componentes

Árbol de componentes Tendrás un árbol de componentes que forman tu aplicación y cada persona lo podrá organizar de su manera preferida. Siempre existirá un componente padre y a partir de ahí podrán colgar todas las ramas que sean necesarias para crear tu aplicación.

En nuestro árbol, como posible organización, podemos tener en un primer nivel los bloques principales de la pantalla de nuestra aplicación.

• Una barra de herramientas, con interfaces para acciones principales (lo que podría ser una barra de navegación, menús, botonera, etc.).

• Una parte principal, donde se desplegarán las diferentes "pantallas" de la aplicación.

• Un área de logueo de usuarios.

Árbol de componentes Luego, cada uno de los componentes principales se podrá subdividir, si se desea, en nuevos árboles de componentes.

• En la barra de herramientas principal podríamos tener un componente por cada herramienta.

• En el área principal podríamos tener un componente para cada "pantalla" de la aplicación o "vista".

• A su vez, dentro de cada "vista" o "pantalla" podíamos tener otra serie de componentes que implementen diversas funcionalidades.

Componentes vs Directivas

Los componentes son **piezas de negocio**, mientras que las directivas se suelen usar para presentación y **problemas estructurales.**

Podemos pensar en un componente como un contenedor donde solucionas una necesidad de tu aplicación.

Tipos de directivas

Existen tres tipos de directivas:

• **Componentes**: Un componente es una directiva con un template. Habrá muchas en tu aplicación y resuelven necesidades del negocio.

• **Directivas de atributos**: Cambian la apariencia o comportamiento de un elemento. Por ejemplo tenemos **ngClass**, que nos permite colocar una o más clases de CSS (atributo class) en un elemento.

• **Directivas estructurales**: Son las que realizan cambios en el DOM del documento, añadiendo, manipulando o quitando elementos. Por ejemplo ngFor, que nos sirve para hacer una repetición (similar al ngRepeat de Angular 1.x), o ngIf que añade o remueve elementos del DOM con respecto a una expresión condicional.

Decorador

¿Qué es un decorador?

Básicamente es una implementación de un patrón de diseño de software que en sí sirve para extender una función mediante otra función, pero sin tocar aquella original, que se está extendiendo. El decorador recibe una función como argumento (aquella que se quiere decorar) y devuelve esa función con alguna funcionalidad adicional.

Las funciones decoradoras comienzan por una "@" y a continuación tienen un nombre. Ese nombre es el de aquello que queramos decorar, que ya tiene que existir previamente. Podríamos decorar una función, una propiedad de una clase, una clase, etc.

¿Qué información se agrega con el decorador? **@Component({**

moduleId: module.id, selector: 'test-angular2-app', templateUrl: 'test-angular2.component.html', styleUrls: ['test-angular2.component.css'] })

¿Qué información se agrega con el decorador?

• **moduleId**: No vamos a ver todavía esta propiedad, es algo que tiene que ver con CommonJS y sirve para poder resolver Urls relativas.

• **selector**: este es el nombre de la etiqueta nueva que crearemos cuando se procese el componente. Es la etiqueta que usarás cuando quieras colocar el componente en cualquier lugar del HTML.

• **templateUrl**: es el nombre del archivo .html con el contenido del componente, en otras palabras, el que tiene el código de la vista.

• **styleUrls**: es un array con todas las hojas de estilos CSS que deben procesarse como estilo local para este componente. Como ves, podríamos tener una única declaración de estilos, o varias si lo

consideramos necesario.

Crear un nuevo Componente

Ng generate Para crear un nuevo componente debemos ejecutar:

ng generate component nombre_componente

Ng generate Si observas ahora la carpeta "src/app" encontrarás que se ha creado un directorio nuevo con el mismo nombre del componente que acabamos de crear.

Utilizar componente creado En el lugar de la aplicación donde lo vayas a usar el componente, tienes que escribir el HTML necesario para que se muestre. El HTML no es más que la etiqueta del componente,

que se ha definido en la función decoradora, atributo "selector"

Utilizar componente creado Si queremos utilizar dicho componente debemos agregar en el HTML que queramos la etiqueta
<app-login></app-login>

Utilizar componente creado En el componente desde donde pretendas usar ese otro componente necesitas importar su código.

Como veras debes indicar la carpeta en la cual se encuentra tu componente (login) y

el nombre del componente (login.component)

Utilizar componente creado En el componente desde donde pretendas usar ese otro componente necesitas importar su código.

Como veras debes indicar la carpeta en la cual se encuentra tu componente (login) y el nombre del componente (login.component)

Ejercicio 1 Crear el componente **<app-home></app-home>**

Este componente deberá tener la home del sitio que queramos desarrollar, por ejemplo un ecommerce.

Nota: No incluir en el componente home el head, header y footer. Estos los vamos a incluir directamente en el archivo index.html

ngClass

Directivas Nos permite alterar las clases CSS que tienen los elementos de la página.

Las directivas se usarán para solucionar necesidades específicas de manipulación del DOM y otros temas estructurales.

[class] Lo primero es decir que la directiva ngClass no es necesaria en todos los casos. Las cosas más simples ni siquiera la necesitan. El atributo "class" de las etiquetas si lo pones entre corchetes funciona como propiedad a la que le puedes asignar algo que tengas en tu modelo

ngClass La directiva ngClass es necesaria para, de una manera cómoda asignar cualquier clase CSS entre un grupo de posibilidades.

A esta directiva le indicamos como valor: 1. Un array con la lista de clases a aplicar.

Ese array lo podemos
especificar de manera literal en el HTML.

Definición del array con javascript:
Vista (html)

Controlador (ts)

ngClass 1. Un objeto con propiedades y valores (lo que sería un literal de
objeto Javascript). Cada nombre de propiedad es una posible clase CSS que se podría asignar al elemento y cada valor es una expresión que se evaluará condicionalmente para aplicar o no esa clase.*Vista (html)*

Controlador (ts)

ngFor

ngFor La directiva ngFor, capaz de hacer una repetición de elementos dentro de la página. Esta repetición nos permite recorrer una estructura de array y para cada uno de sus elementos replicar una cantidad de elementos en el DOM.

Vista (html)

Controlador (ts)

ngFor con objetos *Vista (html)*

Controlador (ts)

nglf

nglf Si la condición se cumple, su elemento se inserta en el DOM, en caso contrario, se elimina del DOM.

Vista (html)

Controlador (ts)

Ejemplos Todo lo visto anteriormente se podrá ver en el modelo de practica

Ejemplo_directivas

Ejercicio 2 Continuando con el ejercicio 1 utilizar la directiva **ngFor** para mostrar los productos cargados en un mock up de datos.

Utilizar el resto de las directivas vistas para armar una home atractiva
del ecommerce.

Servicios

Definición Los Componentes son

grandes consumidores de servicios. No recuperan datos del servidor, ni validan inputs de usuario, ni logean nada directamente en consola. Delegan todo este tipo de tareas a los Servicios.

Los servicios deberían contener/hacer algo muy específico. Por
ejemplo, serían susceptibles de encapsular en un servicio:

- Servicio de logging
- Servicio de datos
- Bus de mensajes
- Cálculo de Impuestos
- Configuración de la app

Crear un servicio

Un servicio es solamente una clase, para crearlo debemos

crear el

archivo correspondiente con una clase dentro. Por ejemplo:

En este caso lo creamos en el directorio **services**

Utilizar un servicio desde un componente

Primero debemos importar la clase en la cual esta definido el **servicio**

Utilizar un servicio desde un

componente

Luego debemos incluir dicha clase en el constructor

En el ejemplo vemos que en el constructor vemos que se incluye la

clase **public _peppapigServices: PeppaPigService**

Ejercicio 3

Aplicar la utilización de servicios para la

home desarrollada en el
punto anterior.

Hacer un mock up con un array de objetos (json) en el cual cada json
es un producto a visualizar en la home.

Formularios

Validación de formularios

Para validar formularios utilizaremos **formBuilder** Primero debemos importar el componente **ReactiveFormsModule** en

app.module.ts e incluirlo en **imports:**

Validación de formularios

Luego debemos incluir **FormBuilder**, **Validators** dentro del componente en el cual queramos realizar la validación de formularios:

Vamos a declarar la variable **loginForm,** la cual contendrá la lógica de validación del formulario. Por otro lado en el constructor debemos recibir el **FormBuilder** como parámetro.

Validación de formularios

En la vista debemos declarar **[formGroup]** en el form que queramos

validar:

En cada elemento del formulario a validar, en lugar de utilizar un

ngModel, debemos utilizar **formControlName.** Le debemos asignar el mismo nombre que le asignamos en el controlador.

Clase validators

Nos provee las siguientes validaciones:

• **Validators.required** = Comprueba que el campo sea llenado.

• **Validators.minLength** = Comprueba que el campo cumpla con un mínimo de caracteres.

• **Validators.maxLength** = Comprueba que

el campo cumpla con un máximo de caracteres.

• **Validators.pattern** = Comprueba que el campo cumpla con un patrón usando una expresión regular.

• **Validators.email** = Comprueba que el campo cumpla con un patrón de correo válido.

Clase validators

• **loginForm.get('nombre').errors**: Si el campo nombre del formulario myForm tiene algún error esta condición será true.

• **loginForm.get('nombre').dirty**: Devuelve true si el usuario ha interactuado con el campo en cuestión.

• **loginForm.get('nombre').hasError('requir**

ed'): Devuelve true si el campo nombre arroja un error de tipo required sobre el campo nombre.

• **loginForm.invalid**: Devuelve true si el campo nombre arroja un error de tipo required sobre el campo nombre.

Ejercicio 4

Al final de la pagina home desarrollara agregar un formulario de

registros de usuarios que tenga los siguientes campos:

• Nombre (*)

• Apellido (*)

• Teléfono

• Email (*)

• Password (*): Debe tener entre 6 y 10 caracteres

(*) Campos obligatorios