

Introducción a



Una super guía para un
Framework Super Heróico



KEEP CALM
AND HACK



Keep Calm And Hack

Nosotros ponemos los recursos, tú el entusiasmo

KeepCalmAndHack es un colectivo de profesionistas comprometidos con el desarrollo de la comunidad de tecnologías de la información de Latinoamérica. El propósito del presente documento es proveerte de una introducción sencilla a los beneficios del desarrollo bajo el framework de AngularJS.

Version

Version	Descripción	Fecha	Autor
1.0	Primera version	Feb 24, 2016	Cristian Colorado KeepCalmAndHack.com

Introducción a AngularJS

AngularJS es un framework de JavaScript desarrollado por Google cuyo objetivo es proveer de una herramienta de desarrollo front-end para el patrón de diseño MVC.

AngularJS extiende el código HTML a través de atributos, dichos atributos son interpretados como directivas, las cuales tienen asignado un comportamiento, entre las que destacan ligar datos con nuestro HTML.

Comenzaremos nuestra introducción revisando algunos de los conceptos principales de AngularJS.

Bootstrapping

Bootstrapping es el proceso de inicialización, mediante el cual AngularJS lee nuestro HTML, interpreta y evalúa las directivas necesarias para implementar la lógica necesaria.

Una vez que el navegador carga nuestra librería, AngularJS espera a que nuestra página esté en estado completo. Posteriormente busca la directiva NG-APP, la cual especifica cual es el segmento de HTML que servirá como raíz para nuestra SPA.

Finalmente interpreta todas las expresiones dentro de nuestro HTML y despliega el resultado en la vista.

```
1 <html>
2   <head>
3     <title>Bootstrapping</title>
4   </head>
5   <body ng-app>
6     <h1>Hello World!</h1>
7     <h2> 1 + 2 = {{ 1 + 2 }}</h2>
8     <script type="text/javascript" src="angular.js"></script>
9   </body>
10 </html>
```

La forma más simple de desplegar datos en la vista se realiza mediante la interpolación, en la cual se encapsula una expresión a ser evaluada por AngularJS entre llaves dobles `{{ }}`.

En nuestro ejemplo podemos observar en la línea 7, que la expresión `"1 + 2"` es encapsulada entre llaves dobles. Dicha expresión es evaluada y desplegada por AngularJS.

Hello World!

1 + 2 = 3

La interpolación también nos sirve para desplegar elementos de nuestro modelo, lo cual es conocido como "Data Binding", un concepto que abordaremos a detalle más adelante.

Módulos & Controladores

Un módulo es un contenedor dentro el cual almacenaremos componentes como controladores, servicios, directivas personalizadas, etc. Los cuales en conjunto representan la lógica de negocios de nuestra aplicación.

Podemos crear un módulo haciendo uso del método `"module"` del objeto `"angular"`. Este método recibe dos parametros, primero el nombre del módulo, el segundo parámetro es un arreglo con la lista de dependencias necesarias para que el código funcione correctamente.

```
1 var todoApp = angular.module("todoApp", []);
2
3 todoApp.controller("mainController", function() {
4     this.user = {
5         name: "Cristian"
6     }
7 });
```

Utilizaremos nuestro módulo para crear componentes como controladores los cuales especifican el comportamiento para un segmento de nuestra vista.

Podemos crear un controlador haciendo uso del método “*controller*” de nuestro objeto módulo. Este método recibe dos parametros, el primero es el nombre de nuestro controlador, el segundo parámetro es una función que encapsula la nuestra lógica de negocios. En nuestro ejemplo creamos un nuevo objeto dentro de nuestro controlador con el objetivo de utilizarlo en nuestra vista.

Sabias que...

Este tipo de declaración es conocida como declaración implícita, y tiene consecuencias negativas en caso de que se quiera optimizar el código usando herramientas de minificación. Se aconseja realizar declaración explícita, la cual abordaremos en nuestro blog más adelante.

```
1 <html>
2   <head>
3     <title>TODO</title>
4     <link href="public/assets/css/bootstrap.css" type="text/css" rel="stylesheet" />
5   </head>
6   <body ng-app="todoApp">
7     <div class="container" ng-controller="mainController as mainCtrl">
8       <div class="row">
9         <h2>Bienvenido {{mainCtrl.user.name}}!</h2>
10      </div>
11    </div>
12    <script type="text/javascript" src="public/assets/js/angular.js">
13    </script>
14    <script type="text/javascript" src="public/assets/js/app.js"></script>
15  </body>
16 </html>
```

En nuestro HTML, definiremos nuestra directiva NG-APP a nivel body para especificar este como elemento raíz de nuestra aplicación, y mediante esta directiva definimos el módulo *todoApp* como el contenedor de nuestra lógica de negocios.

Durante el proceso de bootstrapping, AngularJS cargará el módulo así como las dependencias necesarias para su funcionamiento.

A continuación utilizamos la directiva NG-CONTROLLER para asignar un controlador a un segmento de la vista, en este caso “mainController” contendrá toda la lógica necesaria para que nuestro div “container” funcione correctamente (ver línea 7).

Para ello utilizaremos la siguiente sintaxis:

<nombre del controlador> as <alias>

Especificamos el nombre de nuestro controlador y utilizamos la palabra reservada “as” para asignar un alias, el cual podemos usar dentro de nuestra vista para acceder a los detalles de su implementación.

En nuestro ejemplo, accedemos al objeto “user” e interpolamos su nombre dentro de nuestro título, para dar la bienvenida al estudiante (ver línea 8).



Sabías que...

NG-CONTROLLER originalmente solo recibe la referencia al nombre del controlador declarado en nuestro módulo, la sintaxis utilizada en este ejemplo permite realizar una abstracción, a través de la cual nuestro controlador es adicionado al objeto que agrupa los elementos de nuestro modelo conocido como scope. El objetivo es facilitar la lectura del código. Detalles sobre el scope serán proporcionados más adelante a través de nuestro blog.

Data Binding

Anteriormente utilizamos interpolación “{{ }}” como la forma más sencilla de realizar data binding.

No obstante se aconseja hacer uso de la directiva NG-BIND, la cual oculta el elemento de html hasta que Angular haya evaluado la expresión, de esta manera se evita que el usuario vea el código en caso de que haya un retraso el proceso de carga.

```
1 <html>
2   <head>
3     <title>TODO</title>
4     <link href="public/assets/css/bootstrap.css" type="text/css" rel="stylesheet" />
5   </head>
6   <body ng-app="todoApp">
7     <div class="container" ng-controller="mainController as mainCtrl">
8       <div class="row">
9         <h2>Bienvenido <span ng-bind="mainCtrl.user.name"></span>!</h2>
10      </div>
11    </div>
12    <script type="text/javascript" src="public/assets/js/angular.js">
13    </script>
14    <script type="text/javascript" src="public/assets/js/app.js"></script>
15  </body>
16 </html>
```

Para ello hemos creado un nuevo elemento tipo *span* dentro de nuestro título, al cual agregamos la directiva *ng-bind* y proveemos a través de la expresión el objeto a evaluar.

A este tipo de data binding se le conoce como ligado de datos unidireccional (One Way Data Binding), debido a que nuestros datos son actualizados en una dirección, esto es, de nuestro controlador a la vista.

Colecciones

AngularJS nos provee de una directiva que podemos utilizar para desplegar datos más complejos como lo son las colecciones o arreglos. NG-REPEAT utiliza un segmento de nuestro HTML como template y lo usa para reproducir un segmento de vista por cada elemento de un arreglo.

Para este ejemplo agregaremos a nuestro objeto `user` un arreglo, que representa una lista de tareas así como su estatus:

```
1 var todoApp = angular.module("todoApp", []);
2
3 todoApp.controller("mainController", function() {
4     this.user = {
5         name: "Cristian",
6         tasks: [
7             { name: 'Estudiar AngularJS', complete: false },
8             { name: 'Aplicar a trabajo', complete: false },
9             { name: 'Tomar vacaciones', complete: false }
10        ]
11    };
12 });
```

Para desplegar esta lista de tareas, dentro de nuestra vista, crearemos una nueva tabla con dos columnas: tarea y completa. Así como un renglón que nos servirá como template, donde desplegamos la información de cada tarea (ver línea 17-20).

```
1 <html>
2   <head>
3     <title>TODO</title>
4     <link href="public/assets/css/bootstrap.css" type="text/css" rel="stylesheet" />
5   </head>
6   <body ng-app="todoApp">
7     <div class="container" ng-controller="mainController as mainCtrl">
8       <div class="row">
9         <h2>Bienvenido <span ng-bind="mainCtrl.user.name"></span>!!</h2>
10      </div>
11      <div class="row">
12        <table class="table">
13          <thead>
14            <th>Tarea</th>
15            <th>Completa</th>
16          </thead>
17          <tr>
18            <td><!-- task.name --></td>
19            <td><!-- task.complete --></td>
20          </tr>
21        </table>
22      </div>
23    </div>
24    <script type="text/javascript" src="public/assets/js/angular.js">
25    </script>
26    <script type="text/javascript" src="public/assets/js/app.js"></script>
27  </body>
28 </html>
```

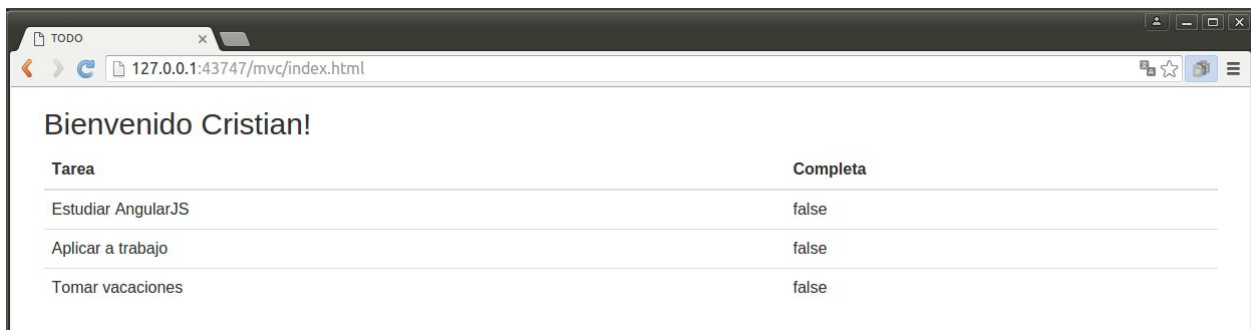
Utilizaremos la directiva `NG-REPEAT` para iterar nuestras tareas, para ello deberemos enviar la expresión con la sintaxis:

`<iterador> in <colección>`

AngularJS iterará sobre cada elemento de nuestra colección y para cada uno creará una vista basado en nuestro template(nuestro elemento tr), al cual le dará acceso al elemento de nuestra colección a través de la variable iterador. Podremos usar esa variable dentro de nuestro template para desplegar información.

```
1 <html>
2   <head>
3     <title>TODO</title>
4     <link href="public/assets/css/bootstrap.css" type="text/css" rel="stylesheet" />
5   </head>
6   <body ng-app="todoApp">
7     <div class="container" ng-controller="mainController as mainCtrl">
8       <div class="row">
9         <h2>Bienvenido <span ng-bind="mainCtrl.user.name"></span>!</h2>
10      </div>
11      <div class="row">
12        <table class="table">
13          <thead>
14            <th>Tarea</th>
15            <th>Completa</th>
16          </thead>
17          <tr ng-repeat="t in mainCtrl.user.tasks">
18            <td ng-bind="t.name"></td>
19            <td ng-bind="t.complete"></td>
20          </tr>
21        </table>
22      </div>
23    </div>
24    <script type="text/javascript" src="public/assets/js/angular.js">
25  </script>
26    <script type="text/javascript" src="public/assets/js/app.js"></script>
27  </body>
28 </html>
```

El resultado de nuestro código será similar al siguiente:



Tarea	Completa
Estudiar AngularJS	false
Aplicar a trabajo	false
Tomar vacaciones	false

Sabias que...

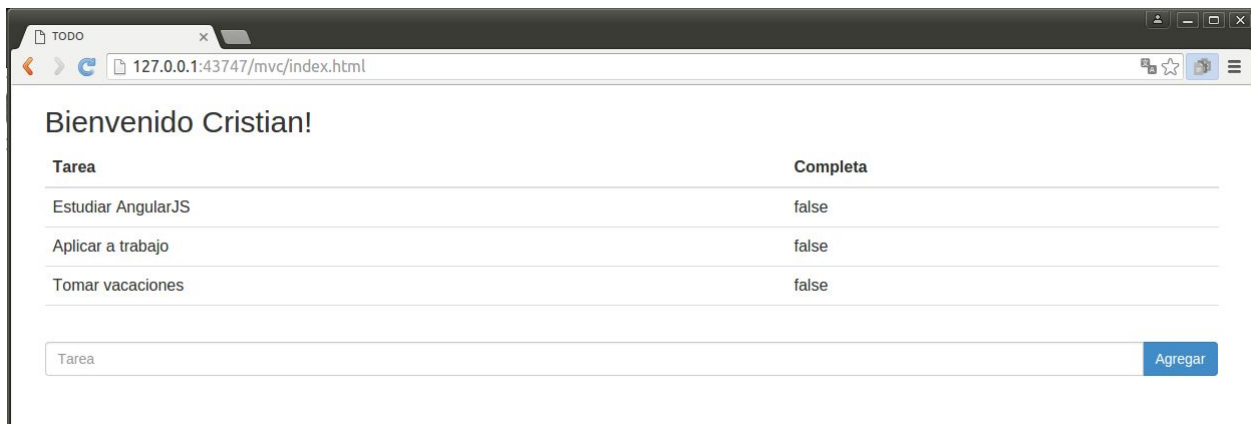
Existen múltiples variaciones para la sintaxis utilizada en la directiva NG-REPEAT, las cuales nos permiten agregar filtros sobre los datos iterados o organizarlos con base en cierto campo. Para saber más sobre estas características sigue nuestro blog.

Two Way Data Binding

Durante el proceso de bootstrapping nuestro HTML es compilado por AngularJS, lo cual produce una vista que es desplegada en nuestro navegador. Cualquier cambio hecho en la vista inmediatamente se ve reflejado en nuestro modelo, y cualquier cambio hecho a nuestro modelo es propagado a la vista. AngularJS utiliza nuestro modelo como la fuente única y absoluta de datos en nuestra aplicación, por lo que el desarrollador solo debe concentrarse en mantener esa fuente de datos actualizada.

La sincronización provista por AngularJS entre vista y modelo se le conoce como Ligado de datos bidireccional (Two Way Data Binding).

Para ejemplificar este concepto, agregaremos una forma a nuestra aplicación, la cual nos permitirá añadir una nueva tarea a nuestra lista. Misma que se actualizará en tiempo real en nuestra tabla.



Tarea	Completa
Estudiar AngularJS	false
Aplicar a trabajo	false
Tomar vacaciones	false

Tarea

Para ello, en nuestro controlador, crearemos una nueva propiedad la cual almacenará la nueva tarea de nuestra forma(#13), temporalmente:

```
1 var todoApp = angular.module("todoApp", []);
2
3 todoApp.controller("mainController", function() {
4     this.user = {
5         name: "Cristian",
6         tasks: [
7             { name: 'Estudiar AngularJS', complete: false },
8             { name: 'Aplicar a trabajo', complete: false },
9             { name: 'Tomar vacaciones', complete: false }
10        ]
11    }
12
13    this.task = {};
14 });
```

En nuestra tabla agregaremos un renglón adicional(#21-24), el cual estara ligado a la propiedad task de nuestro controlador, en él se mostrará un borrador o preview de la nueva actividad.

```
11 <div class="row">
12     <table class="table">
13         <thead>
14             <th>Tarea</th>
15             <th>Completa</th>
16         </thead>
17         <tr ng-repeat="t in mainCtrl.user.tasks">
18             <td ng-bind="t.name"></td>
19             <td ng-bind="t.complete"></td>
20         </tr>
21         <tr>
22             <td ng-bind="mainCtrl.task.name"></td>
23             <td></td>
24         </tr>
25     </table>
26 </div>
```

Agregaremos una nueva forma en nuestra vista, la cual consta de un control de texto(#30) y un botón para agregar nuestra nueva tarea(#32).

```

27     <div class="row">
28       <form class="form">
29         <div class="input-group">
30           <input type="text" class="form-control" placeholder="Tarea">
31           <div class="input-group-btn">
32             <button class="btn btn-primary">Agregar</button>
33           </div>
34         </div>
35       </form>
36     </div>

```

Para realizar data binding en un control, AngularJS nos provee de la directiva NG-MODEL, la cual recibe como expresión la referencia a la propiedad "name" del objeto task de nuestro controlador:

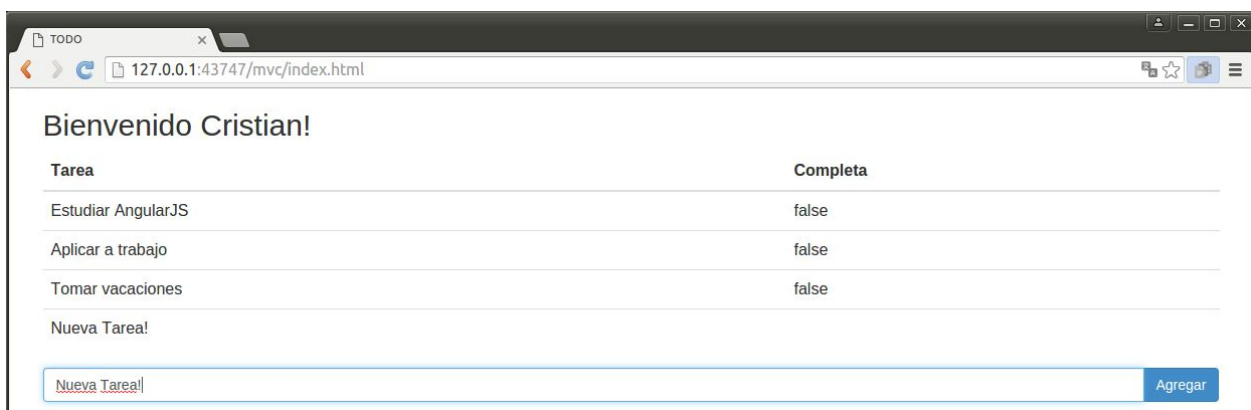
```

27     <div class="row">
28       <form class="form">
29         <div class="input-group">
30           <input type="text" class="form-control"
31             placeholder="Tarea" ng-model="mainCtrl.task.name">
32           <div class="input-group-btn">
33             <button class="btn btn-primary">Agregar</button>
34           </div>
35         </div>
36       </form>
37     </div>

```

En este punto tenemos una propiedad en nuestro controlador que es desplegada en dos segmentos de la vista, un renglón dentro de nuestra tabla y en control dentro de una forma.

AngularJS se encarga de sincronizar los detalles de nuestro modelo(la propiedad task) hacia todos los elementos de la vista que lo utilizan(nuestro control y nuestra tabla), en tiempo real y de manera automática.



Eventos

AngularJS provee de una lista de directivas para manejar eventos generados por elementos. Dichas directivas ejecutarán una expresión la cual puede ser ligada con un segmento de código en nuestro controlador.

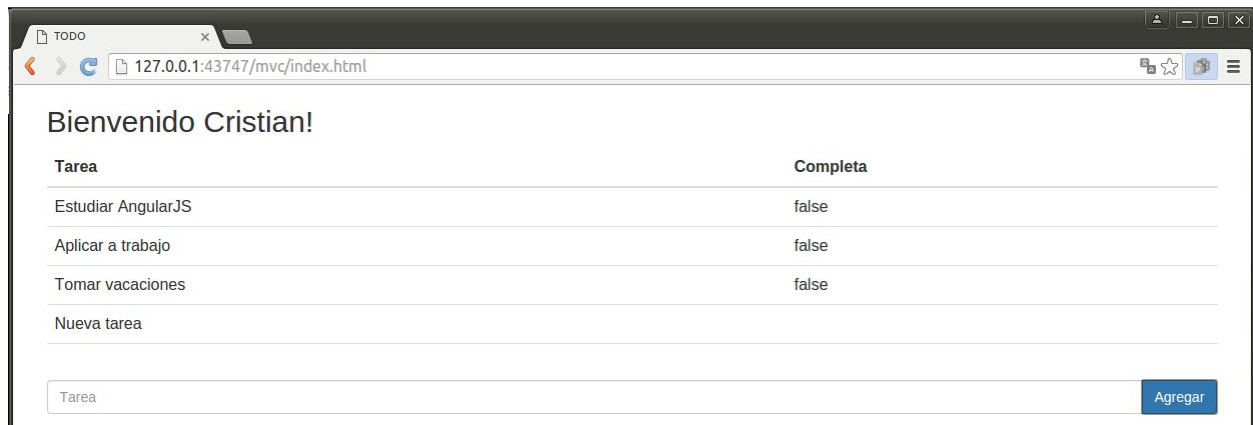
Primero crearemos una nueva función(#15) dentro de nuestro controlador, que estara a cargo de agregar la propiedad “task” dentro de nuestra lista de tareas, posteriormente reiniciará nuestra propiedad task para que pueda ser reutilizada.

```
1 var todoApp = angular.module("todoApp", []);
2
3 todoApp.controller("mainController", function() {
4     this.user = {
5         name: "Cristian",
6         tasks: [
7             { name: 'Estudiar AngularJS', complete: false },
8             { name: 'Aplicar a trabajo', complete: false },
9             { name: 'Tomar vacaciones', complete: false }
10        ]
11    }
12
13    this.task = {};
14
15    this.addTask = function() {
16        this.user.tasks.push(this.task);
17        this.task = {};
18    }
19 });
```

Para invocar esta función utilizaremos la directiva NG-CLICK, la cual recibe como expresión el código a ejecutar cuando el control sea presionado. En este caso accedemos al método “addTask” de nuestro controlador.

```
27 <div class="row">
28     <form class="form">
29         <div class="input-group">
30             <input type="text" class="form-control"
31                 placeholder="Tarea" ng-model="mainCtrl.task.name">
32             <div class="input-group-btn">
33                 <button class="btn btn-primary" ng-click="mainCtrl.addTask()">
34                     Agregar</button>
35             </div>
36         </div>
37     </form>
38 </div>
```


Ahora nuestra aplicación puede agregar nuevas tareas a nuestra lista:



Finalmente podemos agregar controles de tipo checkbox dentro de nuestra tabla para manejar el estatus. Utilizaremos la directiva NG-MODEL para realizar el data binding hacia las propiedades necesarias(#20, #26).

```
11      <div class="row">
12          <table class="table">
13              <thead>
14                  <th>Tarea</th>
15                  <th>Completa</th>
16              </thead>
17              <tr ng-repeat="t in mainCtrl.user.tasks">
18                  <td ng-bind="t.name"></td>
19                  <td>
20                      <input type="checkbox" ng-model="t.complete"/>
21                  </td>
22              </tr>
23              <tr>
24                  <td ng-bind="mainCtrl.task.name"></td>
25                  <td>
26                      <input type="checkbox" ng-model="mainCtrl.task.complete">
27                  </td>
28              </tr>
29          </table>
30      </div>
```

También agregaremos un control de tipo checkbox en nuestra forma(#37):


```
31     <div class="row">
32       <form class="form">
33         <div class="input-group">
34           <input type="text" class="form-control"
35             placeholder="Tarea" ng-model="mainCtrl.task.name">
36           <div class="input-group-addon">
37             <input type="checkbox" ng-model="mainCtrl.task.complete">
38           </div>
39           <div class="input-group-btn">
40             <button class="btn btn-primary" ng-click="mainCtrl.addTask()">
41               Agregar</button>
42           </div>
43         </div>
44       </form>
45     </div>
```

De esta manera ahora podemos actualizar el status de nuestras tareas:

Tarea	Completa
Estudiar AngularJS	<input type="checkbox"/>
Aplicar a trabajo	<input type="checkbox"/>
Tomar vacaciones	<input type="checkbox"/>
Something	<input checked="" type="checkbox"/>

Something ☐ Agregar

Sabias que...

AngularJS provee de servicios de validación en formas y controles, de tal manera que el usuario puede ser notificado sobre entradas inválidas antes de enviar los datos. Esto abona a la experiencia de usuario. Para aprender más sobre estas validaciones mantente al tanto de nuestro blog.

Qué es lo que sigue?

Felicidades por haber completado nuestro ebook, esperamos que te haya dado una idea de los beneficios de utilizar este framework. No obstante existen aún muchos conceptos que deberás aprender para poderle sacar un mayor provecho a AngularJS como lo son:

- Controladores anidados.
- Scopes & Scopes anidados.
- Inyección de dependencias.
- Service y custom services.
- Validaciones sobre formas.
- Rutas y vistas.
- Custom Directives.
- Animaciones.

Te invitamos a seguir de cerca nuestro [blog](#) y [facebook](#) para explorar a detalle estas y más funcionalidades.

Codigo

Puedes encontrar el proyecto desarrollado durante esta guía en nuestro repositorio en github:

Repo:

Estructura de la aplicación

La estructura de una aplicación desarrollada con AngularJS está compuesta por los siguientes elementos:

Archivo	Path	Descripcion
index.html	/	este archivo contiene el código html de nuestra SPA. En el se importan las librerías necesarias y se definen las directivas.
app.js	/public/assets/js	Este archivo contiene la lógica de negocios necesaria para desempeñar todas las funcionalidades de nuestro SPA.
*.css	/public/assets/css	En este folder se almacenan los archivos de estilos utilizados por la aplicación.

Licencia

Attribution-NonCommercial-ShareAlike 4.0 International

