Table of Contents

Introducción a AngularJS.

Dentro de la innumerable cantidad de frameworks que ofrece Javascript contamos con AngularJS, una tecnología que cuenta con el patrón MVC (Modelo Vista Controlador) en sus inicios y que ha evolucionado y usa un patrón MVW (Modelo Vista Lo-que-sea ["Whatever"]) orientado a SPA (Single Page Applications), donde los componentes principales de la aplicación son el Modelo y la Vista (pueden o no usarse controladores, rutas, servicios, directivas personalizadas, filtros, etc).

Este es aplicado en el Front-End de nuestra aplicación web y permite lograr una interacción muy simple con nuestros componentes y el manejo del DOM (Document Object Model).

1. Primeros Pasos:

1.1 Descargar AngularJS :

Descargar AngularJS desde el siguiente Link https://angularjs.org/ y descargar Angular JS 1.5.x en su versión minima

2 Cómo usar Angular JS en nuestra aplicación web

2.1 Directivas nativas:

Una directiva es un atributo personalizado que fue previamente desarrollado en Angular, es decir una funcionalidad que implementaremos en nuestra web, existen muchas directivas y nos permiten desarrollar nuestra web SPA, dentro de las más importantes se encuentran la ng-app, ng-controller, ng-include, ng-repeat, ng-show, ng-if, ng-class, ng-form, ng-model, entre muchas otras, de modo que ahora podremos definir el comportamiento de nuestros elementos dentro del layout mediante las diferentes directivas de Angular.

Definiciones

- `ng-app` : Directiva para indicar el modulo principal de la aplicación en AngularJS.

```
<html ng-app="app">
```

- `ng-init` : Directiva para inicializar una variable desde el HTML.

```
<div ng-init= " saludo='HolaMundo'"   > {{saludo}}
</div>
```

- `ng-controller` : Permite crear un controlador que funcionará dentro de la aplicación.

```
<body ng-controller="AppController">
```

- `ng-class` : Permite reemplazar el atributo `class` de un elemento de forma programática.

```
<p ng-class= "active: true" ></p>
```

- `ng-submit` : Funciona bajo el evento submit dentro de un formulario, generalmente invoca una función.

```
<body ng-app="app" ng-controller="AppController" >
<form  ng-submit="miFuncion()" >
  <input type="text">
  <input type="submit">
</form>
<p> {{texto}} </p>
<script>
  var app = angular.module('app', []);
  app.controller('AppController', function($scope){
    $scope.texto = 'Aún no presionas el submit';
    $scope.miFuncion = function () {
      $scope.texto = 'Presionaste el submit!';
    };
  });
</script>
</body>
```

- `ng-repeat` : Genera una iteración, generalmente se usa para recorrer una lista, similar a `ForEach` .

```
<ul ng-init "nombres=['John Snow', 'Ash Ketchum', 'Gary Oak']" >
  <li ng-repeat="name in nombres">
    {{name}}
  </li>
</ul>
```

- `ng-show` : Permite mostrar u ocultar un elemento, recibe un booleano.

```
<div ng-show="true">Hola Mundo</div>
```

- `ng-if` : Similar a ng-show, oculta o muestra elemento pero los falsos no los muestra en el HTML.

```
<div ng-if="true"> Hola Mundo</div>
```

- `ng-include` : Nos permite agregar un template de HTML, este recibe la ruta del HTML.

```
<div ng-include="vista.html"></div>
```

3 $scope, Controlador y Expresión

3.1 $scope y Expresión:

Scope, o `$scope` es un objeto que guarda una referencia con el modelo en la aplicación. Es un contexto de ejecución para expresiones. Los Scopes pueden organizarse de forma jerárquica imitando la estructura del DOM de la aplicación. Además los Scopes pueden observar expresiones (`$watch`) y propagar eventos (`$apply`).

Los Scopes proveen el contexto en el cual las expresiones son evaluadas. Por ejemplo `{{username}}` no tiene ningún significado, a no ser que sea evaluada dentro de un Scope específico que define la propiedad `username` .

Scope es el "pegamento" entre la Vista y el Controlador. El controlador posee una referencia al Scope, esto aisla al Controlador del DOM haciendolo independiente a las Vistas.

Además el controlador puede añadir comportamientos al Scope. Los métodos definidos a través del Scope pueden ser accedidos desde la vista y asociados a eventos del DOM (onClick, onSubmit, etc.)

El siguiente controlador expone a través de `$scope` la propiedad `username` y el método `sayHello()` :

```
angular.module('scopeExample', [])
.controller('MyController', ['$scope', function($scope) {
  $scope.username = 'World';

  $scope.sayHello = function() {
    $scope.greeting = 'Hello ' + $scope.username + '!';
  };
}]);
```

Fuente:

https://docs.angularjs.org/guide/scope

Ejemplo:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF8">
  <title>Ejemplo Angular</title>
  <link rel="stylesheet" href="">
  <script src="lib/js/angular.min.js"></script>
  <script >
    (function(){
      var app= angular.module("app",[]);
      app.controller("applicationController", ['$scopen',function($scope){
        // ejemplo es una propiedad
        $scope.ejemplo='Esto es un ejemplo de Angular JS';
        // hola() es un método
        $scope.hola = function(){
          $scope.ejemplo = 'hola';
        };
      }]);
    })();
    </script>
  </head>

  <body ng-app="app" ng-controller="applicationController">
    {{ejemplo}}
    <button ng-click="hola()">Hola!<button>
  </body>
</html>
```

https://jsfiddle.net/oqkeu3zs/

4 Controladores

Si bien ya hemos hablado un poco de los controladores que podemos crear en angular, ahora nos focalizamos en nuestra estructura.

Cuando nuestro sistema empieza a crecer, nos vemos en la obligación de separar tareas que en un principio nuestro controlador único hacía. Ahora estas serán delegadas a controladores más pequeños que también viven dentro de nuestra aplicación, por ende, estos también deben ser señalados como una dependencia en nuestro `app.js`. Estos controladores operarán sobre los distintos módulos de nuestro sistema.

Ejemplo:

```
<script>
  // En los [ ] llamamos las dependecias como el 'otro-controlador'
  var app= angular.module('aplicacion',['otro-controlador']);
  app.controller('AplicacionController', function($scope){
    $scope.ejemplo='Esto es un ejemplo de Angular JS';
  });
</script>
```

```
<script>
  // Nombre de la dependencia creada es 'otro-controlador'
  var app= angular.module('otro-controlador',[]);
  app.controller('OtroControlador', function($scope){
    $scope.saludo='Hola Mundo';
  });
</script>
```

5 Desarrollo de Directivas

¿Que es una directiva? Las directivas de Angular sirven para encapsular código HTML de manera que nuestras aplicaciones puedan tener elementos reutilizables, también conocidos como componentes.

AngularJS permite definir directivas propias, estas podrán ser creadas a nivel de elemento `‘E’` o atributo `‘A’` dentro de un tag de HTML.

Las creamos en un archivo Javascript que debe ser importado como dependencia en nuestro app.js y debe contener un template de HTML (similar a lo que se hace con `ng-include`).

Ejemplo:

Crearemos dos directivas que nos permitirán crear un componente de **bootstrap.css** `"bootstrap panel"`

Estas directivas incluirán el fragmento correspondiente de HTML como Elemento `'E'` : `<mi-panel2></mi-panel2>` y como atributo `'A'` : `<div mi-panel1></div>`

El template del panel consistirá en el siguiente fragmento HTML:

```
<div class="panel panel-primary">
  <div class="panel-heading">Panel creado en app/views/panel.html</div>
  <div class="panel-body">
    <img class="img-responsive img-rounded" src="lib/img/img1.jpg">
  </div>
</div>
```

El código en el archivo `directives.js` ubicado en `app/directives.js` :

```
'use strict';

(function () {
  var direc = angular.module('directives-example',[]);

  direc.directive('miPanel1',function(){
    return {
      restric : 'A',
      templateUrl : 'app/views/panel.html'

    }
  });

  direc.directive('miPanel2',function(){
    return {
      restric : 'E',
      templateUrl : 'app/views/panel.html'

    }
  });
})();
```

El código en nuestro `AplicacionController.js` ubicado en `app/controllers/` es:

```
'use strict';

(function () {
  var controlador = angular.module('app-controller',[]);
  controlador.controller('applicationController', ['$scope', function($scope){
    $scope.ejemplo='Esto es un ejemplo de Angular JS';
  }]);
})();
```

El código en `app.js` en la ruta `app/app.js` es:

```
'use strict';

(function(){
    var app= angular.module('app',['directives-example', 'app-controller']);
})();
```

El código en nuestro `index.html` será:

```
<!-- Archivo index.html es el archivo principal, este contiene la aplicación que desar
rollaremos en angular
  además del controlador principal, esta página recibirá toda la lógica que implemente
mos en nuestro proyecto
  basado en esta tecnologia para el FrontEnd de nuestro sistema -->
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Ejemplo Angular</title>
  <!-- hojas de estilo, Bootstrap 3 y una personalizada -->
  <link rel="stylesheet" href="lib/css/bootstrap.min.css">
  <link rel="stylesheet" href="lib/css/main.css">
  <!-- Archivos Javascript -->
  <script src="lib/js/bootstrap.min.js"></script>
  <script src="lib/js/angular.min.js"></script>
  <script src="app/directives.js"></script>
  <script src="app/controllers/AplicacionController.js"></script>
  <script src="app/app.js"></script>
</head>
<body ng-app="app" ng-controller="applicationController">
  <div class="container">
    <h1 ng-bind="ejemplo"></h1>

    <div class="row">
      <div class="col-lg-4">
        <div mi-panel1></div>
      </div>
      <div class="col-lg-4">
        <mi-panel2></mi-panel2>
      </div>
    </div>
  </div>
</body>
</html>
```

Resultado

Los paneles resultantes fueron:

donde el panel de la izquierda es llamado como atributo y el de la derecha como elemento.

6 Filtros

Para poder excluir, ordenar u operar cierta información en nuestra web, Angular también soporta filtros, algunos existentes y otros que podemos desarrollar, un filtro como su nombre bien lo dice nos permite seleccionar cierta información a mostrar y generalmente es sobre un array, por ende es importante dar un correcto uso de estos cuando debemos mostrar cierta información.

Ejemplo: Ordenar por un valor en un array:

```
<div ng-app="miApp" ng-controller="AppController">

<p>Iterando los objetos ordenados por pais:</p>
<ul>
  <li ng-repeat=" x in nombres | orderBy:'pais' ">
    {{ x.nombre + ', ' + x.pais }}
  </li>
</ul>

</div>

<script>
angular.module('miApp', []).controller('AppController', function($scope) {
    $scope.nombres = [
        { nombre : 'Jani' , pais : 'Noruega' },
        { nombre : 'Carl' , pais : 'Suecia' },
        { nombre : 'Margareth' , pais : 'Reino Unido' },
        { nombre : 'Hege' , pais : 'Noruega' },
        { nombre : 'Joe' , pais : 'Dinamarca' },
        { nombre : 'Gustav' , pais : 'Suecia' },
        { nombre : 'Birgit' , pais : 'Dinamarca' },
        { nombre : 'Mary' , pais : 'Reino Unido' },
        { nombre : 'Kai' , pais : 'Noruega' }
        ];
});
</script>
```

Filtro Personalizado:

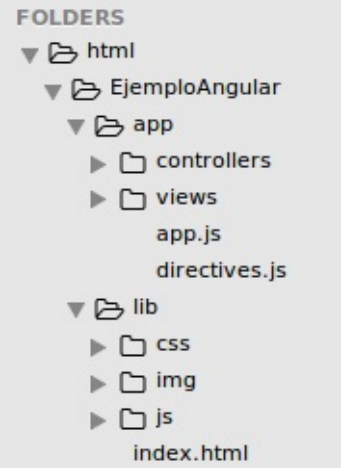
```
<ul ng-app="miApp" ng-controller="appCtrl">
<li ng-repeat="x in names">
    {{x | miFiltro}}
</li>
</ul>

<script>
var app = angular.module('miApp', []);

app.filter('miFiltro', function() {

    return function(x) {
        var i, c, txt = "";
        for (i = 0; i < x.length; i++) {
            c = x[i];
            if (i % 2 == 0) {
                c = c.toUpperCase();
            }
            txt += c;
        }
        return txt;
    };
});
app.controller('appCtrl', function($scope) {
    $scope.nombres = [
        'Jani',
        'Carl',
        'Margareth',
        'Hege',
        'Joe',
        'Gustav',
        'Birgit',
        'Mary',
        'Kai'
        ];
});
</script>
```

7. Estructura del proyecto y IIFE

Estructura del proyecto (Frontend):

Si bien hemos hablado de muchas funcionalidades que podemos desarrollar en angular, ahora, para ordenar un poco toda esta abstracción podemos separar esto en dos carpetas principales, una /app que contenga toda la lógica de nuestros controladores y directivas desarrollada por nosotros mismos y otra /lib que contenga todas las bibliotecas y distintos códigos fuentes de Frameworks a usar (Para el Frontend).

Ejemplo:



IIFE (Immediately-Invoked Function Expression)

Como se puede observar en algunos de los ejemplos, la siguiente expresión es usada para declarar fragmentos de código en javascript:

```
(function () {
  //código en javascript
})();
```

Esta expresión llamada IIFE, (expresión de función de invocación inmediata), es usada para declarar variables de forma estática, es decir, que estas no son visibles al mundo exterior. (Por ejemplo, acceder a las variables o funciones desde la consola del navegador)

En el caso de los ejemplos, cada definición de controladores, filtros, etc. solo será accesible para angular a través de `Angular.module`, y no hay necesidad de exponer los módulos fuera de ese contexto.

Además del encapsulamiento de las propiedades en JavaScript, esta expresión permite evitar (imposibilita) la definición de **variables globales**, que además de ser un **muy mal estilo de programación** emplearlas, son una gran fuente de errores de difícil seguimiento y que dificultan la modularización del código (aumentando el acoplamiento o dependencia entre los componentes).

8 Actividades

1 - Dada la siguiente lista de objetos:

```
var users = [
  {
    nombre: 'Juan',
    apellido: 'Nieves',
    imagen: 'https://pbs.twimg.com/profile_images/613894506480439296/MGWBzW3B.jpg',
    vivo: true
  },
  {
    nombre: 'Daenerys',
    apellido: 'Targaryen',
    imagen: 'https://pbs.twimg.com/profile_images/699219278927675393/de5Cjrju.png',
    vivo: false
  },
  {
    nombre: 'Eddard',
    apellido: 'Stark',
    imagen: 'https://pbs.twimg.com/profile_images/623606708531818496/8ap1ZAb6.jpg',
    vivo: true
  }
];
```

por cada objeto de la lista deberá hacer una carta con el siguiente formato:

esto debe completarse con los valores de los objetos correspondientes.

Para el desarrollo de esta actividad use lo aprendido en la guía para crear las tarjetas sin duplicar el código de la vista (defina la estructura HTML de la tarjeta solo una vez en la página). Para el diseño puede usar las hojas de estilo que estime conveniente.

2 - A continuación deberá crear un formulario que permita agregar más personajes a la lista. El diseño del formulario debe ser ad hoc al de las cartas de personajes.