**ANGULAR**

MVC

* Petición 🡪 código PHP alojado en la raíz del dominio
* Respuesta 🡪 el server construye todo el HTML en cada petición

MV1V2M es el patrón más utilizado por las Single Page Application (utilizado por Angular JS)

* 1ª Petición 🡪 URL alojado en el dominio.
* 1ª Respuesta 🡪 el servidor envía respuesta con el contenido HTML, CSS y JS con toda la información que no vamos a usar todo el rato y que por tanto no habrá que volver a peticionar.
* 2ª Petición 🡪 Por Sockets o por XML HttpRequest. El servidor solo devuelve el contenido de la base de datos en un JSON, y desde Front existe el JS necesario para asociar ese contenido mediante **enrouting**

Socket: canal de comunicación permanente entre máquinas a través de un puerto. El puerto determina el protocolo de comunicación. **Socket = ip:port**

Con los sockets

Google se casó con Microsoft poniéndole los cuernos a Facebook y Facebook se cabreó sacando su React de los cojones.

Otros patrones de arquitectura:

* MVPresentación
* Arquitectura por Componentes es el patrón utilizado por Angular 2

AC

* También utiliza enrouting
* El estado de la aplicación es el valor de todos los campos y variables a tiempo real (ejemplo: guardar y cargar).

En las SPAs el estado de la aplicación se mantiene siempre, las SPAs son persistentes y por tanto si recargo página, o cierro y vuelvo abrir, el estado se recupera!

Entre Angular 2 y Angular JS hay mucho cambio pero el routing y las directivas se mantuvieron igual

Otros Frameworks:

* Backbone.js (MVVM)
* Bue
* Redux

cli 🡪 Command Line Interface

cd Proyectos

git clone <https://github.com/angular/quickstart.git>

cd quiqkstart

(eliminar la carpeta)

npm install

npm start

CARPETAS DENTRO DE / quickstart-master

\*Karma sirve para pasar pruebas “unitarias” Jasmine)

\*Protractor sirve para pasar pruebas end to end

Significa probar todos estos campos de una vez (de principio a fin)

1-Rellenar formulario

2-Validar form

3-Enviar respuesta

4-Proceso de respuesta

\*(linter)//Programa que valida si pasa una serie de pruebas que predefino yo, para los estilos, no en el código( ex. Que no supere 14 caracteres)//

\*Lintado: son reglas de estilo para el typescript. El transpilador puede quejarse o no.

\*Bs-Config: Le dice al servidor donde está la carpeta base y la carpeta raíz

SOA: Arquitectura creada por Servicios

COMPONENTES: (/Son los trocitos que componen mi APP)

app : Su función es linkar el resto de componentes. De la App cuelga todo el proyecto -🡪 de cada componente “cuelgan” otros servicios.

Por cada componente nuevo se creara una carpeta nueva.

(Cada componente tiene su propio MVC, Es decir su html, css, js)

En la carpeta src estará mi proyecto entero

<https://angular.io/guide/quickstart>

1. Instalar el CLI de Angular si no está instalado ( **$ ng -v** )
2. Crear mi nuevo proyecto Angular: $ cd Proyectos
3. **$ ng new** foldername (este comando también realiza un git init)
4. Abrir el repositorio git que me ha creado ng
5. **$ ng serve –open**

**ng –v** Ver las versiones del comando ng **ng --version**

Para lanzar procesos en segundo plano (background) por la bash se escribe un **&** al final del comando y después Intro!

El proceso en segundo plano lleva asociado un pid [1] pidNumber (Process ID)

* Para finalizar un proceso en segundo plano ejecutar el comando **$ kill --9**

Aunque al usar una consola de Unix en una máquina de Windows, éste último no permite que se cierre el proceso. Ctrl+shift+Esc 🡪 Node js 🡪 finalizar proceso

**$ ps** Equivale a Ctrl+shift+Esc

**$ ps aux** 🡪 muestra todos los procesos en ejecución

1. **$ ng serve –open &**

**$ ps aux | grep**

# src

* index.html 🡪 llama a app component (selector HTML app…)
* styles.css 🡪 común a todos los componentes (font-face)
* main.ts 🡪 también para cosas globales
* app
  + Componentes

Los comandos del cli (ng) se ejecutan siempre en la carpeta raíz

**$ ng generate component nombreDelComponente**

Todos los componentes estarán dentro de la carpeta src, anidados al mismo nivel (sibblings)

# Definición de la arquitectura de componentes

Definir un componente es definir las propiedades y métodos que va a tener.

Servicios 🡪 otros objetos que no forman parte de la arquitectura de componentes (no tienen vista, no tienen HTML y CSS)

Los componentes tienen servicios 🡪 **inyecciones de dependencia**.

[example JSON dependecies]

The CLI created the first Angular component for you. This is the root component and it is named app-root. You can find it in ./src/app/app.component.ts.

# Crear clases por consola de comandos

**$ ng generate class className**

**NG crea un fichero className.ts y lo guarda en el fichero ts en src/app**

**export class className {**

**id: number;  
 name: string;**

**}**

**En el componente donde queramos utilizar dicha clase**

**import {className} from ‘ruta./className’**

# Filters

Documentación oficial 🡪 **pipes |**

Performed at html file

# [ ( Banana in a box ) ] 🡪 Directiva del modelo ¿y otras?

Doube binding = 2way binding 🡪 View & Model are listening each other

A data input from the client must be connected with the model data as output and viceversa.

* The Banana In A Box is inserted in HTML tags (input types, etc). Inside the Banana in a box insert a “*directive”* **in the component.html file (plantilla)**
* **In the component.ts file** import the FormModules from @angular/forms
* **In app.module.ts file** add again the import twice (at the begininning and in the import sub-JSON of the NgModule JSON

# Directivas

**<div \*ngFor=”let variableName of arrayName>**

<!-- Replica todo el bloque de código que haya dentro del div tantas veces como elementos tenga el array y una vez para cada elemento -->

**<div \*ngIf=”js condition with a comparison operator>**

Otras directivas famosas son las asociadas al concepto **evnt binding** and **class binding**

# ( event binding )

Se recomiendo declarar las *funciones manejadoras* con la nomenclatura onEvent

<button (event)=”funcionManejadora(parameter)”>Editar</button>

# [ class binding ]

Declare a .class of the component at the **component.html file** and define it at the **component.css file**. And then apply the class depending on the comparison condition

[class.className]=”comparison condition”

# Enrouting = navigation menú

The enrouting concept exists in all frameworks, being for SPA or not, back, front, etc

The CLI to generate it is:

**$ ng generate module app-routing --flat --module=app**

What does this command make?:

* Imports the module in the **app.module.ts file**
* Generate a new file in app directorio named **app-routing.module.ts file**

**app-routing.module.ts**

This is the main file where all components are imported by the directory path. This main file (**app-routing.module.ts**) is generated inside app directory.

This file is generated with two imports and one NgModule JSON by default. However we have to add more code manually:

* **import { RouterModule, Routes } from ‘@angular/router’;**
* Import all components called in Routes []

Desde este fichero se importan todos los componentes con las rutas del directorio

* **const routes: Routes = [  
  { path: ‘’, redirectTo: ‘/firstURLparameter’, pathMathc: ‘full’ },  
  { path: ‘/firstURLparameter’, component: IndexComponent },  
  { path: ‘/secondURLparameter’, component: NameComponent },**

**]**

* **imports: [   
  CommonModule,  
  RouterModule.forRoute (routes)  
  ]**
* **exports: [ RouterModule ]** as a new key: value of the NgModule JSON

**app.component.html**

* **<app-newComponentName></app-newComponentName>**
* **<router-outlet></router-outlet>**

**app.component.ts**

**app.menu.html** 🡪 Los botones del mando (MENU COMPONENT)

**<nav>**

**<a routerLink= firstURLparameter’  
 <a routerLink= seconfURLparameter’**

**Tantos routerLinks**

**</nav>**

TODO LO QUE HASTA AHORA SE HACÍA EN app.component.ts, se debe de hacer en el fichero typescript del componente al que enruta el path vacío

En el Frontal estamos consumiendo los datos en crudo desde un mock-nombre.ts pero en el futuro debe ser una API Rest y habrá que tomar los datos desde back-end.