

## ANGULARJS

---

Para la realización de esta práctica se ha realizado un proyecto global llamado AngularJS practice del que cuelgan distintas carpetas con el código empleado para la realización de cada ejercicio. Se adjuntara este archivo con la explicación del código empleado y una captura de pantalla con los resultados obtenidos.

### EJERCICIO 1:

*Desarrolla una página web equivalente que permita introducir los campos de un jugador de baloncesto (te puedes basar en los atributos que hemos utilizado con Spring), y muestre en tiempo real el valor introducido.*

*Explica, de manera muy resumida, el mecanismo que utiliza AngularJS para sincronizar los elementos de la interfaz de usuario HTML (explica la directiva específica que se utiliza en este ejemplo).*

### CAPTURA:



http://localhos...201/index.html x +

localhost:8383/AngularJS practice/Exercise 1/index.html

Enter player statistics:

Player name:

Baskets:

Assists:

Ferran Buireu

178

99

### EXPLICACIÓN:

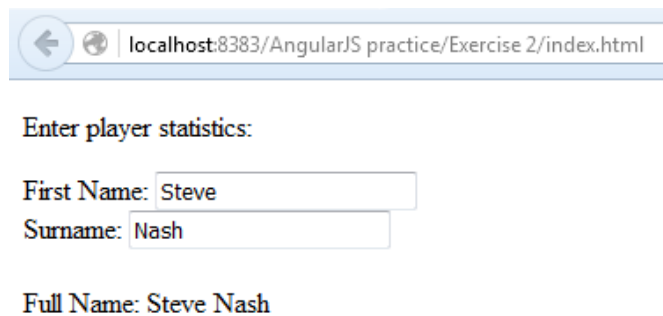
En este ejercicio se usa la directiva *ng-model* para crear un *binding* en tiempo real. El texto insertado en el *input* es mostrado usando el nombre del *ng-model* y las llaves ({}): {{name}}

En este caso se mostrara las variaciones que sufra el input con el *ng-model* de llamado *name*, ya que las dobles llaves hacen referencia al nombre que contiene y son capaces de mostrarlo en tiempo real.

## **EJERCICIO 2:**

*Extiende el ejercicio número 1, de manera que se incluya un controlador que gestione los datos de la vista. El controlador ha de proporcionar unos datos iniciales por defecto que se mostrarán en la vista. Cuando el usuario actualiza los elementos de la vista (los elementos de la interfaz de usuario en HTML), ¿se actualizan automáticamente los datos en el controlador? Justifica tu respuesta.*

### **CAPTURA:**



Enter player statistics:

First Name:

Surname:

Full Name: Steve Nash

### **EXPLICACIÓN:**

Éste caso es parecido al del ejercicio anterior, con la diferencia que se usa la directiva de *AngularJS* llamada *ng-controller* y el *ng-model*.

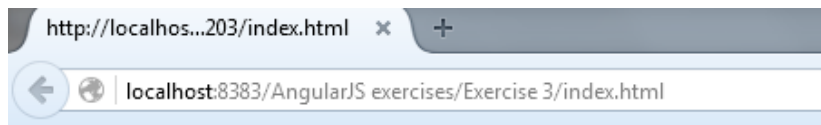
El código insertado dentro del *ng-controller* se muestra usando de igual manera que en el ejercicio anterior, con las dobles llaves, llamando a *firstName* y a *surname*.

En el *ng-controller*, se usa el *scope* para que el nombre por defecto insertado en el input sea el nombre que se halla en el *scope*. Éste se encarga de trabajar con el modelo y pasar los datos en tiempo real.

### **EJERCICIO 3:**

Utilizando la directiva *ng-init*, inicializa un objeto jugador de baloncesto con sus diversos atributos. Posteriormente, muestra sus atributos en diversos elementos HTML mediante la directiva *{{}}*. Explica en qué componentes de AngularJS se debe manipular el DOM: directivas, controladores, vistas o servicios. Justifica tu respuesta.

### **CAPTURA:**



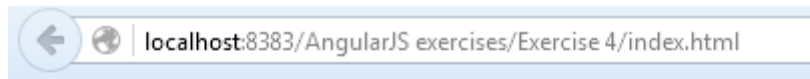
### **EXPLICACIÓN:**

La directiva *ng-init* inicializa un objeto y le asigna unos valores. Posteriormente se puede acceder a dichos elementos mediante las dobles llaves. *AngularJS* es capaz de manipular el DOM a todos los niveles, lo que se conoce como estilo imperativo. Se consigue una solución unidireccional a un problema bidireccional en aquellos lugares donde no se ha especificado identificador o clase.

#### **EJERCICIO 4:**

*Inicializa un array para almacenar diversos jugadores de baloncesto. Posteriormente, utiliza la directiva `ng-repeat` para mostrar todos los jugadores de baloncesto con sus diversos atributos. Explica cómo funciona la gestión de scopes con la directiva `ng-repeat`.*

#### **CAPTURA:**



Looping with objects:

- Steve, Canada
- Ferran, Catalonia
- Alessandro, Italy

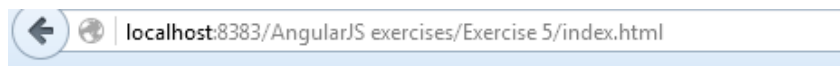
#### **EXPLICACIÓN:**

En éste ejercicio se ha creado un *array* de objetos utilizando la directiva del ejercicio anterior *ng-init* para crear jugadores y los atributos de los mismos. Posteriormente utilizando el *ng-repeat* se crea un bucle muy simple dónde se pueden llamar a los objetos y sus atributos usando la *key* adecuada (x en este caso) y el nombre del *ng-init*, con éste método se puede acceder al *scope* de los objetos creados, ya que permite a la vista utilizarlo fácilmente.

### **EJERCICIO 5:**

Inicializa un array que almacene diversos jugadores de baloncesto en un controlador. Posteriormente, utiliza la directiva *ng-repeat* para mostrar todos los jugadores. Utiliza también un filtro, para mostrar los jugadores ordenando mediante varios atributos (por ejemplo, nombre, total canastas, total asistencias, país, etc.). La idea es que puedes experimentar con diversos atributos para comprobar que la ordenación funciona correctamente con todos los tipos de datos. Justifica en qué casos es preferible realizar la ordenación en el cliente mediante AngularJS, y en qué casos es preferible realizar la ordenación en el servidor mediante Spring. Explica ventajas e inconvenientes de cada enfoque.

### **CAPTURA:**



Looping with players (Name|Country|Total basket(ordered by)|Total asist):

- Heather, Norway , 12 , 78
- Will, Sweden , 13 , 91
- Steve, Canada , 20 , 580
- David, Spain , 28 , 100
- Charly, Denmark , 34 , 56
- Tracy, USA , 70 , 18
- Alfred, Denmark , 84 , 64

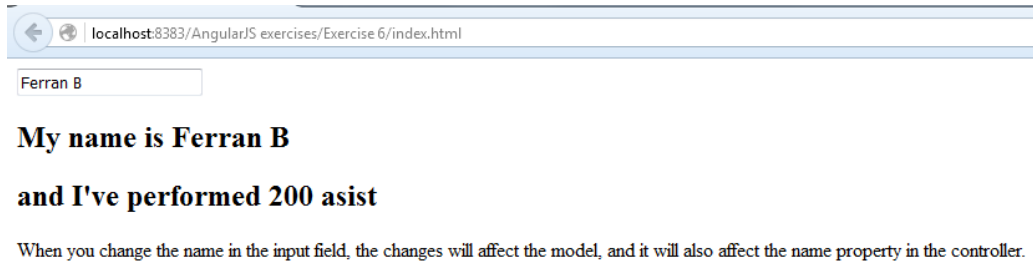
### **EXPLICACIÓN:**

Para la realización del ejercicio 5 se ha almacenado diversa información tanto de jugadores como de sus diversos atributos (canastas totales, asistencias totales, nombre y país). De la misma manera, se recorre mediante la directiva *ng-repeat* el *array* de jugadores. El controlador se encarga de mediar entre la vista y los objetos, usando el *scope* para guardar su información. Utilizando unas pocas líneas de código se puede recorrer un *array* de objetos y ordenarlos por cualquiera de sus atributos. Según cuales sean los datos que se manejan es preferible que la ordenación se haga mediante *Spring*, ya que no interesa que el usuario pueda acceder a ellos usando cualquier editor de código implementado en cualquier navegador, ya que de éste modo se impide al usuario acceder a los datos en el *front-end*. Otro motivo para filtrar en el *back-end* es por la eficiencia de la página, para impedir que se sobrecargue y se muestren muchos resultados.

## **EJERCICIO 6:**

*Repaso sobre la sincronización automática que realiza angular entre la vista, el controlador y el modelo. Experimenta con el siguiente código de ejemplo, comprobando la sincronización que se efectúa con los atributos de un jugador de baloncesto.*

### **CAPTURA:**



### **EXPLICACIÓN:**

La explicación de la función de las directivas *AngularJS* en éste ejercicio es muy parecido a los de los ejercicios anteriores. Los cambios que se realizan en el *input* afectan a las propiedades del nombre y se ven reflejadas en el controlador. Como siempre, el *scope* se ocupa de mediar entre la vista y el controlador, modificando los datos en tiempo real.

## **EJERCICIO 7:**

*Desarrolla una aplicación similar que obtenga un listado de jugadores de baloncesto de un API REST Spring. En primer lugar, debes programar el API REST Spring que devuelva el listado de jugadores de baloncesto, tal y como hemos hecho hasta ahora. Debes probar que el API REST funciona correctamente mediante una herramienta tipo:*

*Una vez que hayas comprobado que el API REST funciona correctamente, puedes consumir el listado de jugadores desde angular, tal y como enseña el código de ejemplo. Puedes guardar tu fichero HTML en la carpeta webapp de JHipster. Demuestra que sabes depurar correctamente la aplicación incluyendo capturas de pantalla del depurador en el navegador (por ejemplo Google Chrome), y también capturas del depurador en el Idea (para depurar el servidor basado en Spring).*

## **CAPTURA:**



- FC Barcelona Lassa, Barcelona
- Real Madrid, Madrid
- Phoenix Suns, Phoenix
- DKV Joventut, Badalona
- Club Baloncesto Málaga, Málaga
- JSF Nanterre, Nanterre

### **EXPLICACIÓN:**

Para la realización de éste ejercicio se ha usado la entidad *equipos* en vez *jugadores*, tal y como muestra el enunciado.

Los pasos que se han seguido son han sido los siguientes:

- Ejecución el proyecto y se comprueba en el *API* cómo la petición se ejecuta correctamente y se muestran los equipos en formato *JSON*.
- Creación del *script* (código en el repositorio correspondiente) y su copia en el directorio correspondiente del proyecto (*webapp* en éste caso).
- Arrancar el proyecto en modo *debug*.
- Acceder a la *URL* correspondiente del *script* (resultado en las capturas anteriores)

En éste caso, se ha programado el *API REST* de *Spring* para que devuelva una lista de equipos del proyecto en cuestión. A continuación se ha creado un *script* que selecciona los equipos del proyecto, los recorre y los muestra en pantalla tal y como se ha explicado en los ejercicios anteriores.

Dicho script, ejecuta una petición *HTTP* que devuelve los equipos en una función llamada *myData*, el *scope* es el intermediario entre dicha función y el bucle que recorre el *array* de objetos equipos para luego mostrarlos.



## EJERCICIO 9:

Extiende el ejercicio número 7 para obtener el listado de jugadores del API REST Spring. En este ejercicio, debes mostrar la información de los jugadores en formato tabla, especificando una columna para cada atributo del jugador.

## CAPTURA:

The screenshot shows an IDE with a project structure on the left, Java code in the center, and a browser window at the bottom.

**Project Structure:**

- dto
- errors
- util
  - AccountResource
  - AuditResource
  - EntrenadorResource
  - EquipoResource
  - LogsResource
  - package-info.java
  - UserResource
- Application
- ApplicationWebXml
- resources
  - config
  - i18n
  - mails
  - templates
  - banner.txt

**Java Code:**

```
method = RequestMethod.GET,
produces = MediaType.APPLICATION_JSON_VALUE)
@Timed
public List<Equipo> getAllEquipos() {
    log.debug("REST request to get all Equipos"); log: "Logger[cl
    return equipoRepository.findAll(); equipoRepository: "org.sp
}

/**
 * GET /equipos/{id} -> get the "id" equipo.
 */
@RequestMapping(value = "/equipos/{id}",
method = RequestMethod.GET,
produces = MediaType.APPLICATION_JSON_VALUE)
@Timed
public ResponseEntity<Equipo> getEquipo(@PathVariable Long id) {
    log.debug("REST request to get Equipo : {}", id);
    Equipo equipo = equipoRepository.findOne(id);
    return Optional.ofNullable(equipo)
        .map(result -> new ResponseEntity<>(
            result,
            HttpStatus.OK))
}
```

**Debugger:**

- Frames: "http-nio-8080-exec-2" @1...
- Variables: this = {EquipoResource@11859}, equipoRepository = {Proxy158@11864} "org.springframework.data.jpa.repository.support.SimpleJpaRepository@6a555576"

**Browser:** 127.0.0.1:8080/ex07.html

**Table:**

myApp			
1			
2	customersCtrl		
6	myData: [...]		
9	ngRepeat		
x:			
# id: 1			
# nombreEquipo: "FC Barcelona Lassa"			
# localidad: "Barcelona"			
10	ngRepeat		
x: {...}			4
11	ngRepeat		

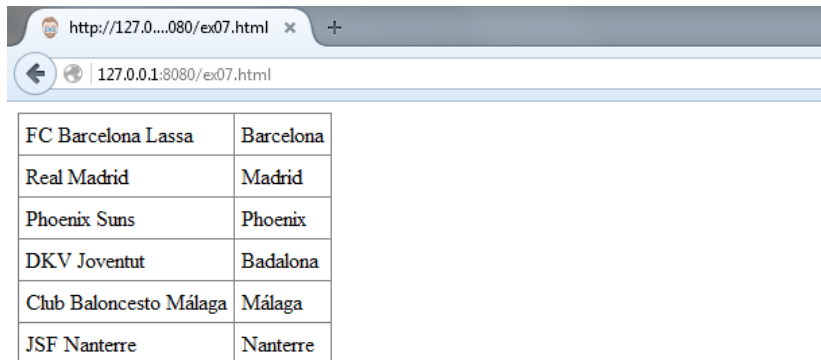
### **EXPLICACIÓN:**

Para realizar éste ejercicio se han seguido los pasos anteriormente descritos en el ejercicio 7, con la diferencia que se ha editado el código para que muestre los resultados en una pequeña tabla en el bucle que realiza la directiva *ng-repeat*. Es interesante observar, cómo añadiendo varios *breakpoints* (mostrados en las capturas anteriores), la petición que realiza nuestro pequeño *script* a *api/equipos* queda parada, demostrando así que efectivamente se realiza una petición a la *API* del proyecto. El complemento para observar código *AngularJS* en el navegador, a su vez también es útil para observar cómo funciona las directivas de *AngularJS*, concretamente el *ng-repeat*. Cómo viene siendo habitual, el *scope* sirve de interlocutor entre la vista y el controlador.

## **EJERCICIO 10:**

*Mejora la tabla del ejercicio anterior utilizando estilos CSS. Te puedes basar en el código de ejemplo proporcionado, y puedes proponer tus propios estilos.*

### **CAPTURA:**



A screenshot of a web browser window. The address bar shows the URL `http://127.0.0.1:8080/ex07.html`. Below the browser window, a table is displayed with two columns. The first column lists basketball teams, and the second column lists their locations. The teams are FC Barcelona Lassa, Real Madrid, Phoenix Suns, DKV Joventut, Club Baloncesto Málaga, and JSF Nanterre. The locations are Barcelona, Madrid, Phoenix, Badalona, Málaga, and Nanterre.

FC Barcelona Lassa	Barcelona
Real Madrid	Madrid
Phoenix Suns	Phoenix
DKV Joventut	Badalona
Club Baloncesto Málaga	Málaga
JSF Nanterre	Nanterre

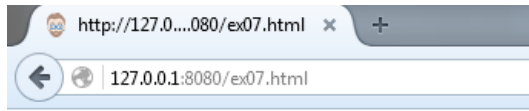
### **EXPLICACIÓN:**

Para realizar éste ejercicio se ha creado una tabla simple implementado encima de la directiva *ng-repeat* anteriormente explicada. Se ha usado el *tag style* para modificar el CSS para añadir estilo a la tabla. El estilo trabaja sobre los *tags table, th* i *td*.

## **EJERCICIO 11:**

*En este ejercicio, debes utilizar la nueva directiva `ng-if` para aplicar diversos estilos a las columnas en función de que la fila sea par o impar.*

### **CAPTURA:**



The screenshot shows a web browser window with the address bar displaying `http://127.0.0.1:8080/ex07.html`. Below the browser window, a table is displayed with two columns. The rows alternate between a light gray background and a white background, demonstrating the effect of the `ng-if` directive.

FC Barcelona Lassa	Barcelona
Real Madrid	Madrid
Phoenix Suns	Phoenix
DKV Joventut	Badalona
Club Baloncesto Málaga	Málaga
JSF Nanterre	Nanterre

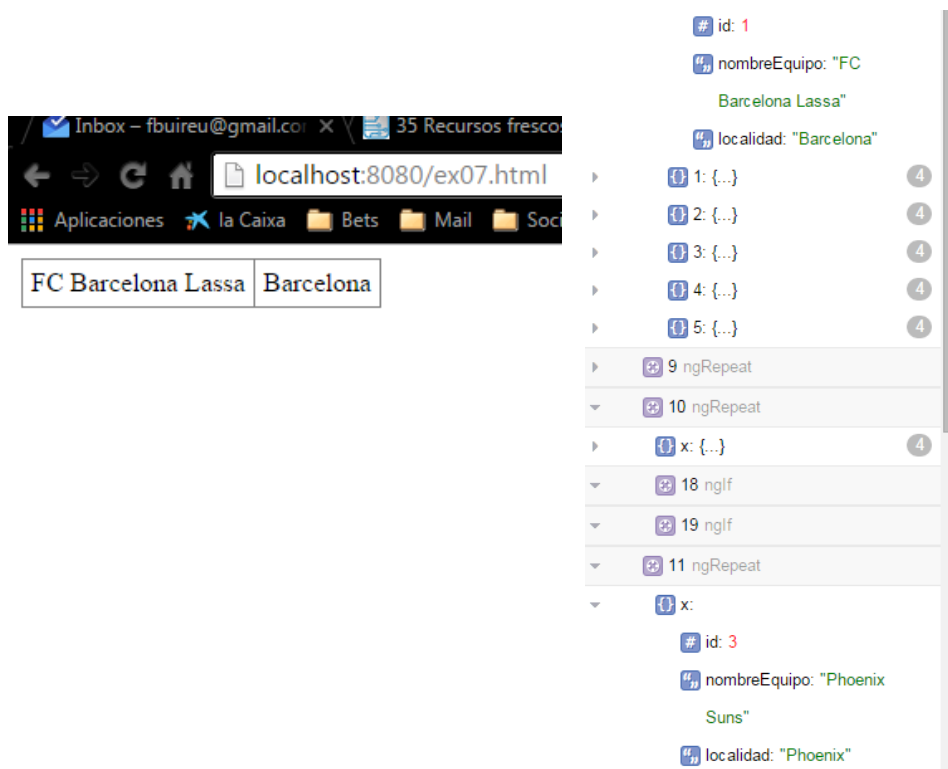
### **EXPLICACIÓN:**

En éste ejercicio se ha innovado usando la directiva `ng-if`, que añade un elemento condicional a la tabla. Cuando los elementos cumplen la condición anunciada en el elemento condicional (en éste caso para elementos pares e impares), se modifica su estilo CSS para añadir un color de fondo.

## EJERCICIO 12:

Extiende el ejercicio número 11 de manera que sólo se muestren los jugadores que hayan conseguido 50 o más canastas.

### CAPTURA:



### EXPLICACIÓN:

En éste caso, se ha realizado el ejercicio para los equipos cuya ciudad sea Barcelona. Para ello usamos la directiva *ng-show*, que se inicializa en el *tag tr*. Se ejecutará tantas veces como equipos haya. Insertamos *x.localidad* (*x* es la *key* para éste bucle) y añadimos la cláusula deseada, Barcelona para éste caso. Con el *ng-inspector*, se aprecia cómo el bucle se ejecuta de todas formas, apareciendo el resto de equipos, pero *AngularJS* los oculta, ya que no cumplen la condición en la sentencia.

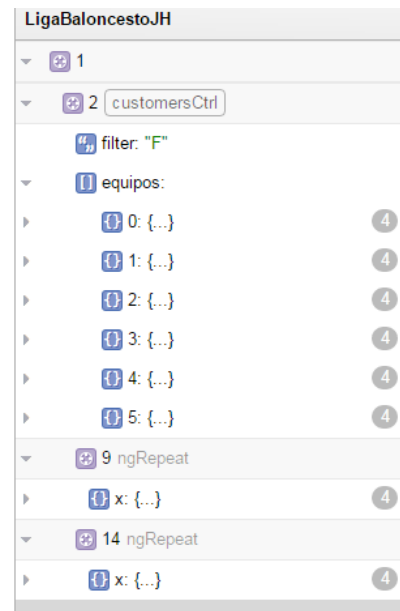
### **EJERCICIO 13:**

Extiende el ejercicio número 11, de modo que sea posible filtrar los jugadores por cualquiera de sus atributos, tal y como se enseña en la documentación oficial de AngularJS:

#### **CAPTURA:**

Introduce una letra para filtrar:

- FC Barcelona Lassa
- JSF Nanterre



#### **EXPLICACIÓN:**

Para realizar éste ejercicio se ha usado la directiva *filter*, insertada en el *tag ng-repeat*. Se Crea un input que tiene un *ng-model* determinado. Al usar el mismo parámetro del modelo en el *filter*, AngularJS es capaz de filtrar en tiempo real los resultados recogidos en la petición *HTTP*. Cómo en los ejercicios anteriores, el bucle recorre los nombres de los equipos y los muestra usando las dobles llaves *{{}}* y el *scope* media entre la vista y el controlador.

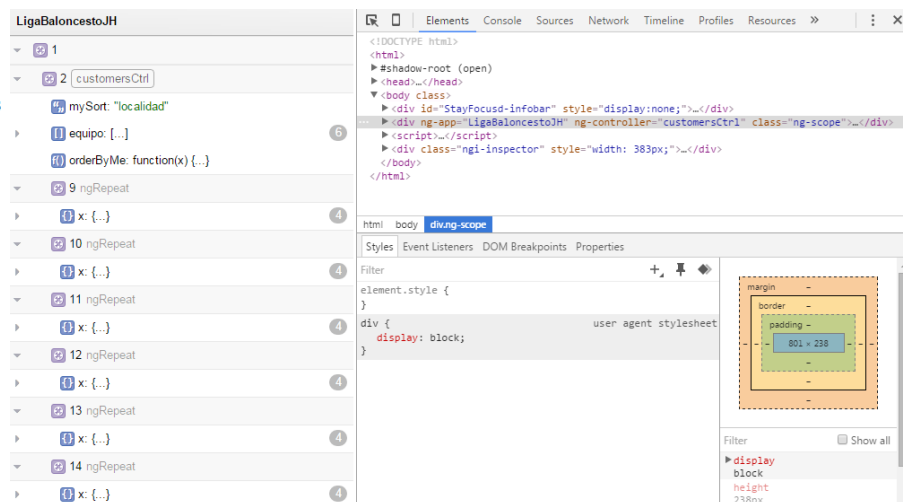
## EJERCICIO 14:

Extiende el ejercicio número 11, de manera que sea posible ordenar por diversos atributos de la clase jugador.

### CAPTURA:

Clica los headers para ordenar los campos

Nombre equipo	Localidad
DKV Joventut	Badalona
FC Barcelona Lassa	Barcelona
Real Madrid	Madrid
Club Baloncesto Málaga	Málaga
JSF Nanterre	Nanterre
Phoenix Suns	Phoenix



### EXPLICACIÓN:

Para realizar éste ejercicio se ha usado la tabla del ejercicio 10, la cual ya crea una tabla y realiza un bucle mostrando los equipos y su localidad. Además, se le ha añadido dos *tags table header* (*th*) con la función *ng-click*. En el controlador se le añade la siguiente función:

```
$scope.orderByMe = function (x) {  
    $scope.mySort = x;  
    }  
});
```

Dicho código llama a la función *orderByMe*, que está incrustada en el *ng-click*. El *mySort* realiza la función de ordenación insertado en el *tag tr* de la tabla.

### **EJERCICIO 15:**

*Desarrolla un formulario para dar de alta un jugador, tal y como se especifica en el código de ejemplo. Para este ejercicio, no es preciso que realmente se comunique con el API REST de Spring para crear el jugador en la base de datos. De manera opcional, para subir nota puedes implementar la creación realista comunicándote con el API REST de Spring investigando el código de JHipster. En este caso, es preciso que el ejercicio 15 utilice el mismo nombre de módulo que JHipster, para que se ejecute el código de AngularJS específico para temas de autenticación (y para ello, obviamente, es preciso incluir los scripts que implementan autenticación).*

### **CAPTURA:**

### **EXPLICACIÓN:**

Para realizar el ejercicio 15, se ha adaptado el código para que guarde un equipo en vez de un jugador.