Angular 2 – Parte 1-AppPelículas

Elementos que conforman Angular 2

COMPONENTES

Un componente al final va a controlar un trozo de pantalla o de la vista.
Todo lo que se puede ver en pantalla es controlado y gestionado por este tipo de elementos.
La lógica de un componente dentro una clase en Angular 2 es que da soporte a una vista interactuando con ella a través de un API con propiedades y métodos.
El componente hace de mediador entre la vista a través de la plantilla y la lógica de la app donde se incluirá el modelo de datos, es decir una especie de controlador.

PLANTILLAS

Las plantillas van a definir la vista de los componentes.
Son htmls y tienen sintaxis especial de Angular. Trabajando con el databinding y las directivas.

DECORADORES Y METADATOS

Con los decoradores (patrón de diseño) vamos a configurar dinámicamente atributos/metadatos de las clases y componentes.
Los metadatos van a describir a las clases pero también describen relaciones, por ejemplo si tenemos un componente y una plantilla el metadato se va a encargar de decirle a Angular que ese componente y esa plantilla van juntos, entre otras muchas cosas.

SERVICIOS

Son clases con un objetivo claro, facilita la reutilización, son un tipo de componente dentro de la arquitectura de Angular 2 y mediante la inyección de dependencias los podemos usar en otros componentes principales digamos.

PROVIDERS

Son servicios que nos proveen de datos o funcionalidades mediante sus métodos. Existen providers/servicios propios de Angular o creados por nosotros mismos.

DIRECTIVAS

Son funcionalidades aplicables al DOM y a los elementos HTML en las plantillas de un componente. Por ejemplo una directiva puede servir para controlar que un div se muestre o no o recorrer un array en la vista (directivas estructurales, estructuras condicionales y de control) o incluso también puede servir para darle una un estilo u otro a un elemento del HTML o también para interactuar con el modelo de datos del componente.
Básicamente son nuevos atributos para aplicarle a cualquier cosa en nuestra plantilla/vista.
Más información sobre la arquitectura de Angular 2 en la documentación oficial:https://angular.io/docs/ts/latest/guide/architecture.html

Instalar Angular 2

NPM

Tener instalada un versión mínima 3 de npm, para ver (`npm -v`), actualizar (`npm install npm -g`).

Instalación/creación proyecto en Angular " Final Release

En www de wamp creamos directorio (mkdir curso-angular), realmente no hace falta tener en marcha WampServer para que funcione ya que el propio compilador de Angular crea su propio servidor, pero por buenas prácticas, o para cuando se necesite conectarse a una base de datos, imágenes o crear un virtual host, siempre es mejor tener los proyectos de angular dentro de WampServer.

Creación de archivos

Para empezar con la instalación, se deben crear una serie de archivos Json que el npm analizará y empezará a instalar las dependencias configuradas en estos archivos, principalmente, descargará de internet la carpeta "typings" y "node_modules".

Crear archivo tsconfig.json: touch tsconfig.json.

Sirve para configurar el compilador de TypeScript

```
{
  "compilerOptions": {
    "target": "es5",
    "module": "commonjs",
    "moduleResolution": "node",
    "sourceMap": true,
    "emitDecoratorMetadata": true,
    "experimentalDecorators": true,
    "removeComments": false,
    "noImplicitAny": false
  }
}
```

Crear el fichero typings.json.

Creamos el archivo typings.json este fichero también es para que TypeScript funcione correctamente:

```
{
  "globalDependencies": {
    "core-js": "registry:dt/core-js#0.0.0+20160725163759",
    "jasmine": "registry:dt/jasmine#2.2.0+20160621224255",
    "node": "registry:dt/node#6.0.0+20160909174046"
  }
}
```

Creamos el fichero package.json

Añadimos el archivo package.json a la raíz del proyecto, que nos sirve para definir las dependencias necesarias del proyecto y que el gestor de paquetes de node las gestione e instale lo necesario, le metemos el siguiente contenido:

```
{
  "name": "angular2-final-victorroblesweb",
  "version": "1.0.0",
  "scripts": {
    "start": "tsc && concurrently \"npm run tsc:w\" \"npm run lite\" ",
    "lite": "lite-server",
    "postinstall": "typings install",
    "tsc": "tsc",
    "tsc:w": "tsc -w",
    "typings": "typings"
  },
  "license": "ISC",
  "dependencies": {
    "@angular/common": "2.0.0",
    "@angular/compiler": "2.0.0",
    "@angular/core": "2.0.0",
    "@angular/forms": "2.0.0",
    "@angular/http": "2.0.0",
    "@angular/platform-browser": "2.0.0",
    "@angular/platform-browser-dynamic": "2.0.0",
    "@angular/router": "3.0.0",
    "@angular/upgrade": "2.0.0",
    "core-js": "^2.4.1",
    "reflect-metadata": "^0.1.3",
    "rxjs": "5.0.0-beta.12",
    "systemjs": "0.19.27",
    "zone.js": "^0.6.23",
    "angular2-in-memory-web-api": "0.0.20",
    "bootstrap": "^3.3.6"
  },
  "devDependencies": {
    "concurrently": "^2.2.0",
    "lite-server": "^2.2.2",
    "typescript": "^2.0.2",
    "typings":"^1.3.2"
  }
}
```

Crear archivo systemjs.config.js

```
**
 * System configuration for Angular 2 samples
 * Adjust as necessary for your application needs.
 */
(function (global) {
  System.config({
    paths: {
      // paths serve as alias
      'npm:': 'node_modules/'
    },
    // map tells the System loader where to look for things
    map: {
      // our app is within the app folder
      app: 'app',
      // angular bundles
      '@angular/core': 'npm:@angular/core/bundles/core.umd.js',
      '@angular/common': 'npm:@angular/common/bundles/common.umd.js',
      '@angular/compiler': 'npm:@angular/compiler/bundles/compiler.umd.js',
      '@angular/platform-browser': 'npm:@angular/platform-
browser/bundles/platform-browser.umd.js',
      '@angular/platform-browser-dynamic': 'npm:@angular/platform-browser-
dynamic/bundles/platform-browser-dynamic.umd.js',
      '@angular/http': 'npm:@angular/http/bundles/http.umd.js',
      '@angular/router': 'npm:@angular/router/bundles/router.umd.js',
      '@angular/forms': 'npm:@angular/forms/bundles/forms.umd.js',
      // other libraries
      'rxjs':                      'npm:rxjs',
      'angular2-in-memory-web-api': 'npm:angular2-in-memory-web-api',
    },
    // packages tells the System loader how to load when no filename and/or
no extension
    packages: {
      app: {
        main: './main.js',
        defaultExtension: 'js'
      },
      rxjs: {
        defaultExtension: 'js'
      },
      'angular2-in-memory-web-api': {
        main: './index.js',
        defaultExtension: 'js'
      }
    }
  });
})(this);
```

Una vez creados los archivos, ejecutamos comando para las instalación:

```
npm install
```

Dentro de App:

Toda App de Angular 2 necesita de este archivo principal, es el que está configurado para cargar de inicio, este es el app.component.ts, se puede usar otro nombre pero por buenas practicas, aprovechar recursos de Github, ejemplos, etc. es mejor usar el mismo nombre.

App.component.ts (todo minúsculas)

```
// Importar el núcleo de Angular
import {Component} from 'angular2/core';

// Decorador component, indicamos en que etiqueta se va a cargar la
plantilla
@Component({
    selector: 'my-app',
    template: '<h1>Hola mundo con Angular 2 !!</h1>'
})

// Clase del componente donde iran los datos y funcionalidades
export class AppComponent { }
```

app.module.ts

Es el controlador de los módulos que necesitará la App.

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

@NgModule({
  imports:      [ BrowserModule ],
  declarations: [ AppComponent ],
  bootstrap:    [ AppComponent ]
})

export class AppModule { }
```

Estamos usando la función decorador @NgModule y le estamos indicando una serie de metadatos.

- **imports**: aquí le pasamos los módulos o componentes internos de angular que es necesario que se carguen en este módulo. Por defecto es necesario cargar BrowserModule.
- **declarations**: le pasamos los componentes y directivas que se van a usar en este modulo. En este caso pues se va a usar por defecto el componente AppComponent.
- **boostrap**: le indicamos el componente principal que se va a cargar en la aplicación cuando se lance. Normalmente siempre será AppComponent.

main.ts

```
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
import { AppModule } from './app.module';
const platform = platformBrowserDynamic();
platform.bootstrapModule(AppModule);
```

index.html en la raíz del proyecto (fuera de app)

```
<!DOCTYPE HTML>
    <html>
      <head>
        <base href="/">
        <title>Angular 2 - Hola mundo!</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1">

        <script src="node_modules/core-js/client/shim.min.js"></script>
        <script src="node_modules/zone.js/dist/zone.js"></script>
        <script src="node_modules/reflect-metadata/Reflect.js"></script>
        <script src="node_modules/systemjs/dist/system.src.js"></script>

       <script src="systemjs.config.js"></script>
       <script>
         System.import('app').catch(function(err){ console.error(err); });
       </script>
      </head>

      <body>
        <my-app>Cargando...</my-app>
      </body>
    </html>
```

Con todos los archivos creados, se lanza el compilador para que cree un "mini-servidor" para ejecutar la apliación con el comando "npm start".

Mostrar propiedades de un componente

En app.component.ts se crea una simple variable en la clase y se asigna en el template. Se le conoce como interpolación

```
// Importar el núcleo de Angular
import {Component} from '@angular/core';

// Decorador component, indicamos en que etiqueta se va a cargar la plantilla
@Component({
    selector: 'my-app',
    template: '<h1>{{titulo}} con Angular 2 !!</h1>'
})

// Clase del componente donde iran los datos y funcionalidades
export class AppComponent {
      public titulo ="HOLA MUNDO";
}
```

```
// Importar el núcleo de Angular
import {Component} from '@angular/core';

// Decorador component, indicamos en que etiqueta se va a cargar la plantilla
@Component({
    selector: 'my-app',
    template: '<h1>{{titulo}}</h1>'
})

// Clase del componente donde iran los datos y funcionalidades
export class AppComponent {
      public titulo ="Películas con Angular 2";
}
```

Se prepara para crear la estructura título, director, año
Se modifica el template: usar comillas de acento abierto para poder hacer saltos de línea.

```
      <h1>{{titulo}}</h1>
                  <ul>
                  <li>Titulo:<strong>{{pelicula}}</strong></li>
                  <li>Director:<strong>{{director}}</strong></li>
                  <li>Año:<strong>{{anio}}</strong></li>
                  </ul>
```

En app.component.ts

```
public titulo ="Películas con Angular 2";
      public pelicula ="Batman v Superman";
      public director ="Zack Snider";
      public anio = "2016";
```

Usar el ctrl para seleccionar más de una línea y añadir que son de tipo string.

```
public titulo:string ="Películas con Angular 2";
      public pelicula:string ="Batman v Superman";
      public director:string ="Zack Snider";
      public anio:number = 2016;
```

Crear Vistas y Modelo de Datos.

Sirve para separar la vista del componente.
Dentro de app se crea carpeta "view" y el peliculas.html copiando el html del template anterior.
En vez de template, se añade templateUrl

```
@Component({
    selector: 'my-app',
    templateUrl: 'app/view/peliculas.html'
})
```

Buenas prácticas: Usar el método constructor para declaración de variables y llamada a métodos de inicio.

```
public titulo:string;
	public titulo:string;
	public pelicula:string;
	public director:string;
	public anio:number;

	//Inicializar las variables de la clase.
	constructor (){
		this.pelicula ="Batman v Superman";
		this.titulo ="Películas con Angular 2";
		this.director="Zack Snider"
		this.anio = 2016;
	}
```

Buenas prácticas: crear modelo de datos
Dentro de App, carpeta Model y archivo película.ts

```
export class Pelicula{
	//se definen dentro del paréntesi porque son variables que recibirá al
crear el objeto en el app.component
	constructor (
		public id: number,
		public titulo: string,
		public director: string,
		public anio:number

		){}
}
```

En app.component.ts

```
import {Pelicula} from "./model/pelicula";
export class AppComponent {
	public titulo:string = "Películas con Angular 2";
	public pelicula: Pelicula;

	//Inicializar las variables de la clase.
	constructor (){
		this.pelicula = new Pelicula(1, "Batman v Superman","Zack
Snider", 2016)
	}
}
```

Falla porque en el template hay que re-asignar las variables como objeto
{{pelicula.titulo}}, {{pelicula.director}}, {{pelicula.anio}}

Se añade un debug para mostrar por consola el objeto.
En el constructor:

```
this.debug();
```

en la clase

```
debug(){
            console.log(this.pelicula);
      }
```

Añadir hojas de estilos al component/hojas de estilo independientes

Cuando queremos que carguen css solo para ese componente, si queremos estilos globales se declaran en el index.html.

Se crear carpeta assets→css y archivo styles.css
En el @Component:

```
@Component({
    selector: 'my-app',
    templateUrl: 'app/view/peliculas.html',
    styleUrls:["../assets/css/styles.css"]
})
```

Styles.css

```
@import url('https://fonts.googleapis.com/css?family=Dosis');
h1 {
      font-family: 'Dosis', sans-serif;
      color:#444;
}
```

Directivas NgIf, NgClass y evento Click

Se usan en los templates junto con el html, son funciones específicas de Angular2 que nos dan la funcionalidad de programar en el html (leguaje de plantilla).

NgIf

Ejemplo de mostrar contenido si una variable esta *true*.

En la clase AppComponent:

```
public mostrarDatos: boolean;
```

En el constructor:

```
this.mostrarDatos = true;
```

En el template película.html

```
<ul *ngIf="mostrarDatos===true">
```

Hay que mirar que si no se cumple no se genera ni el html.

Evento Click

Declaramos botón en la plantilla

```
<button (click)="onShowHide()">Mostrar datos</button>
```

En component, el contructor del mostrarDatos es *false*.

Se crea el método en component

```
onShowHide(){
            this.mostrarDatos=true;
      }
```

NgClass.Añadir un atributo de clase cuando cumpla una condición del modelo.

Se le añade al button atributo mara que se le inserte al html una clase CSS en el caso de estar true.

<button (click)="onShowHide()" [class.hideData]="mostrarDatos===true">Mostrar datos</button>

Se añade regla css en styles.css

```
.hideData{
	display: none;
}
```

Intercambiar boton mostrar/ocultar

Por defecto mostrarDatos es false (contructor)

En app.component.ts

```
onShowHide(value){
		this.mostrarDatos=value;
	}
```

En el template

```
<button (click)="onShowHide(true)"
[class.hideData]="mostrarDatos===true">Mostrar datos</button>
<button (click)="onShowHide(false)"
[class.hideData]="mostrarDatos===false">Ocultar datos</button>
```

TwoWayDataBinding a través del (NgModel)

Cuando se muestra algo con las dobles llaves, es un tipo de dataBinding del component al template. El two way es cambiar el valor del modelo y del template, es bidireccional.

Cambiar Titulo: <input type="text" [(ngModel)]="pelicula.titulo" />

Mostrar contenido por consola, se modifica la función debug

```
debug(titulo=null){
		if (titulo!= null){
			console.log(this.pelicula.titulo);
		} else{
			console.log(this.pelicula);
		}

	}
```

La plantilla

```
<p>Cambiar Titulo: <input type="text" (keypress)="debug(true)"
[(ngModel)]="pelicula.titulo" /></p>
```

Directiva NgFor y Arrays

Se va a crear un listado bajo el detalle de película.

En app.component.ts

```
public peliculas:Array<Pelicula>; //
```

constructor

```
this.peliculas =[
                    new Pelicula(1, "Batman v Superman","Zack Snider", 2016),
                    new Pelicula(2, "Pelicula 2","Director 2", 2016),
                    new Pelicula(3, "Pelicula 3","Director 3", 2016),
                    new Pelicula(4, "Pelicula 4","Director 4", 2016),
                    new Pelicula(5, "Pelicula 5","Director 5", 2016)
             ]
```

En el template:

```
<h1>{{titulo}}</h1>
<p>Película favorita</p>
<button (click)="onShowHide(true)"
[class.hideData]="mostrarDatos===true">Mostrar datos</button>
<button (click)="onShowHide(false)"
[class.hideData]="mostrarDatos===false">Ocultar datos</button>
<ul *ngIf="mostrarDatos===true">
      <li>Titulo:<strong>{{pelicula.titulo}}</strong></li>
      <li>Director:<strong>{{pelicula.director}}</strong></li>
      <li>Año:<strong>{{pelicula.anio}}</strong></li>
</ul>
<p>Cambiar Titulo: <input type="text" (keypress)="debug(true)"
[(ngModel)]="pelicula.titulo" /></p>
<hr/>
<p>Listado de Películas</p>
<ul *ngFor="let pelicula of peliculas" style="margin-bottom: 20px">
      <li>Titulo:<strong>{{pelicula.titulo}}</strong></li>
      <li>Director:<strong>{{pelicula.director}}</strong></li>
      <li>Año:<strong>{{pelicula.anio}}</strong></li>
</ul>
```

Y si no se recibe algún dato?: Añadir condición para mostrar o no el dato.

```
//isset para controlar si existe
      <li *ngIf="pelicula.anio">Año:<strong>{{pelicula.anio}}</strong></li>
```

Se edita como *null* un año del array para ver que funciona.

Múltiples componentes, evento click y two-way-data-binding

Nuevo archivo en *view:* películas-list.html

```
<p>Peliculas list component</p>
```

En carpeta components, se crear peliculas-list.component.ts

```
// Importar el núcleo de Angular
import {Component} from '@angular/core';
//Importar el modelo pelicula
import {Pelicula} from "../model/pelicula"; //Cambia la ruta!!
// Decorador component, indicamos en que etiqueta se va a cargar la plantilla
@Component({
    selector: 'peliculas-list',
    templateUrl: 'app/view/peliculas-list.html'
})

// Clase del componente donde iran los datos y funcionalidades
export class PeliculasListComponent {

}
```

En app.component.ts

```
@Component({
    selector: 'my-app',
    templateUrl: 'app/view/peliculas.html',
    directives:[PeliculasListComponent],
    styleUrls:["../assets/css/styles.css"]
})
```

En el template películas.html

```
<peliculas-list></peliculas-list>
```

Refactor

A partir de aquí y ver que se muestra por pantalla se elimina de app.component.ts el código del AppComponent que se define ahora en películas-list.component

```
// Clase del componente donde iran los datos y funcionalidades
export class PeliculasListComponent {
	public pelicula: Pelicula;
	public mostrarDatos: boolean;
	public peliculas:Array<any>;

	//Inicializar las variables de la clase.
	constructor (){
		this.pelicula = new Pelicula(1, "Batman v Superman","Zack
Snider", 2016);
		this.debug();
		this.mostrarDatos = false;
		this.peliculas =[
			new Pelicula(1, "Batman v Superman","Zack Snider", 2016),
			new Pelicula(2, "Pelicula 2","Director 2", 2016),
			new Pelicula(3, "Pelicula 3","Director 3", 2016),
			new Pelicula(4, "Pelicula 4","Director 4", 2016),
			new Pelicula(5, "Pelicula 5","Director 5", null)
		]
	}
	debug(titulo=null){//Parametro opcional, si lo recibe o no
		if (titulo!= null){
			console.log(this.pelicula.titulo);
		} else{
			console.log(this.pelicula);
		}

	}
	onShowHide(value){
		this.mostrarDatos=value;
	}
}
```

Eliminar en app.component.ts

```
import {Pelicula} from "./model/pelicula";
```

Se elimina la inserción del CSS en el app.component y se añade como global en el index.html

```
<link rel="stylesheet" type="text/css" href="assets/css/styles.css"/>
```

Nuevo componente footer

Nuevo archivo en components: películas-footer.component.ts, se coje de plantilla inicial la de app component y se deja

```
// Importar el núcleo de Angular
import {Component} from '@angular/core';

// Decorador component, indicamos en que etiqueta se va a cargar la plantilla
@Component({
    selector: 'peliculas-footer',
    templateUrl: 'app/view/peliculas-footer.html'
})

// Clase del componente donde iran los datos y funcionalidades
export class PeliculasFooterComponent {
}
```

En app.component se añade

```
import {PeliculasFooterComponent} from "./components/peliculas-
footer.component";
```

Se crear películas-footer.html

```
<hr/>
Copyright 2016.
```

Se llama en películas.html

```
<peliculas-footer></peliculas-footer>
```

Evento Click

Conseguir que al hacer click en el título de la lista se actualice la película favorita

En películas-list.component.ts se añade otro método

```
onCambiaPelicula(pelicula){
            this.pelicula=pelicula;
      }
```

En el template películas-list.html

```
<li
(click)="onCambiaPelicula(pelicula)">Titulo:<strong>{{pelicula.titulo}}</stro
ng>
```

**Corregir declaración inicial de variable de la película en con this.pelicula = this.peliculas[0]; después de la declaración del array ya que no hace el binding al declararla como nuevo objeto.

Añadir una class en caso de ser la elegida

Primero crear el puntero incial en list-component

```
public peliculaElegida: Pelicula;
```

En el constructor:

```
this.peliculaElegida = this.peliculas[0];
```

Modificar método

```
onCambiaPelicula(pelicula){
            this.pelicula=pelicula;
            this.peliculaElegida=pelicula;
      }
```

En el template

```
<li (click)="onCambiaPelicula(pelicula)" class="pelicula-title"
[class.clicked]="peliculaElegida.id===pelicula.id">Titulo:<strong>{{pelicula.
titulo}}</strong></li>
```

En el styles.css se añade

```
.pelicula-title{
      cursor: pointer;
}
.pelicula-title:hover{
      text-decoration: underline;
}
.clicked{
      color:red;
      font-size: 20px;
}
```

Crear y usar un servicio y como se usa en un componente.

Los servicios son clases con un objetivo claro, facilitan la reutilización. Mediante la inyección de dependencias los podemos utilizar en componentes principales. Normalmente un servicio nos va a servir para proveedor de datos, guardar datos en un modelo, peticiones http (Ajax), etc.
En app crear carpeta *services* y archivo películas.service.ts

```
import {Injectable} from "@angular/core"; //Objeto inyectable de Angular

//Decorador
@Injectable()

export class PeliculasService{
	getPeliculas(){
		return “Hola mundo desde un servicio”;
	}
}
```

En películas-list.component

```
//importar servicio
import {PeliculasService} from "../services/peliculas.service";
```

en el @Component

```
@Component({
    selector: 'peliculas-list',
    templateUrl: 'app/view/peliculas-list.html',
    providers:[PeliculasService]
})
```

En el constructor

```
constructor (private _peliculasService:PeliculasService){
		//this.debug();
		this.mostrarDatos = false;

		this.peliculas =this.peliculas =[
			new Pelicula(1, "Batman v Superman","Zack Snider", 2016),
			new Pelicula(2, "Pelicula 2","Director 2", 2016),
			new Pelicula(3, "Pelicula 3","Director 3", 2016),
			new Pelicula(4, "Pelicula 4","Director 4", 2016),
			new Pelicula(5, "Pelicula 5","Director 5", null)
		];
		this.peliculaElegida =this.peliculas[0];
		this.pelicula = this.peliculas[0];

	}
```

Utilizar el servicio y llamar al método

```
public datoServicio;
```

Y en el constructor

```
this.datoServicio = this._peliculasService.getPeliculas();
```

Y en el template

```
{{datoServicio}}
```

Crear el array de las películas en el servicio

Crear archivo mok.peliculas.ts en services

```
import {Pelicula} from "../model/pelicula";

export const PELICULAS:Pelicula[] = [
                    new Pelicula(1, "Batman v Superman","Zack Snider", 2016),
                    new Pelicula(2, "Pelicula 2","Director 2", 2016),
                    new Pelicula(3, "Pelicula 3","Director 3", 2016),
                    new Pelicula(4, "Pelicula 4","Director 4", 2016),
                    new Pelicula(5, "Pelicula 5","Director 5", null)
];
```

En películas.service.ts

```
import {Injectable} from "angular2/core";
import {PELICULAS} from "./mock.peliculas"
//Decorador
@Injectable()

export class PeliculasService{
      getPeliculas(){
            return PELICULAS;
      }
}
```

Hacer console.log de películas para ver el objeto en el constructor

```
console.log(this.peliculas);
```

Añadir un par de películas más al array para ver que se actualiza todo.

Routing Básico-Navegación en Angular

Desde la última versión de angular (RC7/final) se introduce un nuevo archivo para configurar la rutas (páginas) de la App y poder usar el método "route" para enlazar contenidos, paso de variables, etc.
Además de los módulos propios del rounting (routes y RouterModule), se tienen que declarar/importar todos los componentes de la App.
La constante appRoutes es la que declara las rutas de la App.

Crear archivo app.routing.ts

Este códido muestra la configuración de las rutas con todos los componentes de la App, así que primero, se crean los componentes en blanco: películas-footer.component, crear-pelicula.component y peliculas- contacto.component

```
import {ModuleWithProviders} from "@angular/core";
import {Routes, RouterModule} from "@angular/router";

import {PeliculasListComponent} from "./components/peliculas-list.component";
import {PeliculasFooterComponent} from "./components/peliculas-
footer.component";
import {ContactoComponent} from "./components/películas-contacto.component";
import {CrearPeliculaComponent} from "./components/crear-pelicula.component";

const appRoutes: Routes=[
{
      path:'',
      redirectTo:'/',
      pathMatch:'full'
},
{path: "",  component: PeliculasListComponent},
{path: "contacto", component: ContactoComponent},
{path: "crear-pelicula", component: CrearPeliculaComponent},
{path: "crear-pelicula/:titulo/:director/:anio",  component:
CrearPeliculaComponent}
];

export const appRoutingProviders:any[]=[];
export const routing: ModuleWithProviders= RouterModule.forRoot(appRoutes);
```

En películas.html ya no hace falta declarar el tag películas-list o cualquier otro para el contenido dinámico de cada página, se cambia por el tag <router-outlet></router-outlet> que ya de forma automática se encargará de cargar cada página creando el "selector" declarado en cada decorador @Component de cada módulo.
Se declara una navegación básica

```
<h1>{{titulo}}</h1>
<nav>
      <ul>
            <li><a [routerLink]="['']">Inicio</a></li>
            <li><a [routerLink]="['crear-pelicula']">Añadir
pelicula</a></li>
            <li><a [routerLink]="['contacto']">Contacto</a></li>
      </ul>
</nav>
<router-outlet></router-outlet>
<peliculas-footer></peliculas-footer>
```

Nuevos Componentes

Nuevo componente contacto.ts

```
// Importar el núcleo de Angular
import {Component} from '@angular/core';

// Decorador component, indicamos en que etiqueta se va a cargar la plantilla
@Component({
    selector: 'contacto',
    templateUrl: 'app/view/contacto.html',

})


// Clase del componente donde iran los datos y funcionalidades
export class ContactoComponent {
      public titulo:string = "Contacto";

}
```

Contacto.html

```
<h2>{{titulo}}</h2>
<p>Para contactar con nosotros envía e-mail a info@peliculas.sss o llame al
333 99 99 09 de lunes a viernes de 9:00 a 17:00. <p/>
```

Crear nuevas películas

Crear componente y vista

Crear-pelicula.component.ts

```
// Importar el núcleo de Angular
import {Component} from 'angular2/core';



// Decorador component, indicamos en que etiqueta se va a cargar la plantilla
@Component({
    selector: 'crear-pelicula',
    templateUrl: 'app/view/crear-pelicula.html',

})


// Clase del componente donde iran los datos y funcionalidades
export class CrearPeliculaComponent {


}
```

Crear-pelicula.html

```
<h2>Crear pelicula</h2>
```

Añadir elementos de formulario en el template

```
<h2>Crear pelicula</h2>
<form #miFormulario="ngForm" (ngSubmit)="onSubmit()">
	<p>
		Titulo:
		<input type="text" #titulo="ngModel" name="titulo"
[(ngModel)]="nuevaPelicula.titulo" required minlength="5" />
		<span *ngIf="!titulo.valid">No valido</span>
	</p>
	<p>
		Director:
		<input type="text"  #director="ngModel" name="director"
[(ngModel)]="nuevaPelicula.director" required pattern="[A-Za-z]*" />
		<span *ngIf="!director.valid">No valido</span>
	</p>
	<p>
		Año:
		<input type="text" #anio="ngModel" name="anio"
[(ngModel)]="nuevaPelicula.anio" required pattern="[0-9]{4}" />
		<span *ngIf="!anio.valid">No valido</span>
	</p>
	<input type="submit" value="Crear pelicula"
[disabled]="!miFormulario.form.valid"  />
	</form>
```

Crear objeto con los datos del formulario

Crear la clase, hay que importar el modelo de película y el router.

```
// Importar el núcleo de Angular
import {Component, OnInit} from '@angular/core';
import {Router, ActivatedRoute, Params} from '@angular/router';
import {Pelicula} from "../model/pelicula";
//importar servicios
import {PeliculasService} from "../services/peliculas.service";


// Decorador component, indicamos en que etiqueta se va a cargar la plantilla
@Component({
    selector: 'crear-pelicula',
    templateUrl: 'app/view/crear-pelicula.html',
    providers: [PeliculasService]
})

// Clase del componente donde iran los datos y funcionalidades
export class CrearPeliculaComponent implements OnInit {
	public TituloPelicula:string="";
	public nuevaPelicula:Pelicula;

	constructor(private _peliculasService: PeliculasService,
		private _route:ActivatedRoute,
		private _router:Router,
		){

	}
	onSubmit(){

		this._peliculasService.insertPelicula(this.nuevaPelicula);
		this._router.navigate([""]);
	}
```

```
	ngOnInit():any{
		this._route.params.forEach((params: Params) =>{
			this.TituloPelicula=params["titulo"];
		this.nuevaPelicula = new Pelicula(
			0,
			params["titulo"],
			params["director"],
			parseInt(params["anio"]),
			);
		} );

	}
}
```

Bug: Al añadir varias películas se puede observar que se seleccionan varias en el clicked ya que comparten el mismo ID, cambiar por el título.

Validación de campos

Se añade un span para mostrar "no valido" cuando no cumpla.

```
<h2>Crear pelicula</h2>
<form #miFormulario="ngForm" (ngSubmit)="onSubmit()">
	<p>
		Titulo:
		<input type="text" #titulo="ngForm" ngControl="titulo"
[(ngModel)]="nuevaPelicula.titulo" required/>
		<span *ngIf="!titulo.valid">No valido</span>
	</p>
	<p>
		Director:
		<input type="text"  #director="ngForm" ngControl="director"
[(ngModel)]="nuevaPelicula.director" required />
		<span *ngIf="!director.valid">No valido</span>
	</p>
	<p>
		Año:
		<input type="text" #anio="ngForm" ngControl="anio"
[(ngModel)]="nuevaPelicula.anio" required/>
		<span *ngIf="!anio.valid">No valido</span>
	</p>
	<input type="submit" value="Crear pelicula"  />
	</form>
```

Se comprueba

Se añade condición disabled cuando el formulario no sea válido

```
<input type="submit" value="Crear pelicula"
[disabled]="!miFormulario.form.valid"  />
```

Se valida el título con un mínimo de 5 caracteres.
Minlenght="5";
EL director sin números
Pattern="[A-Za-z]*"
Anio 4 numeros
pattern="[0-9]{4}"