alejandrocerezo

Vue.js puede que lo pete en el futuro

Los que lo petan ahora son:

**Angular** backbone ember y **react**

En http://todomvc.com hay unos ejemplos de los frameworks

**M**ongoDB

**E**xpressJS

**A**ngular

**N**odeJS

Da lugar a mean.io

Angular permite la creación de test unitarios y End to end usando Jasmine y Karma

En <http://webcomponents.org> se pueden ver los componentes

En <https://www.polymer-project.org/1.0/> está la librería para utilizar web components

Hay un plugin para Chrome que se llama Augury que añade una nueva pestaña a la herramienta de depuración del navegador

<https://github.com/alce65/Curso_Angular2>

El interfaz gráfico de git es <https://desktop.github.com/>

El comando parar configurar un proxy es:

git config –global http.proxy <http://rlopezh:contraseña>@proxy.indra.es:8080

Visual Studio code como ide de desarrollo

Webpack es el gestor de módulos que se usa para el empaquetado, transpilado…, incluido en Angular-cli

<https://cli.angular.io/>

Configurar el proxy para npm:

npm config set proxy <http://rlopezh:Cipotu3___@proxy.indra.es:8080>

npm config set https-proxy <http://rlopezh:Cipotu3___@proxy.indra.es:8080>

Para la instalación de angular-cli se ejecuta:

npm install –g angular-cli

Para crear el proyecto hay que ejecutar dentro de la carpeta que queramos

**ng new hola-mundo**

Desde dentro de la carpeta del proyecto, al escribir

**ng serve**

levanta un servidor

con localhost:4200 se accede a la aplicación

Unos plugins insteresantes para el visualStudio son:

Path intellisense

Auto Import

TSLint

Angular v4 TypeScript Snippets

Se puede bajar un proyecto de GitHub y luego ejecutar npm install para que leyendo el fichero de configuración se descargue todas las dependencias necesarias.

Ng build guarda el proyecto en la carpeta dist para poder ser distribuido.

En el fichero package.json hay dependencias y dependencias de desarrollo, éstas últimas no se usan cuando se publica el proyecto.

npm install xxx instala el paquete indicado. Con npm install –s lo instala e introduce una dependencia en el fichero package.json, con –sd lo añade como dependencia de desarrollo.

En la parte de arriba del fichero package se pueden crear scripts dentro de su propia sección para realizar las acciones deseadas.

En el fichero main.ts es la cabecera de todo el proceso de generación de código en Angular.

|  |
| --- |
| import { enableProdMode } from '@angular/core'; |
|  | import { platformBrowserDynamic } from '@angular/platform-browser-dynamic'; |
|  |  |
|  | import { AppModule } from './app/app.module'; |
|  | import { environment } from './environments/environment'; |
|  |  |
|  | if (environment.production) { |
|  | enableProdMode(); |
|  | } |
|  |  |
|  | platformBrowserDynamic().bootstrapModule(AppModule) |
|  | .catch(err => console.log(err)); |

**ng bould –t production** crea la carpeta de distribución con lo justo y necesario para que ocupe poco.

La página <https://kangax.github.io/compat-table/es6/> da una tabla con la compatibilidad de los navegadores con las características de js

var data = [{precio: 12}, {precio: 34}, {precio: 19}]

data.forEach( elem => {

if(true){

const iva = 1.16

let precioFinal = elem.precio \* iva

console.log(`el precio final es ${precioFinal}`);

}

})

Este bucle recorre todos los elementos del array. Las comillas usadas en el log son las que están al lado de la P del teclado, eso indica que respeta los espacios, los saltos de línea y las variables que se introduzcan.

const para definir constantes y let para definir variables que no son globales, son locales al ámbito, función, bucle….

<https://www.typescriptlang.org/>

<https://www.gitbook.com/book/basarat/typescript/details>

En la declaración de los componentes, hay un selector que será el nombre que se deberá usar en la pantalla para utilizarlo.

En el fichero .angular-cli.json hay propiedades que se pueden modificar y que se utilizan a la hora de crear elementos del proyecto.

[Sas es lo que más se lleva ahora para los css]

**ng generate** o **ng generate X “nombre”** genera lo que indiquemos con la X

Esto crea las carpetas y los ficheros necesarios para definir el elemento.

ng g c caca crea un componente llamado caca colgando de la carpeta raíz app

ng g c pie crea un módulo pie que utilizaremos para meter un footer al a página principal.

Crea la estructura de carpetas con los ficheros y actualiza el fichero app.module.ts para importarlo y usarlo.

Para utilizar el nuevo componente se mete <app-pie></app-pie> en app.component.html (componente principal)

Creamos un módulo con **ng g m commos** donde meteremos elementos comunes. Crea una carpeta con dos ficheros. Luego tenemos que importarlo desde el módulo principal app.module.ts

import { CommonsModule } from './commons/commons.module';

…

imports: [

BrowserModule,

CommonsModule

],

Creamos un componente cabecera dentro del módulo commons ng g c commons\cabeza

Crea los ficheros dentro de la carpeta commons y actualiza el fichero de módulo de commons

Para utilizar el componente modificamos el fichero app.component.html

<app-cabeza></app-cabeza>

Para que funcione, hay que añadir la exportación en el fichero de la definición del módulo commons:

@NgModule({

imports: [

CommonModule

],

declarations: [CabezaComponent],

exports: [CabezaComponent]

})

En el pie.component.ts ponemos

ngOnInit() {

this.autor = 'Raul López';

this.anno = '2017';

}

Y en pie.component.html ponemos

<p>

{{autor}} - {{anno}}

</p>

Las directivas estructurales modifican el DOM. Son como componentes sin vista.

Para que funciones el doble bindig hay que importar:

import { FormsModule } from '@angular/forms';

import { AppComponent } from './app.component';

@NgModule({

declarations: [

AppComponent

],

imports: [

BrowserModule,

CommonsModule,

FormsModule

],

El código dentro del html sería:

<form>

<label for="nombre">Dime tu nombre</label>

<input type="text" name="nombre" id="nombre" [(ngModel)]="nombre">

<button (click)="nombre=''">Borrar</button>

</form>

<p>Hola {{nombre}}</p>

Formulario con checks, radios, select

**component.ts**

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

interface Provincia {

idProvincia:number,

nombre:string

}

@Component({

selector: 'app-root',

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

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

})

export class AppComponent {

aImpresoras:Array<string> = ["HP Pagewide Pro 477dw",

"Epson Stylus SX235W",

"Lexmark CS310/410",

"Brother HL-3140CW"]

  aProvincias:Array<Provincia> = [

{idProvincia:2, nombre:"Castellón"},

{idProvincia:3, nombre:"Alicante"},

{idProvincia:1, nombre:"Valencia" },

{idProvincia:7, nombre:"Teruel"},

{idProvincia:5, nombre:"Tarragona"}

];

impresoraSeleccionada:string;

provinciaSeleccionada:Provincia = {idProvincia:0, nombre:""};

  print:boolean = false;  //true si se selecciona imprime

  isClaro:boolean = false; //true si se selecciona claro

mostrarFinal:boolean = false;

  avisarPrint = function () {

      console.info("Modificada la opción de imprimir con valor :" + this.print);

  }

avisarProvincia = () => {

this.mostrarFinal=true

console.info("Provincia Seleccionada")

console.dir(this.provinciaSeleccionada)

}

}

**component.html**

<header>

<h1>Formulario</h1>

</header>

<article>

<!--Radio-Buttos-->

<h2>Selecciona una Impresora</h2>

<div \*ngFor="let impresora of aImpresoras">

<input type="radio"

name ="printer" [(ngModel)]="impresoraSeleccionada" [value]="impresora"

(change)="mostrarFinal=true">

<label>{{impresora}}</label>

</div>

<!--Checkbox-->

<h2>Seleción de opciones</h2>

<p>

<input id="print" type="checkbox" [(ngModel)]="print" (change)="mostrarFinal=true">

<label for="print">Imprimir resultado</label><br>

<input id="tono" type="checkbox" [(ngModel)]="isClaro" (change)="mostrarFinal=true">

<!-- ng-true-value=("claro") ng-false-value=("oscuro") -->

<label for="tono">Tono claro</label>

</p>

<!--Select/options-->

<p>

<label for="select">Provincia </label>

<select id="select" [(ngModel)]="provinciaSeleccionada" (ngModelChange)="avisarProvincia()">

<option \*ngFor="let provincia of aProvincias"

[ngValue]="provincia">

{{provincia.nombre}}

</option>

</select>

</p>

<div \*ngIf="mostrarFinal">

<h2>Resultado</h2>

<ul>

<li>Impresora seleccionada: {{impresoraSeleccionada}}</li>

<li>Opción print seleccionada: {{print}}</li>

<li>Opción claro seleccionada: {{isClaro}}</li>

<li>Provincia elegida: {{provinciaSeleccionada.nombre}}</li>

</ul>

</div>

</article>

**Validaciones**

Si se va a validar en angular, hay que decirle al formulario que no haga la validación de html5 poniendo novalidate

**Comunicación entre componentes**

De padre a hijo

Texto alternativo generado por el equipo:
El componente padre puede especificar propiedades en el 
componente hijo como si fuera un elemento nativo HTML 
vista 
<hijo [title]='appTitle'></hij0> 
(padre) 
El valor de title en el hijo corresponderá a la propiedad appTitle 
en el padre 
class controller : 
(hijo) 
vista 
(hijo) 
@lnput() 
private title: string; 
<h l>{{title}}</h 1> 

De hijo a padre

Texto alternativo generado por el equipo:
El componente hijo puede generar eventos que son atendidos 
por el padre como si fuera un elemento nativo HTML 
El padre se suscribe al evento : le asigna una función manejadora. 
La variable $event apunta al evento generado 
vista 
<h ea der id 
(padre) 
@Output() 
evento: new EventEmitter<boolean>(); 
class controller : 
(hijo) 
(en alguna circunstancia) 
th is.evento.em it(parámetros) 

**Filtros**

Los filtros se usan:

variable | filtro

Son siempre filtros de salida

Existe un decorador @Pipe con el que podríamos crear nuestro propio filtro.

**Inyecciones**

Se puede crear la estructura de un servicio ejecutando:

ng g s nombreDelServicio