**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 en**routing**

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 componentes 🡪 COMP1, COMP2, COMP3.

Por cada componente nuevo se creara una carpeta nueva.

(Cada uno de estos tres subcomponentes tiene su propio MVC, Es decir su html, css, js)

En la carpeta src estará mi proyecto entero

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**

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
* styles.css🡪 común a todos los componentes(x ejemplo 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 estáran 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.

Los métodos se implementan en las clases.

Servicios🡪 Inyección de dependencia.(Son objetos, que no forman parte de mi arquitectura de componentes)El servicio no tiene vista, ni html, ni css – es acción.

Los componentes tienen servicio 🡪 inyecciones de dependencia. (JSON dependencies) .

Cada servicio es único para su función a desempeñar.

El servicio no puede funcionar por si solo; se instancia una vez y luego se “inyecta”.

Curiosidad: (Las const por convenio se escriben en mayúsculas, x ejm: conts PI = 3.141592

* FILTROS: A una variable se le pueden aplicar filtros para que los modifique (podemos buscar la lista de filtros ya que son muchísimos, hay para strings ( uppercase, lowercase) para numer también .. etc…

Mirara en la doc. Oficial de angular

Se pueden anidar o encadenar filtros.

* BANANA IN BOX

Double binding es meter algo entre doble() y luego {} 🡪{()}

Tiene que haber un input para que se puede modificar a posteriori.

Cuando empleamos doble llave {{}} , estamos consumiendo los datos.

-Dentro de la banana in a box insertamos la “directiva” en el componente hmtl(plantilla)

-En component.ts imporatmos el FormModules de @angular/forms

-In app module.ts lo añadimos otra vez e importamos dos veces ( al prinicipio y importamos sub-JSON del NgModule JSON.

\*Cada vez que un año de algo nuevo en app componentes, debo añadirlo también a app.module.(es como el constructor, es el que cargará todo lo que necesito para mi aplicación)

(eventos)

[clases]

{{lectura}}

[(double – binding)]

Directivas : ng

## (eventbinding)

Se recomienda declarar las funciones manejadoras con la nomenclatura on Event

Otras directivas famosas son las asociadas al concepto event-binding y class-binding

<button(click)=”onEvent(myHero)”> (\*onEvent es una función manejadora que creo yo y por la tanto la puedo llamar como quíera, pero se suele llamar así por convenio

## (classBingding)

Declare a class of the component at the component.html file y las define en **component.css.file.** Y aplica la clase dependiendo de la comparación de la condición.

## Directivas:

<div \* ngFor =”let myHero of heroes”> // ngFor coge la variable letmyHero de elementos del array héroes – Y lo pinta :

<div\*ngIf = “myHero.name !== “”>”Sirve para pintar algo si se cumple una condición, xejm si myHero.name no está vacío.

\*\*Todo lo que tenga que ver con eventos se pone (entre paréntesis), y entre las llaves lo que vamos a consumir y se ve a tiempo real.

)

## (Enrouting – menu navegación 🡪 todos los frame Works tienen este concepto)

Tiene que tener un componente general (app) y un lugar donde se cargará el elemento al que estamos llamando.

**<router-oulet></router-oulet>** Es la etiqueta donde se va a meter “ el mando a distancia”

El enrutador es un fichero que linka strings a un component🡪 Es “el mando a distancia”

Comando ng: genera un nuevo .ts y toca el appModule.

Este es el CLI para generarlo:

$ ng generate module app-routing –flat –module=app –para generar un nuevo módulo

-En las path : que es la terminación de la URL – se pondrá lo que diga el de SEO

-A continuación añadimos el componente relacionado con el path.

Ex: {path: ‘contact’ , component: ContactComponenet