformando y adaptando mediante acciones.
- **Página fija, Vistas cambiantes:** Como en el caso de una aplicación de escritorio, nos mantenemos en un “marco único” y fijo, mientras que “vistas dinámicas” van ofreciéndonos las distintas posibilidades del uso y navegación.
- **Página fija, no URL fija:** Es posible que la dirección URL sufra cambios en base a las actividades de uso de la plataforma y vaya modificándose aunque ese “*marco único*” se mantenga fijado. Esto es un tanto reduccionista (existen SPA que no transforman sus direcciones), pero es útil para comprender su mecánica.
- **Viajar ligera de equipaje:** Las peticiones cliente — servidor tienden a ser más laxas y más livianas que en el caso de una plataforma web al uso.

Datos, solo datos. Y además muchos procesos quedan del lado del navegador web del cliente a partir de diversas herramientas (como LocalStorage, por ejemplo). Datos y preferentemente en formato JSON.

Varias vistas, no varias páginas

Tenemos entonces un sólo documento HTML y muchas vistas.



Fuente de la imagen: <https://angular.io/guide/architecture-components>

Template y Vista

Un template es una bloque HTML que le dice a Angular cómo renderizar el componente por lo que, define la vista. Además, puede definir una jerarquía de vistas, las cuales contienen vistas embebidas, hosteadas incluso por otros componentes.

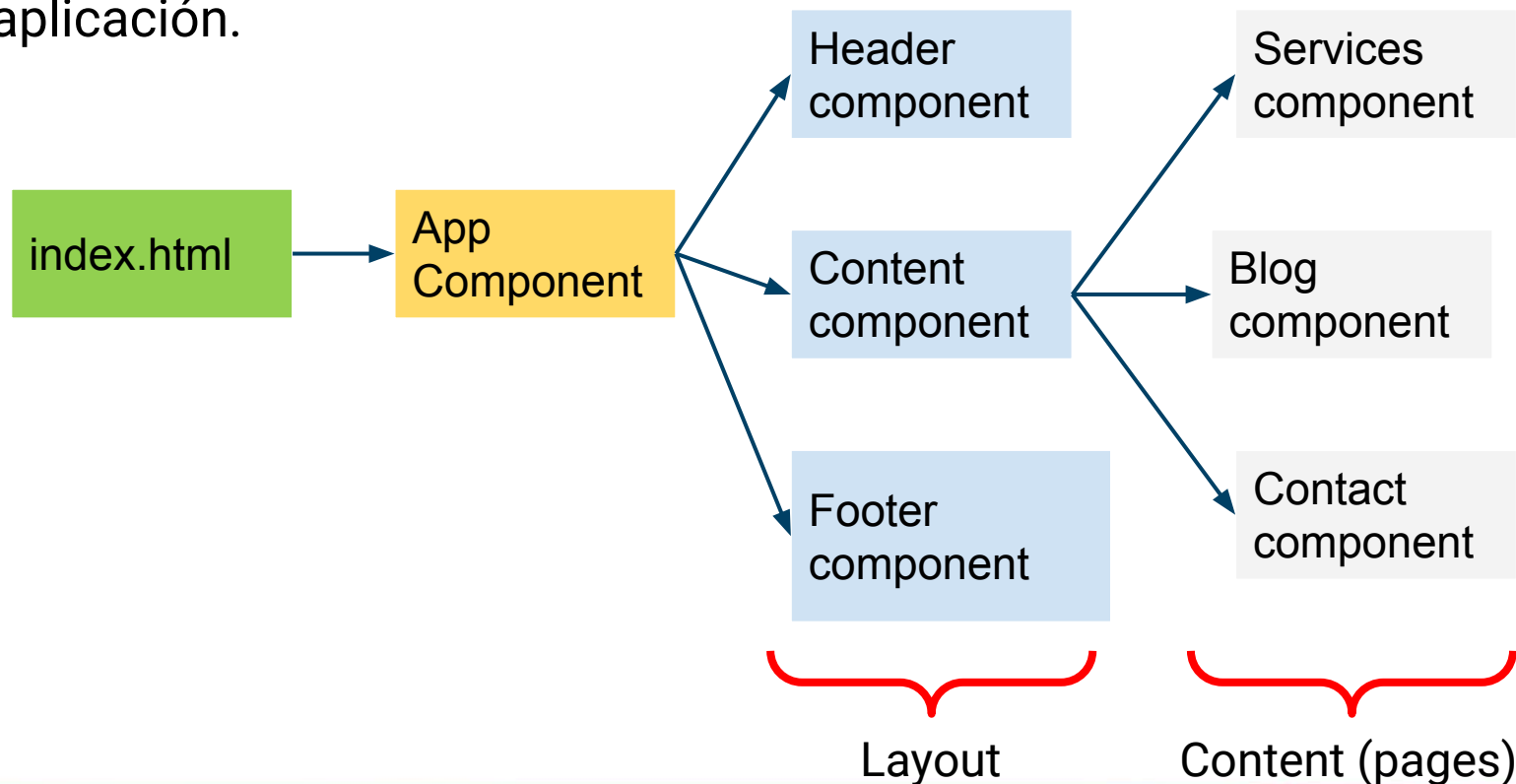


Fuente de la imagen: <https://angular.io/guide/architecture-components>

Las vistas permiten mostrar o ocultar bloques HTML de la interfaz de usuario.

Ejemplo de aplicación SPA.

Es relevante que sí necesitamos un index.html, pero realmente lo podríamos dejar casi vacío, porque indicaremos todo el comportamiento a partir del App Component, el módulo raíz de la aplicación.



¿Cuáles son sus ventajas y desventajas?

Para debatir en clase!!

Ventajas

Velocidad y rendimiento

- **Generación de código:** Angular convierte tus plantillas en código altamente optimizado para las máquinas virtuales de JavaScript de hoy en día, ofreciendo todas las ventajas del código escrito a mano con la productividad de un framework.
- **Universal:** Ejecuta la primera vista de tu aplicación en “node.js”, “.NET”, PHP, y otros servidores para renderizado de forma casi instantánea obteniendo solo HTML Y CSS.
- **División del código:** Las aplicaciones de Angular se cargan rápidamente gracias al nuevo enrutador de componentes. Este ofrece una división automática de códigos para que los usuarios solo carguen el código necesario para procesar la vista que solicitan.

Ventajas

Productividad

- **Plantillas:** Permite crear rápidamente vistas de interfaz de usuario con una sintaxis de plantilla simple y potente.
- **Angular CLI:** Las herramientas de línea de comandos permiten empezar a desarrollar rápidamente, añadir componentes y realizar test, así como previsualizar de forma instantánea la aplicación.
- **IDEs:** Obtén sugerencias de código inteligente, detección de errores y otros comentarios en la mayoría de los editores populares e IDEs.

Ventajas

Productividad

- **Testing:** Utiliza Karma para realizar pruebas unitarias, y Protractor para realizar pruebas end-to-end de forma rápida y estable.
- **Animación:** Permite crear animaciones complejas y de alto rendimiento con muy poco código a través de la intuitiva API de Angular.
- **Accesibilidad:** Posee características para crear aplicaciones accesibles con los componentes disponibles para ARIA.

<https://developers.google.com/web/fundamentals/accessibility/semantic-s-aria>

Desventajas

- **Se pierde el SEO**, ya que los robots no leen o interpretan el contenido cargado en segundo plano. (Hay formas de evitarlo pero no es lo natural, por ende no se aconsejan este tipo de aplicación para proyectos que necesitan presencia en la web tales como por ejemplo e-commerce).
- **La primer carga puede tardar un poco** ya que el peso del proyecto estará en función del tamaño del mismo.
- **Curva de aprendizaje** medio alto.
- Al ser un framework de javascript, si el usuario desactiva javascript en el cliente, nada será visible.

Si tuvieras que diseñar ahora la **landing page** de tu aplicación ¿cómo definirías el layout?

Para trabajar en equipos!

DEMO

Módulos

La forma más prolija de crear es componentes es a través de módulos ya que nos permite **agrupar componentes**.

Es decir, en lugar de colocar todos los componentes, directivas o pipes en el mismo módulo principal, lo agrupamos de una manera lógica, en función de nuestras preferencias.

Por ejemplo:

Podríamos crear un módulo que contenga los componentes semánticos de nuestra aplicación web inicialmente.

¿Cómo crear un módulo?

Para crear un módulo, ejecutar el siguiente comando:

ng generate module <<module-name>>

o su abreviado:

ng g m <<module-name>>

Una vez ejecutado el comando en nuestro proyecto, dentro de la carpeta "src/app" se crea un subdirectorio con el mismo nombre del módulo generado. Dentro encontraremos además el archivo con el código del módulo.

```
└─ src
  └─ app
    └─ nombre
       TS nombre.module.ts
```

Nota: Angular CLI aplica las convenciones de nombres más adecuadas y como los módulos son clases, internamente les coloca en el código la primera letra siempre en mayúscula.

Anatomía de un módulo

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

Como se puede observar, Angular define los módulos (que no son otra cosa que clases) a través del decorador @NgModule.

En el mismo, se puede observar dos arrays bien definidos:

- **imports:** importaciones que necesita
- **declarations:** aquí deberán estar listados todos los componentes u otros artefactos que incluye este módulo.

¿Qué es un componente?

Un componente es un contenedor, un bloque básico de construcción para las páginas web.

Es un elemento que está compuesto por:

- Un archivo template (html)
- Un archivo de lógica, (.ts)
- Un archivo para css, para estilos (.css)
- Un archivo para las pruebas unitarias (.spec.ts).

Typescript
(Lógica)

HTML
(Vista)

CSS
(Estilos)

Pruebas
Unitarias

Anatomía de un componente (metadata)

Los componentes, como el resto de artefactos en Angular, son clases TypeScript decoradas con funciones específicas. En este caso el decorador **@Component()**, define el componente identificando la clase y su metadata.

La metadata de un componente le dice a Angular dónde obtener los bloques de construcción que necesita para crear y presentar el componente y la vista.

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'app';
}
```

Archivo: app.component..ts

Analicemos un poquito el archivo de “.ts”

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

Importamos el componente

```
@Component({
```

Le damos nombre al
“selector” con este
llamaremos al componente

```
  selector: 'app-root',
```

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

Llamamos al template “html” del
component

```
  styleUrls: ['./app.component.css']
```

Llamamos a los estilos
del component

```
})
```

```
export class AppComponent {
```

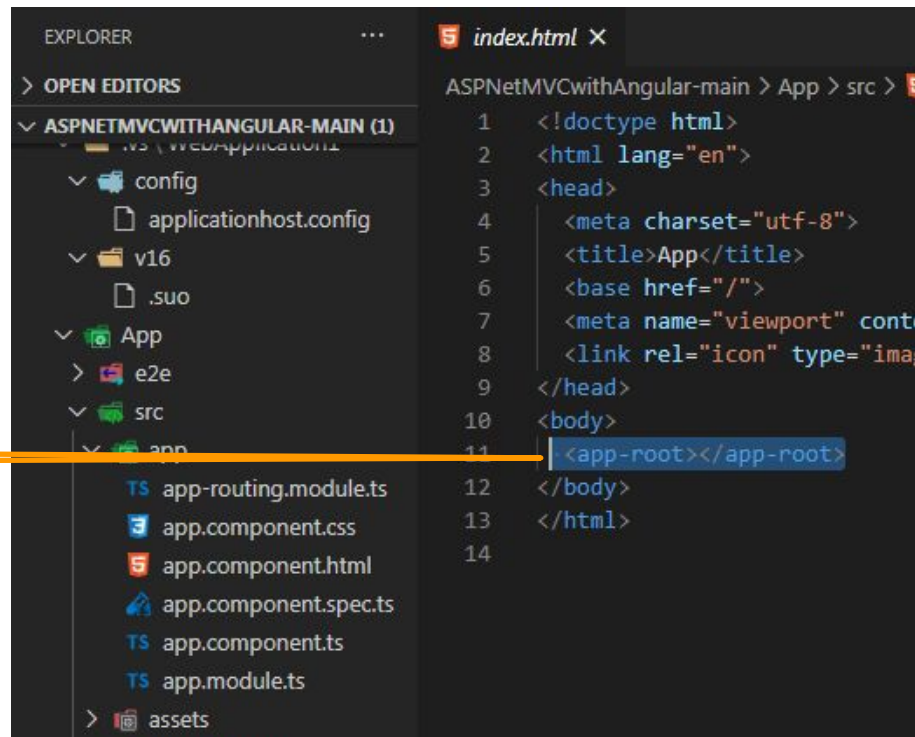
```
  title = 'app';
```

Exportamos la clase para que
sea utilizada al momento de
cargar la página

```
}
```

Selector llama al archivo root que está en Index.html

```
TS app.component.ts x
1  import { Component } from '@angular/core';
2
3  @Component({
4    selector: 'app-root',
5    templateUrl: './app.component.html',
6    styleUrls: ['./app.component.css']
7  })
8  export class AppComponent {
9  }
10
```



EXPLORER

ASPNETMVCWITHANGULAR-MAIN (1)

- config
 - applicationhost.config
- v16
 - .suo
- App
 - e2e
 - src
 - app
 - app-routing.module.ts
 - app.component.css
 - app.component.html
 - app.component.spec.ts
 - app.component.ts
 - app.module.ts
- assets

index.html X

ASPNetMVCwithAngular-main > App > src >

```
1  <!doctype html>
2  <html lang="en">
3  <head>
4    <meta charset="utf-8">
5    <title>App</title>
6    <base href="/">
7    <meta name="viewport" content="width=device-width, initial-scale=1">
8    <link rel="icon" type="image/png" href="assets/icon.png">
9  </head>
10 <body>
11   <app-root></app-root>
12 </body>
13 </html>
14
```

Los componentes definen **nuevas etiquetas HTML** para ser usados dentro de otros componentes. Excepcionalmente en este caso por ser el componente raíz se consume en el página index.html. El nombre de la nueva etiqueta se conoce como selector. En este caso la propiedad selector: "app-root" permite el uso de este componente dentro de otro con esta invocación `<app-root></app-root>`. En este caso el componente raíz.

```
app.module.ts - app - Visual Studio Code

app.module.ts X

src > app > app.module.ts > ...
1  import { BrowserModule } from '@angular/platform-browser';
2  import { NgModule } from '@angular/core';
3
4  import { AppRoutingModule } from './app-routing.module';
5  import { AppComponent } from './app.component';
6
7  @NgModule({
8    declarations: [
9      AppComponent
10   ],
11   imports: [
12     BrowserModule,
13     AppRoutingModule
14   ],
15   providers: [],
16   bootstrap: [AppComponent]
17 })
18 export class AppModule { }
19
```

Además de incluirlo en el index (a través de la etiqueta) ya que, está destinado a ser usado en la página principal, en el index.html. Eso obliga a registrarlo de una manera especial en el módulo raíz. Hay que incluirlo en el array bootstrap: [AppComponent], es ahí donde se incluyen los componentes con la capacidad de lanzar bootstrap la aplicación.

Sintaxis del Template

Un template luce como un HTML regular, excepto que contiene además, [sintaxis propia de angular](#).

Ejemplo:

```
<h1>Angular Sintaxis</h1>
  <p>Con la sintaxis propia de angular, puedes extender el vocabulario HTML de tus aplicaciones web.
  Por ejemplo, Angular te ayuda a obtener y establecer elementos DOM dinámicamente con funciones
  tales como build-in, data binding, entre otros</p>
  <p>Además, si bien casi todos los elementos HTML son válidos en los templates de angular y dado
  que estamos hablando de un template y no de una entera página web, no necesitas concentrarte
  en los elementos html, body, link, etc. sino que puedes hacer foco en lo que estás desarrollando.</p>
  <ul>
    <li *ngFor="get list" (click)="selectList(option)">
      {{option.name}}
    </li>
  </ul>
  <app-option-detail *ngIf="selectedOption" [option]="selectedOption"></app-option-detail>
```

¿Cómo crear un componente?

Para crear un componentes, ejecutar el siguiente comando:

ng generate component <<component-name>>

o su abreviado:

ng g c <<component-name>>

Nota: puedes saltarte algunos elementos utilizando --skip

¿Cómo crear un componentes dentro de un módulo?

Para crear un componentes, ejecutar el siguiente comando:

ng generate component <<module-name/component-name>>

o su abreviado:

ng g c <<module-name/component-name>>

Nota: puedes saltarte algunos elementos utilizando --skip

Exportar del módulo hacia afuera

Si deseamos que este módulo pueda utilizarse desde otros módulos y exponer cosas hacia afuera deberemos agregar nueva información (componentes a exportar) al decorador del módulo: el array de exports.

```
@NgModule ({  
  declarations: [ HeaderComponent, ContentComponent,  
FooterComponent ],  
  imports: [  
    CommonModule  
  ],  
  exports: [ HeaderComponent, ContentComponent,  
FooterComponent ]  
})  
  
export class LayoutModule { }
```

¿Cómo usar el componente en otros módulos?

Para importar el componente, debemos importar es el módulo entero donde se ha colocado ya que, el propio módulo hace la definición de aquello que se quiere exportar en "exports".

Requiere varios pasos:

1. Importar el módulo

```
import { LayoutModule } from './ui/ui.module';
```

¿Cómo usar el componente en otros módulos?

2. Declarar el módulo en el decorador @NgModule/imports del módulo hacia dónde lo queremos llevar.

```
@NgModule ({  
  declarations: [  
    AppComponent,  
  ],  
  imports: [  
    BrowserModule,  
    AppRoutingModule,  
    LayoutModule  
  ],  
  providers: [],  
  bootstrap: [AppComponent]  
})
```

¿Cómo usar el componente en otros módulos?

3. Usar el componentes HTML mediante el tag.


```
<app-mi-componente></app-mi-componente>
```

Haciendo uso de módulos y componentes intenta llevar todo lo que has desarrollado en el index con bootstrap a módulos y componentes:




Programa PIL

PIL Home ¿Quienes somos? Iniciar Sesión

Sección Principal



Escribe tus comentarios en nuestras redes sociales!

@Programa PIL 2021

Para trabajar en equipos!!



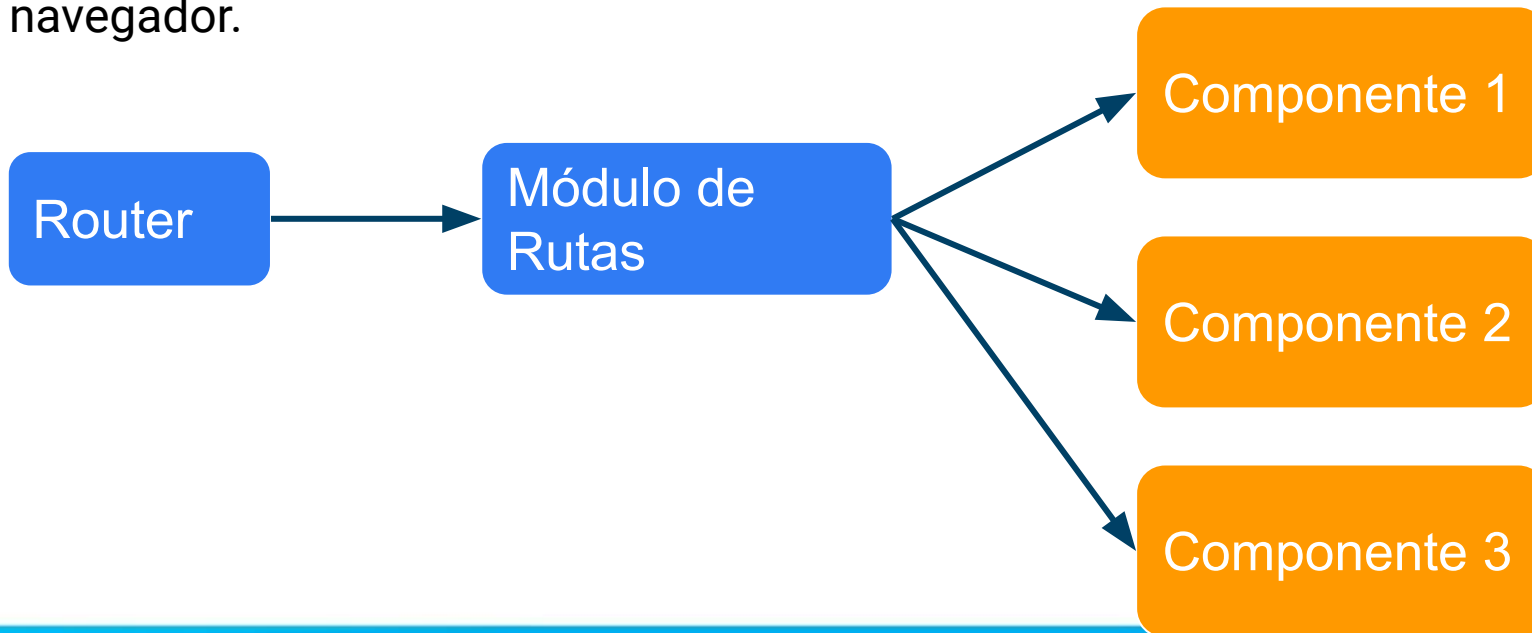
Angular

Sistema de Routing

Sistema de Routing

El Sistema de Routing de Angular tiene por objeto mostrar las vistas de la aplicación en función de una ruta "virtual".

Las rutas las denominamos "virtuales" puesto que, realmente sólo existe un "index.html", no habrá un "contacto.html", "blog.html" u otro archivo *.html para cada ruta, sino que siempre será el "index.html" el que se entregue al navegador.



Sistema de Routing

Para usar esta funcionalidad, debemos acceder al archivo “app-routing.module.ts” de la aplicación de angular y realizar los siguientes pasos:

1. Definir las rutas como sigue:

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { HomeComponent } from '../pages/home/home.component';
import { IniciarSesionComponent } from '../pages/iniciar-sesion/iniciar-sesion.component';
import { QuienesSomosComponent } from '../pages/quienes-somos/quienes-somos.component';

const routes: Routes = [ {path: 'home', component: HomeComponent},
{path: 'iniciar-sesion', component: IniciarSesionComponent},
{path: 'quienes-somos', component: QuienesSomosComponent}
],

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```


Sistema de Routing

```
import { IniciarSesionComponent } from
'./pages/iniciar-sesion/iniciar-sesion.component' ;
import { QuienesSomosComponent } from
'./pages/quienes-somos/quienes-somos.component' ;

const routes: Routes = [ {path: 'home', component:
HomeComponent},
{path: 'iniciar-sesion', component:
IniciarSesionComponent },
{path: 'quienes-somos', component:
QuienesSomosComponent },

];

@NgModule ({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})

export class AppRoutingModule { }
```

path: La ruta a que deseamos configurar.

component: El componente asociado a esa ruta. Para que funcione tienes que importar el componente en la parte de arriba.

pathMatch: Esto es opcional, significa que toda la ruta URL tiene que coincidir (no solo cierta parte).

Sistema de Routing

2. Luego, (si creaste el proyecto de angular sin routing) debes importar las rutas en el archivo “app.module.ts”. Para ello importamos la ruta y lo añadimos, esta vez en la parte de imports:

```
import { AppRoutingModule } from './app-routing.module';  
import { AppComponent } from './app.component';  
import { LayoutModule } from './layout/layout.module';  
import { PagesModule } from './pages/pages.module';  
  
@NgModule({  
  declarations: [  
    AppComponent  
  ],  
  imports: [  
    BrowserModule,  
    AppRoutingModule ,  
    LayoutModule,  
    PagesModule],  
  providers: [],  
  bootstrap: [AppComponent]  
})  
export class AppModule { }
```

¿Cómo podemos poner links a páginas de nuestra web?

3. Agregar los enlaces utilizando el atributo **routerLink** como se muestra a continuación:

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
  <div class="container-fluid">
    <a class="navbar-brand" >PIL</a>
    <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
data-bs-target="#navbarSupportedContent" aria-controls="navbarSupportedContent"
aria-expanded="false" aria-label="Toggle navigation">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarSupportedContent">
      <ul class="navbar-nav me-auto mb-2 mb-lg-0">
        <li class="nav-item">
          <a class="nav-link" routerLink="/home" routerLinkActive="active"
aria-current="page">Home</a>
        </li>
        <li class="nav-item">
          <a class="nav-link" routerLink="/quienes-somos" routerLinkActive="active"
aria-current="page">¿Quienes somos?</a>
        </li>
        ...
      </ul>
    </div>
  </div>
</nav>
```

¿Qué es el Router outlet de Angular?

Para debatir en clase!!

Router Outlet

router-outlet es una etiqueta especial en Angular que sirve para mostrar los componentes hijos de un componente. Por defecto todos los componentes son hijos del componente AppComponent, por lo que si incluimos esta etiqueta dentro de la vista de AppComponent, se renderizará cada uno de los componentes del routing dependiendo de la página en la que nos encontremos.

```
<router-outlet></router-outlet>
```

Router Outlet

4. Por último, agregar las etiquetas **<router-outlet></router-outlet>** en el html dónde deseo que aparezca el ruteo.


```
<app-header></app-header>
<app-navbar></app-navbar>
<div class="row">
  <div class="col-8">
    <router-outlet></router-outlet>
  </div>
  <div class="col-4">
    <app-aside></app-aside>
  </div>
</div>
<app-footer></app-footer>
```



Rutas por Defecto

Para configurar el redireccionamiento al un componente por defecto, como por ej. al home, configurar el archivo “app-routing.module.ts” como sigue:

```
const routes: Routes = [ {path: 'home', component: HomeComponent},  
  {path: 'iniciar-sesion', component: IniciarSesionComponent},  
  {path: 'quienes-somos', component: QuienesSomosComponent},  
  {path: '', redirectTo: '/home', pathMatch: 'full'}  
];
```



```
@NgModule({  
  imports: [RouterModule.forRoot(routes)],  
  exports: [RouterModule]  
})  
  
export class AppRoutingModule { }
```

Agregando una Página 404

Para configurar la página 404 (not found), configurar el archivo “app-routing.module.ts” como sigue:

```
const routes: Routes = [ {path: 'home', component: HomeComponent},  
  {path: 'iniciar-sesion', component: IniciarSesionComponent},  
  {path: 'quienes-somos', component: QuienesSomosComponent},  
  {path: '', redirectTo: '/home', pathMatch: 'full'},  
  {path: '**', component: PageNotFoundComponent}  
];  
  
@NgModule({  
  imports: [RouterModule.forRoot(routes)],  
  exports: [RouterModule]  
})  
export class AppRoutingModule { }
```



¿Cómo crear rutas con partes dinámicas en Angular?

Ejemplo: Deseamos mostrar los movimientos de una Cuenta

Para debatir en clase!!

Rutas con partes dinámicas en Angular

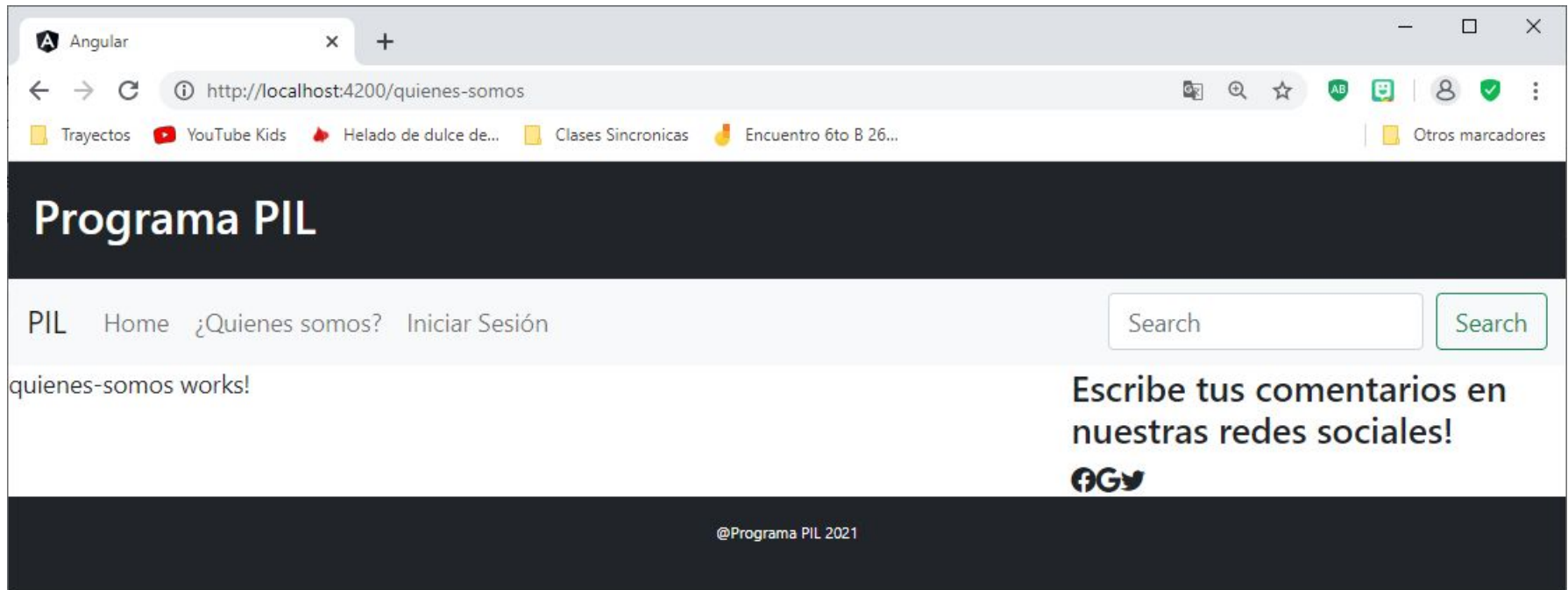
En lugar de crear cada una de esas rutas (cosa que no es recomendable), angular nos permite configurar cierta parte de la ruta dinámica. Para ello, configurar el archivo “app-routing.module.ts” con el path con la `/:<nombre de variable>` como sigue:

```
{path: 'cuenta', component: CuentaComponent},  
{path: 'cuenta/:id', component: CuentaDetalleComponent},
```

Con esto, se crean todas las rutas posibles que empiecen con /cuenta. El nombre `id` sirve para guardar en esa variable la ruta específica en la que estamos, es decir, luego dentro del componente `CuentaDetalleComponent` puedes recoger el valor para saber en qué ruta estás. Para ello, dentro del

```
constructor() {  
    console.log(route.snapshot.params['id']);  
}
```

Utilizando el sistema de Routing de Angular agrega funcionalidad a la barra de navegación:



Para trabajar en equipos!!

¿Cómo crear rutas hijas?

Rutas hijas

Para definir rutas con una o más rutas hijas, configurar el archivo “app-routing.module.ts” como sigue:

```
const routes: Routes = [ {path: 'home', component:
 HomeComponent},
 {path: 'iniciar-sesion', component: IniciarSesionComponent},
 {path: 'quienes-somos', component: QuienesSomosComponent},
 {path: '', redirectTo: '/home', pathMatch: 'full'},
 {path: '**', component: PageNotFoundComponent},
 {path: 'admin', component: AdminComponent,
 children: [
   {path:'', component: AdminMainComponent},
   {path:'propiedades', component: PropiedadesComponent}
 ]}
 ];
```



Referencias

<https://angular.io/guide/architecture-components>

<https://angular.io/guide/router-tutorial>

<https://angular.io/api/router/Route>