26-8-2015

David Letrán González

Miguel Ángel Núñez-Romero Olmo

Documento de Diseño

Proyecto Predicción de Resultados Deportivos

[1. Diseño Front-End 2](#_Toc428810905)

[1.1. Introducción. 2](#_Toc428810906)

[1.2. Tecnología 2](#_Toc428810907)

[1.3. Arquitectura 2](#_Toc428810908)

[1.4. Implementación 3](#_Toc428810909)

[1.4.1. Modelo-Vista-Controlador 3](#_Toc428810910)

[1.4.2. Data-Binding 3](#_Toc428810911)

[1.4.3. Módulos 4](#_Toc428810912)

[1.4.3.1. Angular-Translate 4](#_Toc428810913)

[1.4.3.2. Tc-Angular-ChartJs 4](#_Toc428810914)

[1.4.3.3. FlowJS 5](#_Toc428810915)

[1.4.4. Bootstrap 5](#_Toc428810916)

[2. Diseño Back-End 6](#_Toc428810917)

[2.1. Introducción. 6](#_Toc428810918)

[2.2. Tecnología. 6](#_Toc428810919)

[2.3. Obtención de los datos. 6](#_Toc428810920)

[2.4. Predicciones del sistema. 7](#_Toc428810921)

[2.5. Tareas programables. 8](#_Toc428810922)

[2.6. Comunicación Front-End. 8](#_Toc428810923)

# Diseño Front-End

## Introducción.

Éste documento estará dirigido a los desarrolladores del sistema, no a usuarios finales ni a clientes. Contiene información del diseño de la parte correspondiente al front-end del sistema.

## Tecnología

La tecnología escogida para el desarrollo de la interfaz del sistema es AngularJS, un framework de JavaScript que implementa el patrón Modelo, Vista, Controlador (MVC). Está mantenido por Google y dispone de una amplia comunidad que ayuda a su mantenimiento y con una amplia variedad de contribuciones de usuarios. Además se utilizará Bootstrap, un framework HTML y CSS que permitirá adaptar la aplicación para usuarios que accedan desde dispositivos móviles de manera fácil y rápida, añadiendo además sencillez a la hora de crear un diseño atractivo.

## Arquitectura

La interfaz estará basada en el concepto de SPA (Single Page Application), es decir, estará concentrada en una misma página que cambiará de manera dinámica dependiendo de las interacciones del usuario sin necesidad de comunicarse en todo momento con el servidor.

Se dispondrá de una página principal que contendrá elementos comunes para toda la aplicación, contenida en ella se encontrará una zona dedicada a mostrar el contenido dinámico, es decir aquel correspondiente a cada sección de la aplicación. Los elementos comunes serán el menú superior y el pie de página, quedando todo el espacio entre ellos destinado a el contenido dinámico.

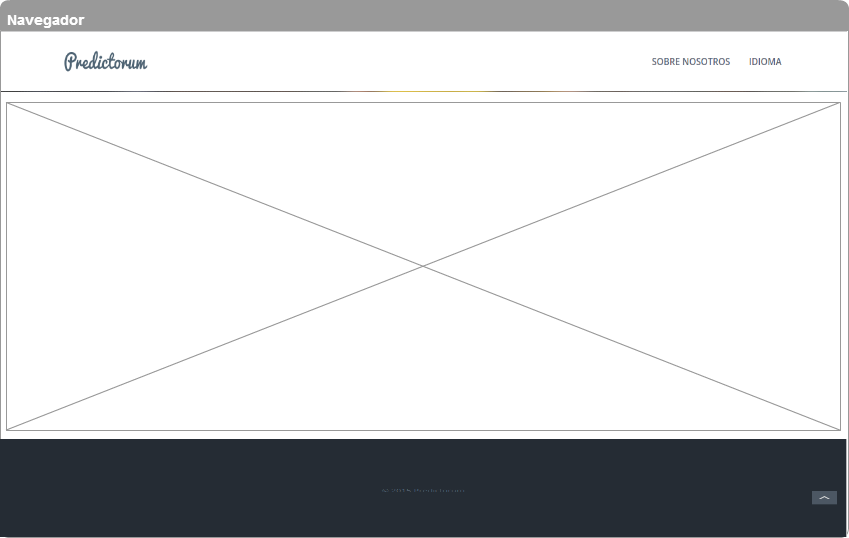


Ilustración 1: Arquitectura de la interfaz

## Implementación

### Modelo-Vista-Controlador

Para lograr el objetivo marcado se utiliza el patrón Modelo-Vista-Controlador, cada sección de la aplicación dispondrá de su propio controlador y servicio, así como de vistas parciales, las cuales se corresponderán con las distintas secciones. De esta manera cuando un usuario interactúa con la aplicación, sus acciones son recogidas por el controlador que tras tratarlas debidamente se comunica con el servicio para que este se encargue de comunicarse con el servidor en caso de que sea necesario.

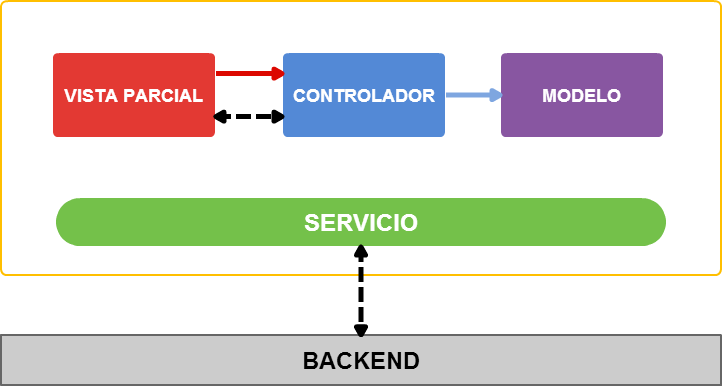


Ilustración 2: Modelo-Vista-Controlador

### Data-Binding

Esto es así gracias al llamado Data-Binding de AngularJS, el cual actualiza la vista siempre que el modelo cambia o actualiza el modelo cuando se cambia este en la vista. Así cada interacción del usuario será capturada por el controlador que podrá mostrar, ocultar o editar la información presente en el modelo sin necesidad de llamar al servidor para ello. Para implementar esto se utilizan las llamadas “directivas”, marcadores localizados en elementos del DOM que se comportan como atributos HTML y se encargan de relacionar dichos elementos con determinados comportamientos o incluso transformarlo tanto a él como a sus subelementos. Cuando la vista parcial se compila, se analizan todas las directivas y se transforman en los elementos que conforman la vista final a partir de la información contenida en el modelo.

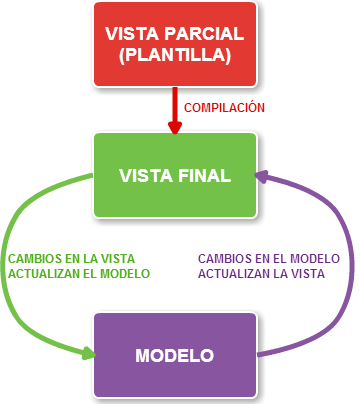


Ilustración 3: Data-Binding

La combinación de estos elementos provoca una experiencia de uso más rápida al usuario gracias a los cambios asíncronos que se producen en la vista que ahorran tiempos de carga que se añadirían si fuera necesario conectarse continuamente con el servidor.

### Módulos

AngularJS permite la agrupación de funcionalidades en módulos que pueden ser más tarde reutilizados por otras aplicaciones como si de librerías se tratasen. Esta utilidad permite que se hagan muchos aportes a la comunidad que facilitan mucho el desarrollo al utilizar los avances de otros desarrolladores. En nuestro caso nos hemos ayudados de ciertos módulos para implementar varias funcionalidades sin tener que desarrollarlas desde cero.

### Angular-Translate

Se trata de un módulo que permite implementar el cambio de idioma para la aplicación, utiliza archivos en formato JSON que contienen las claves que representan a los mensajes que queremos traducir junto con el valor asociado al idioma en concreto. Utiliza cookies para almacenar el idioma elegido por el usuario y que así no tenga que elegirlo cada vez que visita la aplicación. Permite el cambio de idioma sin necesidad de recargar la página.

### Tc-Angular-ChartJs

Consiste en un módulo que integra ChartJs que permite crear gráficos de todo tipo de manera sencilla, una funcionalidad de mucha utilidad cuando se trata de mostrar estadísticas y datos, algo muy presente en Predictorum.



Ilustración 4: tc-angular-chartjs

### FlowJS

Consiste en un módulo que permite la subida de archivos con tolerancia a fallos, se divide el archivo en varias partes y si la subida de alguna de ellas fallase se intentaría de nuevo hasta que se consiguiera. En Predictorum se utiliza para la subida de imágenes en la personalización del perfil de los usuarios, permitiendo previsualizarla antes de guardarla.

### Bootstrap

Para conseguir que la interfaz esté disponible en un formato adecuado para cualquier dispositivo independientemente del tamaño de su pantalla se ha utilizado el framework HTML Y CSS, Bootstrap. Esto se consigue mediante las clases predefinidas en Bootstrap que ayudan a definir el espacio que ocuparan los elementos dependiendo del tamaño de la pantalla del dispositivo en el que se está mostrando. Para ello se parte de que el espacio está dividido en 12 columnas, de manera que es posible definir que dos elementos ocupen una fila, teniendo 6 columnas cada uno de ellos, para pantallas grandes pero que ocupe cada uno 12 columnas, apareciendo así uno encima del otro, para una pantalla pequeña.

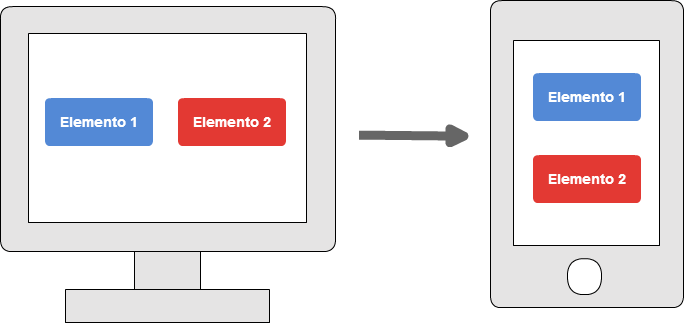


Ilustración : Bootstrap

# Diseño Back-End

## Introducción.

Éste documento estará dirigido a los desarrolladores del sistema, no a usuarios finales ni a clientes. Contiene información del diseño de la parte correspondiente al back-end del sistema.

## Tecnología.

Como framework para el desarrollo back-end se utilizará Spring. Se trata de un framework basado en Java que implementa el patrón MVC visto anteriormente en el diseño back-end. Spring junto con “MySQL” como gestor de base de datos e Hibernate como herramienta de mapeo objeto-relación (ORM) presentan un entorno de desarrollo fácil, seguro y mantenible que harán más liviana la programación de esta parte del proyecto.

## Obtención de los datos.

Para la obtención de los datos que forman la base histórica de resultados deportivos, así como para la creación del calendario de la temporada actual y la actualización del mismo en base al resultado de los partidos a lo largo del tiempo, hemos decidido utilizar la librería Jsoup.

Jsoup es una librería de Java que a través de su API nos permite extraer y manipular datos provenientes de una página web HTML accediendo al DOM y al CSS de la misma.

La página web [www.marca.com](http://www.marca.com) nos ofrece todos los datos necesarios para crear nuestra base histórica, así como ir actualizando nuestro calendario.

Un ejemplo de extracción de datos lo encontramos en el siguiente ejemplo:

Accedemos a la URL:

[www.marca.com/deporte/central-de-datos/futbol/premier-league/2014-2015/calendario.html](http://www.marca.com/deporte/central-de-datos/futbol/premier-league/2014-2015/calendario.html)

Un fragmento de código que representa encuentros futbolísticos en una determinada jornada de liga sería el siguiente:



Ilustración Fragmento de código HTML

Se pueden observar patrones que nos permitirían, accediendo por ejemplo al contenido de las clases “local”, obtener el conjunto de equipos locales que disputan estos partidos.

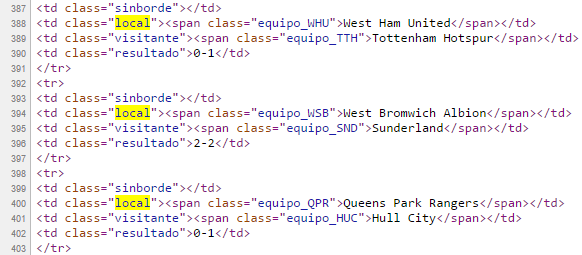


Ilustración Fragmento HTML resaltando una clase CSS.

Del mismo modo que obtenemos los equipos locales podríamos obtener todos los datos que creyésemos oportunos hasta formar el calendario de la temporada actual o nuestra base histórica, recorriendo las diferentes temporadas de cada liga, dependiendo del caso.

## Predicciones del sistema.

Para que el sistema sea capaz de realizar predicciones de forma autónoma hemos utilizado la herramienta Weka.

Weka nos proporciona un conjunto de algoritmos de aprendizaje basándonos en un conjunto de datos.

A continuación se explica sin entrar en detalle cómo le hemos dado uso a la herramienta:

**Paso 1:** Primero definimos los atributos de los que estarán compuestos los vectores que usaremos tanto para el aprendizaje como la predicción. Siendo estos: equipo local, equipo visitante, temporada, puntos equipos local, posición en la liga de los diferentes equipos, ganador del encuentro, goles del equipo local, etc.

**Paso 2:** Creamos un conjunto de entrenamiento usando nuestra base histórica de datos como vectores y como patrón el vector definido previamente.

**Paso 3:** Definimos un modelo de entrenamiento. Para este proyecto hemos utilizado uno basado en Bayes aunque sería conveniente hacer un estudio más detallado para comparar los distintos modelos.

**Paso 4:** Elegimos el atributo que queremos que aprenda entre el conjunto de atributos del vector. Un atributo a aprender sería por ejemplo el ganador de del partido, con 3 posibles valores: local, visitante o empate.

**Paso 5:** Entrenamos el conjunto de entrenamiento una vez elegido el atributo y el modelo.

**Paso 6:** Realizamos la predicción de un determinado encuentro. Esta predicción nos devolverá una lista de números en la que cada número representará la probabilidad de que ocurra el suceso referente a los posibles valores del atributo. En nuestro caso: la probabilidad de que gane el equipo local, gane el equipo visitante o haya empate.

## Tareas programables.

Para llevar a cabo la creación de predicciones por parte del sistema, así como la actualización de los resultados del calendario una vez los encuentros futbolísticos hayan tenido lugar, hemos utilizado la anotación “@Scheduled” que permite a nuestros servicios ejecutarse periódicamente.

Con esto anotación, conseguimos que los métodos en nuestros servicios se ejecuten de forma periódica sin la necesidad de que un usuario lo pida.

## Comunicación Front-End.

La comunicación con front-end se realizará a través una Resp API Json. Los controladores back-end responderán ante peticiones http del tipo GET de clientes con JSON donde irán mapeados los objetos que posteriormente se representarán en la interfaz del usuario. De igual modo, la parte cliente podrá enviar información a través de peticiones POST con JSON incrustados para que el servidor procese o guarde información.

La comunicación se realizará a través de la API implementada en la dirección:

<http://Predictorum/api/>...

Un posible ejemplo de acceso a la Api que devolvería un JSON sería:

<http://Predictorum/api/team/list>