**Introducción**

A la hora de escoger mi trabajo de fin de grado valoré distintas opciones de temáticas muy variadas, pero hubo uno que me llamó especialmente la atención. Este fue el de “Sistema de gestión de ligas de fútbol”, no sé si el motivo fue porque vivimos en un país donde el deporte que predomina es el fútbol y, por este motivo, creo que es un tema que tiene un grado de aplicación alto en la sociedad o porque he jugado al fútbol desde los cinco años, pero es difícil que encontrara un tema donde me sintiera más cómodo que en un trabajo relacionado con el fútbol. Es por este motivo por el que decidí elegir este trabajo de fin de grado.

Tras informarnos tanto mis compañeros como yo de las partes en las que se dividía este trabajo, decidí quedarme con la aplicación móvil de arbitraje para árbitros. El motivo de esta decisión fue simple, no entiendo como en pleno siglo XXI un árbitro necesita utilizar una libreta y un bolígrafo para arbitrar un partido y tampoco entiendo que, sin necesidad, tenga que rellenar un acta una vez finalice el partido en lugar de realizar estas tareas de una manera más rápida, sencilla y automatizada apoyándose en la tecnología.

Poco a poco vemos como la tecnología va apareciendo en el fútbol para ayudar al árbitro, como ha ocurrido este año con la implantación del VAR en la Liga Española. Personalmente, opino que toda ayuda que se le pueda prestar a la figura del árbitro es poca, ya que es muy complicado encontrar un partido donde no exista polémica ni donde los aficionados de ambos equipos opinen que el arbitraje ha sido el adecuado

Por otra parte, recuerdo como en la liga donde yo solía jugar teníamos que esperarnos hasta el miércoles para conocer los resultados de los otros partidos o, por el contrario, enviar a un “mensajero” a los campos donde se disputaban estos partidos para conocerlos el mismo día, un completo sinsentido. Esto se debía a que los miércoles era cuando se reunía el comité y se aceptaban las actas.

Estos motivos entre otros me motivaron para aplicar los conocimientos adquiridos durante la carrera en crear esta aplicación y, de esta manera, conocer nuevas tecnologías, adquirir más conocimientos orientados al mercado laboral y, de algún modo, dar un pequeño empujón para que el mundo del fútbol no se oponga a la tecnología y que la utilice para mejorar evitar situaciones como las expuestas anteriormente.

**Antecedentes**

Como he comentado anteriormente, en la mayoría de los casos el comité realiza todos sus trámites en papel apoyándose simplemente en herramientas tecnológicas para crear los documentos necesarios y los árbitros se encargan de recoger las incidencias de los partidos mediante una libreta y un bolígrafo y tras finalizar el partido, rellenan el acta de este de manera manual que debe ser enviado al Comité para su aprobación a posteriori.

Para observar un mejor ejemplo de esto, voy a resumir los estamentos y funciones en las que se divide la Liga Profesional de Fútbol Española para realizar esto.

**Comité Técnico de Árbitros**

Según la Real Federación Española de Fútbol, el Comité Técnico de Árbitros atiende directamente al funcionamiento del colectivo federativo de árbitros y le corresponden, con subordinación al presidente de la RFEF, el gobierno, representación y administración de las funciones atribuidas a aquellos.

Según la Real Federación Española de Fútbol, el Comité Técnico de Árbitros se encarga de:

1. Establecer los niveles de formación arbitral.
2. Clasificar técnicamente a los árbitros y proponer la adscripción a las categorías correspondientes.
3. Proponer los candidatos a árbitros de categoría internacional.
4. Aprobar las normas administrativas regulando el arbitraje.
5. Coordinar con las Federaciones de Ámbito Autonómico integradas en la RFEF los niveles de formación.
6. Designar a los colegiados en las competiciones de ámbito estatal no profesional.
7. Cualesquiera otras delegadas por la RFEF.

Además, también se encargan de las asignaciones de árbitros en los partidos donde intervengan clubes adscritos en la Liga de Fútbol Profesional junto con miembros de la Real Federación y de La Liga.

**Árbitro Principal**

El árbitro principal del partido es la figura que se debe encargar de hacer cumplir el reglamento durante este.

Estas reglas son:

1. El terreno de juego.
2. El balón.
3. Los jugadores.
4. El equipamiento de los jugadores.
5. El árbitro.
6. Los otros miembros del equipo arbitral.
7. La duración del partido.
8. El inicio y la reanudación del juego.
9. El balón en juego o fuera del juego.
10. El resultado de un partido.
11. El fuera de juego.
12. Faltas y conducta incorrecta.
13. Tiros libres.
14. El penalti.
15. El saque de banda.
16. El saque de meta.
17. El saque de esquina.

El árbitro principal es la única figura en el terreno de juego que puede tomar decisiones, aunque puede consultar estas con su equipo arbitral que, dependiendo del partido y la competición, estará formado por dos o más árbitros adicionales.

**Equipo arbitral**

El equipo arbitral suele estar formado por dos asistentes y el cuarto árbitro.

Los dos asistentes se sitúan en ambas bandas del terreno de juego y aconsejan al árbitro principal en acciones que tenga dudas o no haya podido ver con claridad, pero nunca podrán tomar decisiones de manera individual

El cuarto árbitro es la figura que suplirá al árbitro en caso de que este se lesionase y ayuda al árbitro principal en la anotación de cambios, amonestaciones y expulsiones.

En algunos partidos el equipo arbitral también se compone de quinto árbitro y/o dos árbitros asistentes adicionales.

El quinto árbitro tratará de aconsejar al árbitro principal en sus decisiones mientras que los dos árbitros asistentes adicionales se situarán en las dos áreas del terreno de juego y se encargarán de aconsejar en las jugadas del área correspondiente y en las jugadas de “goles fantasmas”.

Estas tres últimas figuras son menos habituales debido a la inclusión del “ojo de halcón” que ayuda al árbitro a decidir si un balón ha entrado o no en la portería mediante la tecnología.

Este último año, también han aparecido nuevas figuras en el equipo arbitral con la implantación del VAR (Video Assistant Referee). Estas figuras son las encargadas de visualizar las jugadas en la sala de realización, donde observan la señal de televisión y pueden ver las jugadas repetidas del partido. El árbitro se encuentra comunicado con estos asistentes mediante un auricular y puede pedir la revisión de una jugada solo en los siguientes casos:

1. Goles.
2. Penaltis.
3. Expulsiones directas.
4. Confusión de identidad.

En estos casos, los asistentes del VAR visualizarán la jugada y aconsejarán al árbitro principal sobre la decisión final.

**Objetivos**

El objetivo principal de este sistema de gestión es acercar al mundo del fútbol la tecnología de tal manera que se acorten los plazos de las gestiones y que ayude tanto a futbolistas, miembros del comité y árbitros a desempeñar sus funciones.

En el caso de la aplicación de arbitraje, los objetivos principales son los siguientes:

* El árbitro podrá tener controlados los partidos que debe arbitrar en cada momento de forma sencilla desde la aplicación.
* El árbitro podrá añadir a un partido a los jugadores que se presenten teniendo toda la información de estos en la aplicación móvil sin necesidad de que los equipos lleven siempre al campo de fútbol un cuaderno con las fichas de los jugadores de cada equipo.
* El árbitro podrá aplazar un partido directamente desde la app, sin necesidad de ningún tipo de comunicación adicional con el comité.
* El árbitro podrá notificar la no presentación de uno de los equipos directamente desde la app, sin necesidad de ningún tipo de comunicación adicional con el comité.
* El árbitro podrá controlar el tiempo del partido desde la app.
* El árbitro podrá controlar todas las incidencias que ocurren en el partido (goles, tarjetas amarillas y tarjetas rojas) sin necesidad de apuntar en una libreta quién ha cometido la incidencia ni en qué minuto se ha cometido ya que esto lo realizará automáticamente la aplicación.
* El árbitro podrá generar un acta directamente al finalizar el partido, sin necesidad de rellenarlo de manera manual y enviarlo al comité por vías distintas.

El objetivo global será, por tanto, facilitar el arbitraje al árbitro y reducir los tiempos de gestión.

Personalmente, mi objetivo es aplicar todo lo estudiado en la carrera en un entorno real y, a su vez, aprender nuevas tecnologías.

**Estado del arte**

Debido a que la realización tanto del arbitraje como de las gestiones relacionadas con este actualmente se hacen de manera escrita y manual, es complicado encontrar aplicaciones que sigan la línea de esta aplicación, pero existen algunas alternativas interesantes en el mercado. Estas son las alternativas:

**Alternativas**

**Easy Referee**

Se trata una aplicación con una interfaz muy simple donde aparecen las siguientes opciones:

* Tirar una moneda al aire para sortear el campo al inicio del partido.
* Mostrar una tarjeta amarilla donde se puede anotar un registro de todas estas.
* Mostrar una tarjeta roja donde se puede anotar un registro de todas estas.
* Un registro para anotar cualquier suceso del partido.
* Un silbato para reproducir un sonido similar al de un silbato al pulsar sobre él.
* Un cronómetro para cronometrar el partido.

En general, es una app bastante fácil de usar, algo que es necesario para los usuarios de este tipo de apps.

Por otra parte, se trata de una aplicación con muchas carencias, por ejemplo, todos los registros que se guardan en la aplicación deben ser introducidos mediante el teclado, esto implica una pérdida de tiempo bastante alta para el colegiado.

Otra característica que se echa en falta es la generación de un acta al finalizar el arbitraje de un partido.

En resumen, *Easy Referee* es una aplicación bastante intuitiva pero muy limitada en funcionalidades.

Falta imagen

**Árbitro del fútbol**

*Árbitro del fútbol* es una aplicación que nos ofrece una interfaz gráfica más agradable para el usuario que la anterior aplicación.

Permite realizar el arbitraje de un partido y visualizar los partidos que han sido ya arbitrados.

La funcionalidad que nos permite esta aplicación a la hora de arbitrar un partido es demasiado escasa. Permite mostrar tarjetas amarillas o tarjetas rojas al equipo local o visitante, añadir goles a ambos equipos y muestra la cuenta mediante un cronómetro.

A pesar de que visualmente es una aplicación más lograda que *Easy Referee*, su funcionalidad es insuficiente para cualquier árbitro ya que, por ejemplo, no puedes indicar a qué jugador muestras las tarjetas ni, como en la anterior aplicación, tampoco puedes generar un acta una vez se finaliza el partido.

Falta imagen

**Referee Pro**

Esta aplicación es, sin duda alguna, la mejor aplicación de arbitraje que existe actualmente en el mercado. Además, la razón por la que se desarrolló en el año 2012 es exactamente la misma razón que ha hecho que se desarrolle nuestra aplicación:

*“Referee PRO nace en 2012 como el Primer DRSS (Sistema de Señalización Digital, por sus siglas en inglés), la idea de digitalizar los datos oficiales del deporte desde la raíz. Así hemos ayudado a facilitar el trabajo de árbitros y organizadores de ligas con resultados increíbles.*

*Con Referee PRO ya no es necesario administrar datos de la manera tradicional; imprimir identificaciones de jugadores, tomar notas en papel y adivinar el contenido de las Actas Arbitrales, transcribir información a una computadora y perder información crucial en el proceso, así como la falta de transparencia... son cosas del pasado.”*

Según aparece en la página web de la app.

Esta aplicación se encuentra en el mercado desde 2012 y ha ido actualizándose con el paso del tiempo por lo que actualmente es una aplicación muy completa.

Las funcionalidades que esta aplicación proporciona son muy extensas y cubren totalmente cualquier necesidad de un árbitro en un partido.

Algunas desventajas que he encontrado en esta aplicación son que, al proporcionar tanta funcionalidad, la usabilidad se reduce en comparación con las anteriores apps y que, para poder usar la aplicación, es necesario realizar un registro anterior mediante el envío de un e-mail a un particular, la app mejoraría mucho si esto se pudiese realizar directamente desde la app.

**Planteamiento del problema**

A la hora de realizar la aplicación, lo primero que tuvimos en cuenta era que debíamos realizar un *back-end* conjunto ya que nuestro sistema de gestión de ligas deportivas se compone de tres *front-end* distintos. Por lo que, en primer lugar, hablaré de qué es *back-end* y *front-end.*

**Back-end**

Se denomina *back-end* a la parte del software que procesa los datos desde el *front-end*.

El *back-end* suele encargarse de la comunicación con las bases de datos de la arquitectura, de las sesiones de los usuarios en la aplicación y, en el caso del desarrollo web, de servir la página web desde un servidor. En definitiva, la parte *back-end* se encarga de la manipulación de los datos.

La parte *back-end* de nuestro proyecto sería la API REST con la que interactúa la *app*.

**Front-end**

Se denomina *front-end* a la parte del software que interactúa directamente con el usuario.

El *front-end* se encarga de recolectar datos de parte del usuario y darles el formato necesario para enviar estos al *back-end*, a su vez, también se encarga de recibir respuestas del *back-end* y exponer estas de manera entendible para el usuario.

La parte *front-end* del proyecto sería la propia *app* con la que interactúa el árbitro, la página web para el comité y la *app* para los jugadores.

Tras conocer la parte *back* y las partes *front* de nuestro proyecto, pasamos a plantearnos qué tipo de base de datos era más interesante para nuestro proyecto, si una base de datos SQL o una base de datos NoSQL y porqué.

**Bases de datos**

Se denomina bases de datos al “almacén” donde es posible guardar grandes cantidades de datos de forma organizada y que pueden ser consultados y encontrados fácilmente. En informática, las bases de datos son sistemas formados por un conjunto de datos almacenados en discos que permiten el acceso a estos de manera directa o desde otras aplicaciones.

Es extraño encontrar cualquier tipo de sistema informático que no cuente con una base de datos o más hoy en día. Por este motivo, podemos decir que las bases de datos son imprescindibles en cualquier sistema informático y nos atreveríamos a decir que, además, es la parte más importante.

Podemos diferencias dos grandes tipos de bases de datos: Relacionales (SQL) o no relacionales (NoSQL)

**Bases de datos relacionales o SQL**

Se denomina base de datos relacional a una base de datos que cumple con el modelo relacional.

El modelo relacional se trata de un modelo de datos basado en la lógica de predicados y teoría de conjuntos cuya idea fundamental es el uso de relaciones. Cada relación es una tabla donde cada fila es un registro y cada columna es un campo.

Las bases de datos relacionales tienen las siguientes características:

* No pueden existir dos tablas con el mismo nombre ni registro.
* La relación entre tablas se realiza mediante las claves primarias y foráneas.
* Las claves primarias son la clave principal de un registro dentro de una tabla.
* Las claves primarias sirven para relacionar unas tablas con otras y contienen el mismo valor que la clave primaria del registro original.

**Ventajas**

Las principales ventajas de las bases de datos relacionales son:

* Garantizan que no se produzcan registros duplicados.
* Permiten la “eliminación en cascada”.
* Favorecen la normalización.

**Desventajas**

Las principales desventajas de las bases de datos relacionales son:

* Deficiencia con datos gráficos y multimedia.
* Mala compatibilidad con bloques de texto como tipo de dato.
* Difíciles de usar con datos heterogéneos.

**Principales gestores**

Las bases de datos relacionales son el tipo de base de datos más usadas actualmente, es por esto que existen muchos gestores de bases de datos para bases de datos relacionales y, además, la mayoría de estos son mucho más famosos que los gestores de bases de datos no relacionales. Los principales gestores de bases de datos relacionales son los siguientes:

* MySQL
* Oracle
* Microsoft SQL Server

Aunque, como hemos comentado anteriormente, existen muchos más.

**¿Cuándo usar una base de datos no relacional?**

* Cuando se deben minimizar al máximo los posibles fallos.
* Cuando es necesario estructurar la información.
* Cuando sepamos que la información va a mantener una estructura consistente a lo largo del tiempo.

**Bases de no datos relacionales o NoSQL**

Una base de datos no relacional es cualquier base de datos que no siga el modelo de datos relacional. Estas bases de datos son optimizadas para datos sin esquema y de desempeño escalable.

Existen distintos tipos de bases de datos relacionales según el modelo de datos utilizado. Los principales son los siguientes:

* Clave-valor.
* Gráficos.
* Documentos.

**Ventajas**

Las principales ventajas de las bases de datos no relacionales son las siguientes:

* Son más escalables que bases de datos SQL.
* La velocidad de respuesta suele ser mayor que las bases de datos relacionales.
* No necesitan ningún tipo de esquema.
* Permiten distribuir gran cantidad de datos.

**Desventajas**

Las principales desventajas de las bases de datos no relacionales son las siguientes:

* Falta de estructuración en los datos.
* Problemas de incompatibilidad con instrucciones SQL.
* Falta de estandarización.
* Herramientas de administración pobres.

**Principales gestores**

A medida que avanza el tiempo, las bases de datos no relacionales van tomando un papel más importante en el mundo de la informática y cada vez se encuentran en más proyectos. Esto es debido principalmente al aumento exponencial de datos que se produce año tras año y al *Big Data*, ya que esta tecnología es mucho más escalable que las bases de datos relacionales, como ya hemos comentado con anterioridad.

Debido al crecimiento, cada vez son más los gestores de bases de datos no relacionales que aparecen. Los principales son los siguientes:

* MongoDB
* Cassandra
* DynamoDB
* Redis

Aunque existen muchas más, consideramos estas como las más importantes debido a la popularidad que tienen en la actualidad.

**¿Cuándo usar una base de datos no relacional?**

* Cuando los datos no se encuentran estructurados, es decir, con datos heterogéneos.
* Cuando se debe procesar una cantidad muy grande de datos.
* Cuando el tiempo de respuesta es crucial.
* Cuando los datos van a crecer a altas velocidades.

**Tabla comparativa: SQL vs NoSQL**

|  |  |  |
| --- | --- | --- |
|  | Bases de datos relacional | Bases de datos no relacional |
| Modelo de datos | El modelo relacional normaliza los datos en tablas conformadas por filas y columnas. Un esquema define estrictamente las tablas, las filas, las columnas, los índices, las relaciones entre las tablas y otros elementos de las bases de datos. | Las bases de datos NoSQL no suelen contener un esquema. Se suele utilizar una clave de partición para recuperar valores, conjuntos de columnas o documentos JSON o XML semiestructurados, así como otros documentos que contengan atributos de elementos relacionados. |
| Desempeño | Normalmente, el desempeño depende del subsistema de disco. Es necesaria la optimización de consultas, índices y estructura de tabla para lograr el máximo desempeño. | El desempeño es, por lo general, dependiente del tamaño del clúster de hardware subyacente, la latencia de red y la aplicación que efectúa la llamada. |
| Escalado | Lo más sencillo es ampliar la escala con un hardware más rápido. Se requieren inversiones adicionales para que las tablas relacionales abarquen un sistema distribuido. | Está diseñada para escalar horizontalmente con clústeres distribuidos de hardware de bajo costo a fin de aumentar el nivel de procesamiento sin incrementar la latencia. |
| API | Solicita almacenar y recuperar datos que están comunicados mediante consultas que se ajustan a un lenguaje de consulta estructurado (SQL). | Las API basadas en objetos permiten a los desarrolladores almacenar y recuperar fácilmente estructuras de datos en memoria. Las claves de partición permiten que las aplicaciones busquen pares de clave-valor, conjuntos de columnas o documentos semiestructurados que contengan atributos y objetos de aplicación serializados. |
| Herramientas | Las bases de datos SQL normalmente ofrecen un amplio conjunto de herramientas que simplifican el desarrollo de aplicaciones de base de datos. | Las bases de datos NoSQL suelen ofrecer herramientas para administrar los clústeres y el escalado. Las aplicaciones representan la interfaz primaria de los datos subyacentes. |

Una vez conocimos las distintas alternativas para nuestra base de datos decidimos decantarnos por una base de datos NoSQL y, en particular, el gestor MongoDB. Para esto decidimos utilizar mLab.

**mLab**

mLab es un servicio de base de datos en la nube totalmente administrado que ofrece aprovisionamiento y escalados automáticos de bases de datos MongoDB, respaldo y recuperación, monitoreo y alerta 24/7, herramientas de administración basadas en la web y soporte experto. Este servicio se apoya en servicios en la nube de Amazon, Google o Microsoft.

En nuestro caso nuestra base de datos MongoDB en mLab usa Amazon Web Service como proveedor.

A continuación, los tres miembros que desarrollamos conjuntamente la parte *back* del proyecto decidimos utilizar un servicio web que comunicara la base de datos con nuestras aplicaciones. Para ello decidimos usar un servicio REST desarrollado en Java y apoyado en el Framework Spring Boot.

**API REST**

El término REST (Representational State Transfer) se originó en el año 2000, descrito en la tesis de Roy Fielding, padre de la especificación HTTP. Un servicio REST no es una arquitectura software, sino un conjunto de restricciones con las que podemos crear un estilo de arquitectura software, la cual podremos usar para crear aplicaciones web respetando HTTP.

Según Fielding las restricciones que definen a un sistema RESTful serían:

* **Cliente-servidor**:esta restricción mantiene al cliente y al servidor débilmente acoplados. Esto quiere decir que el cliente no necesita conocer los detalles de implementación del servidor y el servidor se “despreocupa” de cómo son usados los datos que envía al cliente.
* **Sin estado**: aquí decimos que cada petición que recibe el servidor debería ser independiente, es decir, no es necesario mantener sesiones.
* **Cacheable**: debe admitir un sistema de almacenamiento en caché. La infraestructura de red debe soportar una caché de varios niveles. Este almacenamiento evitará repetir varias conexiones entre el servidor y el cliente para recuperar un mismo recurso.
* **Interfaz uniforme**: define una interfaz genérica para administrar cada interacción que se produzca entre el cliente y el servidor de manera uniforme, lo cual simplifica y separa la arquitectura. Esta restricción indica que cada recurso del servicio REST debe tener una única dirección, “URI”.
* **Sistema de capas**: el servidor puede disponer de varias capas para su implementación. Esto ayuda a mejorar la escalabilidad, el rendimiento y la seguridad.
* **Operaciones**: las operaciones más importantes que nos permitirán manipular los recursos son cuatro: GET para consultar y leer, POST para crear, PUT para editar y DELETE para eliminar.

El *framework* elegido para crear nuestra API REST es Spring Boot. En primer lugar, voy a explicar brevemente a qué se denomina *framework* en informática y después hablaré de *Spring Boot*.

**Framework**

Un *framework*, entorno de trabajo o marco de trabajo es un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar.

En el aspecto del software, un *framework* se trata de una estructura conceptual y tecnológica que ofrece recursos al programador a la hora de la organización y el desarrollo del software. Resumiendo, podríamos definir un *framework* como el esqueleto en un desarrollo software. En ocasiones, los *frameworks* van ligados a lenguajes de programación concretos, pero existen *frameworks* que no van asociados a ningún lenguaje de programación en particular.

El uso de *frameworks* suele proporcionar varias ventajas al programador, como no necesitar el planteamiento de una estructura global ya que esta ya se encuentra marcada por el *framework* o facilitar la colaboración y su mantenimiento en el futuro gracias a la estandarización servida por el *framework*. Además, existen componentes creados explícitamente para *frameworks* concretos, lo que facilita el desarrollo.

En nuestro proyecto hemos utilizado distintos *frameworks* como *Ionic, Spring o Hibernate.*

**SPRING FRAMEWORK**

Spring es un framework para el desarrollo de aplicaciones y contenedor de inversión de control, de código abierto para la plataforma Java. Si bien las características fundamentales de Spring Framework pueden ser usadas en cualquier aplicación desarrollada en Java, existen variadas extensiones para la construcción de aplicaciones web sobre la plataforma Java EE.

Spring Framework comprende diversos módulos que proveen un rango de servicios:

* **Contenedor de inversión de control**: permite la configuración de los componentes de aplicación y la administración del ciclo de vida de los objetos Java, se lleva a cabo principalmente a través de la inyección de dependencias.
* **Programación orientada a aspectos**: habilita la implementación de rutinas transversales.
* **Acceso a datos**:se trabaja con RDBMS (Relational Database Management System) en la plataforma jJva, usando Java Database Connectivity y herramientas de Mapeo objeto relacional con bases de datos NoSQL.
* **Gestión de transacciones**: unifica distintas APIs de gestión y coordina las transacciones para los objetos Java.
* **Modelo vista-controlador**: Un *framework* basado en HTTP y servlets, que provee herramientas para la extensión y personalización de aplicaciones web y servicios web REST.
* **Framework de acceso remoto:** Permite la importación y exportación estilo RPC, de objetos Java a través de redes que soporten RMI, CORBA y protocolos basados en HTTP incluyendo servicios web.
* **Convención sobre Configuración**: el módulo Spring Roo ofrece una solución rápida para el desarrollo de aplicaciones basadas en Spring Framework, privilegiando la simplicidad sin perder flexibilidad.
* **Procesamiento por lotes**: un framework para procesamiento de mucho volumen que como características incluye funciones de registro/trazado, manejo de transacciones, estadísticas de procesamiento de tareas, reinicio de tareas, y manejo de recursos.
* **Autenticación y Autorización**: procesos de seguridad configurables que soportan un rango de estándares, protocolos, herramientas y prácticas a través del subproyecto Spring Security.
* **Administración Remota**: Configuración de visibilidad y gestión de objetos Java para la configuración local o remota vía JMX (Java Management Extensions).
* **Mensajes**: Registro configurable de objetos receptores de mensajes, para el consumo transparente desde la a través de JMS (Java Message Service), una mejora del envío de mensajes sobre las API JMS estándar.
* **Testing**: Soporte de clases para desarrollo de unidades de prueba e integración.

El corazón de Spring Framework es su Contenedor de inversión de control (IoC). Su trabajo es instanciar, inicializar y conectar objetos de la aplicación, además de proveer una serie de características adicionales disponibles en Spring a través del tiempo de vida de los objetos.

Los objetos creados y gestionados por el Contenedor se denominan objetos gestionados o beans.

Los objetos pueden ser obtenidos por búsqueda de dependencias o por inyección de dependencias. ​

Búsqueda de dependencias es un modelo donde se pide al objeto contenedor un objeto con un nombre específico o de un tipo específico.

Inyección de dependencias es un modelo en el que el contenedor pasa objetos por nombre a otros objetos, ya sea a través de métodos constructores, propiedades, o métodos de la fábrica.

A la hora de alojar la API REST en la nube se nos presentaron distintas opciones: Amazon Web Service, Microsoft Azure y Google Cloud entre otras.

**Amazon Web Services vs Microsoft Azure vs Google Cloud**

A continuación, realizaré una breve comparación entre los tres proveedores de servicios en la nube en distintas facetas.

**Precio**

En el aspecto económico Amazon Web Service es más competitivo respecto a los otros dos proveedores. Además, ofrece capas gratuitas limitadas para probar sus servicios y hasta un crédito de 100 USD para estudiantes y 200 USD para profesores entre otros servicios gratuitos para estos.

**Características**

En este aspecto cada proveedor es mejor o peor dependiendo de los servicios que se necesiten.

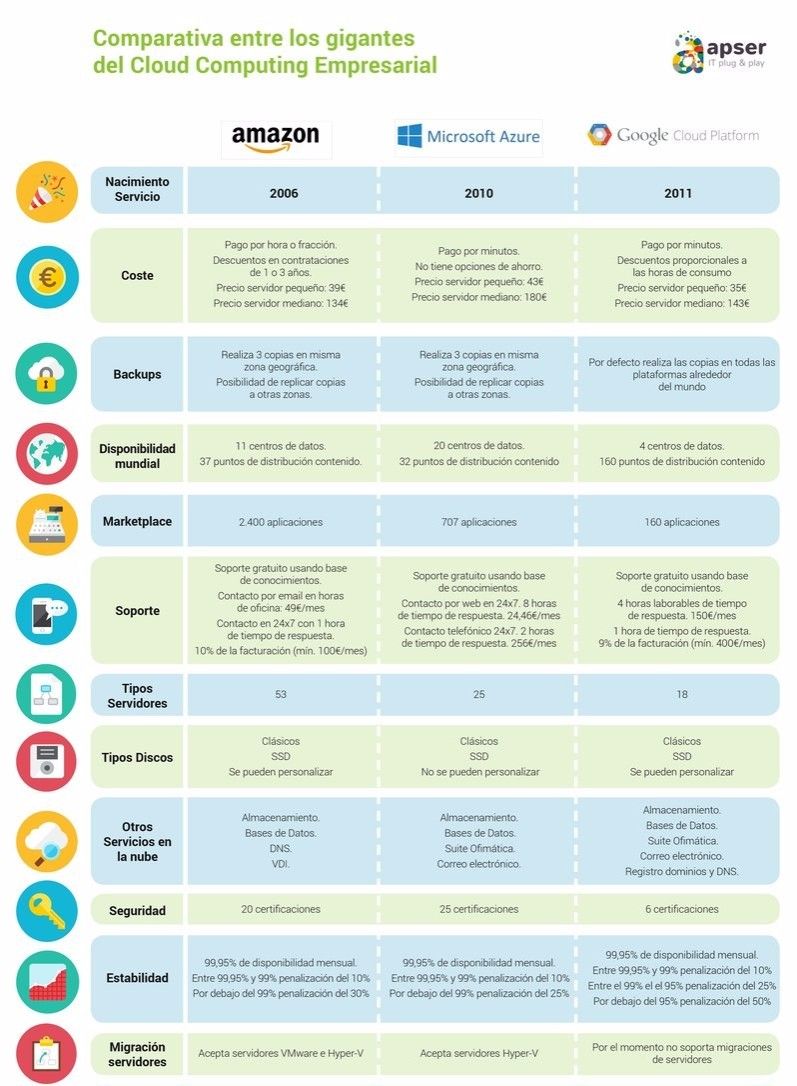
El catálogo de Amazon Web Service es el más amplio de los tres, Microsoft Azure también cuenta con un gran catálogo y adapta sus servicios según el número de usuarios. Además, Microsoft Azure cuenta con varias certificaciones interesantes, más certificaciones que ningún otro proveedor según ellos mismos.

Google Cloud se encuentra orientado mucho más a servicios para empresas que a servicios personales, lo que se denomina *Enterprise Computing.*

**Implementación**

En esta faceta, los tres proveedores ofrecen webs sencillas para implementar sus servicios, además de documentación de cómo realizarlo, por lo que la elección de uno u otro en esta faceta es algo más personal.

A continuación, se muestra una tabla comparativa entre los tres proveedores.



Nuestra elección fue Amazon Web Service ya que su capa gratuita cubría nuestras necesidades y, además, podíamos aprovechar su oferta para estudiantes de la que hablé anteriormente.

Finalmente, la siguiente decisión era cómo realizar la aplicación móvil para el arbitraje. En este caso se presentaban tres opciones principalmente: una aplicación nativa, una aplicación web y una aplicación híbrida.

A continuación, comentaré las ventajas y desventajas de todas ellas.

**Aplicación nativa**

Se denomina aplicación nativa a las aplicaciones que se desarrollan específicamente para cada sistema operativo: *Android, iOS o Windows Phone.*

Estas aplicaciones deben estar realizadas en un lenguaje u otro dependiendo de para qué sistema operativo se haya realizado. En el caso de *Android*, la aplicación se desarrolla usando el lenguaje *Java,* en el caso de *iOS*, la aplicación se desarrollo usando el lenguaje *Objetive-C* o *Swift* y en el caso de *Windows Phone*, la aplicación se desarrolla usando el lenguaje *.NET*.

**Ventajas**

Las ventajas de realizar aplicaciones nativas son:

* **Máximo rendimiento en el dispositivo.** Al conocer todas las funciones posibles del sistema operativo es posible optimizar el rendimiento para la aplicación
* **Mejor experiencia de usuario.** La optimización de las funcionalidades de la aplicación puede implicar una mejor navegación y un uso más sencillo y agradable para el usuario.
* **Disponibilidad en las App Store.** Una de las normas para que una app pueda estar en su correspondiente app store es que sea nativa. Es cierto que también se permite subir aplicaciones que contienen únicamente una parte nativa y otra web, pero la app store proporciona mayor visibilidad a las aplicaciones nativas.

**Desventajas**

Las desventajas de realizar aplicaciones nativas son:

* **Variedad de funcionalidades.** Dependiendo del sistema operativo del dispositivo, la aplicación contará con unas u otras funcionalidades. Esto puede suponer un problema a la hora de desarrollar la misma aplicación para distintos sistemas operativos, ya que es probable que no compartan todas las funcionalidades.
* **Más costosas.** El precio que se debe pagar para el desarrollo de aplicaciones nativas es mayor que para otros tipos de aplicaciones.
* **Código personalizado y exclusivo.** Todo el código generado para la aplicación no podrá ser reutilizable para otros sistemas operativos.

**Aplicación web**

Las aplicaciones web son la opción más sencilla y económica para crear una aplicación para móvil debido a que solo es necesario crear una sola aplicación.

Estas aplicaciones se basan en la creación de una página web *responsive*, esto es una página web que se adapta al dispositivo donde se muestra.

**Ventajas**

Las ventajas de las aplicaciones web son las siguientes:

* **Baratas.** Realizar aplicaciones web es la alternativa más barata ya que únicamente es necesario desarrollar la aplicación una vez.
* **Menor complejidad.** Para el desarrollador es más fácil de realizar ya que no necesita aprender distintos lenguajes de programación.
* **Multiplataforma.** La aplicación se podrá usar en cualquier dispositivo independientemente del sistema operativo de este.

**Desventajas**

Las desventajas de las aplicaciones web son las siguientes:

* **Peor rendimiento.** Las aplicaciones web tienen un peor rendimiento debido a que no aprovechan el potencial del sistema operativo del dispositivo.
* **Peor experiencia de usuario.** El desconocimiento de las características del dispositivo implica una peor experiencia en el uso de este.
* **Menor seguridad.** La seguridad de la aplicación depende de la seguridad del navegador.

**Aplicación híbrida**

Una aplicación híbrida, como su propio nombre indica, se trata de una aplicación que reúne características tanto de las aplicaciones nativas como de las aplicaciones web.

Este tipo de aplicación permite utilizar los estándares para el desarrollo de las aplicaciones web y aprovecha funcionalidades nativas del dispositivo.

**Ventajas**

Las ventajas de las aplicaciones híbridas son las siguientes:

* **Menor coste**. Las aplicaciones híbridas implican un coste menor que las aplicaciones nativas.
* **Mejor experiencia de usuario**. Las aplicaciones híbridas proporcionan una mejor experiencia de uso al usuario debido a que contiene funcionalidades nativas de los dispositivos donde se usa.
* **Multiplataforma.** Las aplicaciones híbridas se pueden usar en cualquier dispositivo, aprovechando funcionalidades específicas de cada uno de ellos dependiendo de su sistema operativo.

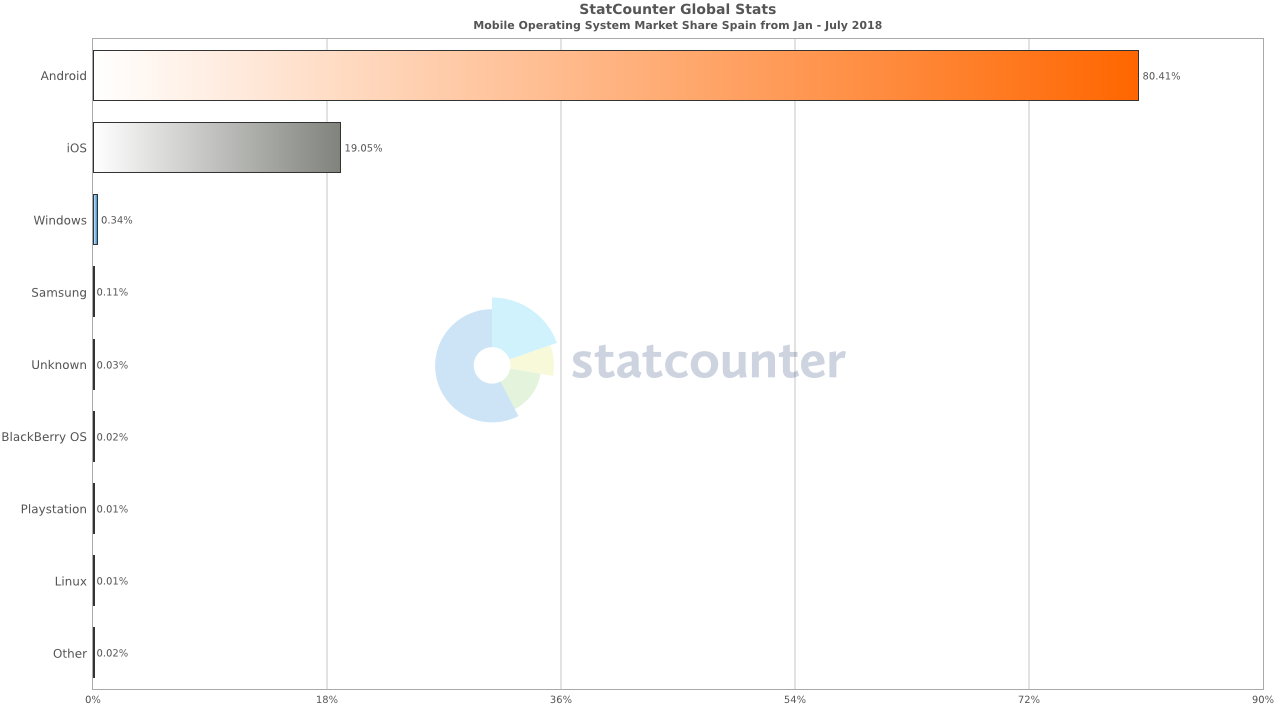
**Desventajas**

Las desventajas de las aplicaciones híbridas son las siguientes:

* **Mayor complejidad.** Para el desarrollador resultan más difíciles de desarrollar debido a las diferentes funcionalidades nativas de cada sistema operativo que tiene que tener en cuenta.
* **Rendimiento ligeramente inferior.** Las aplicaciones híbridas tienen un rendimiento peor que las aplicaciones nativas debido a que cada página debe ser renderizada desde el servidor.

Tras valorar las tres opciones, me decanté por realizar una aplicación híbrida ya que no quería perder mucho rendimiento respecto a una aplicación nativa y quería que la aplicación pudiera ser usada en cualquier sistema operativo y que, por tanto, el sistema operativo no fuera una restricción para el uso de la aplicación.

En el siguiente gráfico podemos ver el porcentaje de dispositivos móviles de distintos sistemas operativos en España durante el año actual.



Para la realización de esta aplicación híbrida me apoyé en el *framework* de *Ionic.* A continuación, explicaré sus características principales.

**Ionic Framework**

*Ionic* es una framework gratuito y open source, para el desarrollo de aplicaciones híbridas basadas en HTML5, CSS y JS. Está construido con Sass y optimizado con Angular4. Esta tecnología trabaja una capa por encima de [*Apache Cordova*](#Cordova)*.*

Las principales características de *Ionic* son las siguientes:

* **Alto rendimiento:** *Ionic* está construido para ser rápido gracias a la mínima manipulación que realiza sobre el DOM, sin usar *jQuery* y con aceleraciones de transición por hardware.
* **Uso de Angular4:** *Ionic* utiliza *Angular4* con el fin de crear un marco más adecuado para desarrollar aplicaciones ricas y robustas. *Ionic* no sólo se ve bien, sino que su arquitectura central es robusta y seria para el desarrollo de aplicaciones. Trabaja perfectamente con *Angular4*.
* **Centro nativo:** *Ionic* se inspira en las SDK de desarrollo móviles nativos más populares, por lo que es fácil de entender para cualquier persona que ha desarrollado una aplicación nativa para *iOS* o *Android*.
* **Diseño atractivo:** *Ionic* está pensado para trabajar con cualquier dispositivo móvil actual por lo que contiene muchos componentes usados en móviles como tipografías o elementos interactivos entre otros.
* **Uso de *CLI****:* Gracias a *CLI* mediante pocos comandos es posible crear un proyecto, compilarlo para distintas plataformas y probarlo.
* **Ionic Creator:** *Ionic* proporciona una interfaz gráfica para poder crear visualmente aplicaciones sin necesidad de escribir ninguna línea de código. De esta manera, Ionic nos permite diseñar nuestras aplicaciones solo con arrastrar y soltar botones, *inputs* y otros elementos.
* **Ionic DevApp:** *Ionic* creó esta aplicación móvil para ayudar a los desarrolladores. Esta aplicación puede previsualizar los proyectos mediante un simple comando si tanto el proyecto como el dispositivo se encuentran conectados en la misma red. Esta aplicación es muy útil para probar las aplicaciones ya que es mucho más rápido que emular en un ordenador o emular en el dispositivo mediante una conexión física con el ordenador.

Falta imagen

**Apache Cordova**

*Apache Cordova*, anteriormente conocido como *Phonegap* se trata *framework* desarrollo de aplicaciones móviles creado en 2009. Este *framework* permite al desarrollador crear aplicaciones móviles usando *HTML5* y *CSS* y acceder a las funciones nativas de cada sistema usando *Javascript.*

Las aplicaciones de este *framework* se basan en un archivo común *config.xml* que proporciona información acerca de la aplicación, especifica los parámetros y su funcionamiento.

*Apache Cordova* proporciona *plugins* y componentes nativos. Estos se pueden invocar mediante código *Javascript* y acceder a sus funcionalidades independientemente del sistema operativo donde se ejecute la aplicación. *Apache Cordova* permite crear tus propios *plugins* y compartirlos con la comunidad.

Existen dos vías distintas para el desarrollo de aplicaciones con *Apache Cordova:*

* **Aplicación multiplataforma mediante *CLI*:** *CLI* es la interfaz de línea de comandos que proporciona *Apache,* para utilizarla se deber poder usar *node* y *npm* desde el sistema. Mediante esta interfaz se pueden crear proyectos con tan solo un comando, así como añadir plataformas y muchas más funciones, de manera simple y cómoda. Esta vía de desarrollo es la recomendada si se pretende realizar un desarrollo multiplataforma.
* **Aplicación centrada en una plataforma.** Si se desea realizar una aplicación para una única plataforma se debe utilizar el *SDK (Software Development Kit)* proporcionado por la plataforma específica.

Este *framework* es la base de *Ionic Framework,* el *framework* utilizado para la realización de las dos aplicaciones móviles del proyecto.

**Diseño e implementación**

A la hora de implementar nuestro sistema se presentaron distintas metodologías posibles, al tratarse de un proyecto que contiene una parte realizada por tres personas y aplicaciones dependientes entre sí.

**Metodología de desarrollo software**

Se denomina metodología del desarrollo software al marco de trabajo utilizado para estructurar, planificar y controlar el proceso de desarrollo de software.

El objetivo de estas es presentar un conjunto de técnicas tradicionales y modernas de modelado de sistemas que permitan desarrollar software de calidad.

Tras esta definición es posible percatarse de que existen muchas metodologías de desarrollo software y muy diversas entre sí.

Las metodologías pueden clasificarse en dos grandes grupos: Metodologías tradicionales y metodologías ágiles

**Metodologías de desarrollo software tradicionales**

Las metodologías de desarrollo software tradicionales tienen este nombre debido a que son las metodologías más antiguas del sector, pero, aún así, son las que aún en la actualidad tienen mayor peso en el sector.

Dentro de esta clasificación podemos encontrar distintas metodologías.

**Metodología en cascada**

Al hablar de la metodología de desarrollo en cascada podemos estar hablando de la metodología de desarrollo software más antigua.

Este modelo se basa fundamentalmente en que no se puede avanzar en el proceso hasta que el paso anterior se haya completado.

Esta metodología tiene las siguientes fases:

1. **Análisis de requisitos.** En esta fase el equipo de desarrollo documenta todas las peticiones del cliente para su posterior implementación. Es muy importante tener en cuenta que no hay marcha atrás.
2. **Diseño del sistema.** Durante esta fase se elaborará la estructura del sistema y se determinarán las especificaciones de cada módulo del sistema.
3. **Diseño del programa.** En esta fase no se realiza ningún tipo de codificación, simplemente se comienzan a realizar los algoritmos en papel que se codificarán en la siguiente fase.
4. **Codificación.** Se trata de la fase más técnica de todas. Durante esta fase se codificará todo el proyecto en el lenguaje de programación que anteriormente se haya decidido.
5. **Pruebas.** Esta fase final de desarrollo involucra todo tipo de pruebas hacia el proyecto para cerciorarse de que el desarrollo es correcto antes de empezar a utilizarse.
6. **Verificación.** Esta fase es similar a la anterior, la diferencia es que quien realiza las pruebas es ya el cliente. Si se realizó una buena fase anterior, esta fase debería ser poco significante.
7. **Mantenimiento.** Probablemente esta sea la fase más crítica, ya que cualquier desarrollo software necesita evolucionar con el tiempo o sino acabará muriendo. En esta fase se incluyen tanto arreglos en la aplicación como nuevos evolutivos.

**Metodología de prototipos**

Esta metodología consiste básicamente en la recogida de requisitos y necesidades del cliente y, en torno a estas, la realización de crear de manera rápida un prototipo para poder mostrar al cliente y, a partir de este prototipo, continuar con el trabajo.

Las fases de esta metodología son similares a las de metodología en cascada, la diferencia es que estas son mucho más rápidas y que existe una retroalimentación con el cliente para ir trabajando en etapas, algo que la metodología en cascada no permite.

En este tipo de metodología es muy importante la involucración del cliente en el proyecto ya que el desarrollo de este esta constantemente dependiendo de las necesidades del cliente.

**Metodología incremental o iterativa**

Esta metodología es, fundamentalmente, una mezcla entre las dos metodologías anteriores. Básicamente consiste en completar varias iteraciones de todas las fases de la metodología en cascada, pero sin finalizar ninguna de estas, consiguiendo de esta manera la evolución del producto al permitir al cliente añadir nuevas especificaciones al producto después de realizar cada iteración.

**Metodología en espiral**

Esta metodología también se trata de una fusión entre la metodología en cascada y la metodología de prototipos, pero añade a estas una fase de gestión de riesgos, una fase que no existe en ninguna de las metodologías anteriores.

En esta metodología se van realizando las fases del modelo en cascada, pero en este caso no llevan el orden establecido ni son todas de obligado cumplimiento.

El objetivo principal de esta metodología de desarrollo se basa en la minimización de riesgos a lo largo del desarrollo para buscar la máxima seguridad.

Es posible diferenciar cuatro grandes fases en la metodología en espiral:

1. Determinación del objetivo. Se obtiene una planificación inicial.
2. Análisis de riesgos. Se tiene en cuenta todo lo que pueda perjudicar al proyecto.
3. Desarrollar, validar y probar. Esta fase dependerá del análisis de riesgos y se llevarán a cabo las tres tareas.
4. Planificación. Se planifica el siguiente ciclo del proyecto.

Todas estas fases se realizarán tantas veces como sean necesarias en el proyecto.

**Metodologías de desarrollo software ágiles**

Ha medida que ha avanzado el tiempo, han ido apareciendo nuevas metodologías de desarrollo del software. Desde principios de siglo, se han desarrollado las metodologías ágiles. Estas metodologías se basan principalmente en el trabajo mediante un número menor de documentación que en otras metodologías tradicionales.

Todas las metodologías ágiles se basan en un documento que reúne todos los principios que una metodología ágil debe seguir: el manifiesto ágil.

**Manifiesto ágil**

El manifiesto ágil es el documento donde se reúnen los principios básicos de las metodologías de desarrollo ágiles. Estos principios se pueden resumir en los siguientes puntos:

* **Equipo**. El desarrollo software se trata de una tarea humana, por lo que el trabajo del equipo es fundamental en este desarrollo. Crear un buen equipo de trabajo y un buen entorno es un principio básico en el desarrollo software.
* **Software funcional**. La documentación en los proyectos debe ser reducida al máximo, solo se debe documentar lo imprescindible. De esta manera, el costo de tiempo para documentar disminuye notablemente.
* **Cliente**. En las metodologías ágiles, el cliente es una pieza fundamental a la hora del desarrollo. La colaboración del cliente debe existir desde el principio hasta el final del proyecto.
* **Cambio**. Todo desarrollo software es, y debe ser siempre susceptible al cambio. Es imprescindible que se vayan produciendo cambios durante el desarrollo del software.

A continuación, se encuentran las principales metodologías de desarrollo software ágiles que siguen los principios del manifiesto ágil.

**Metodología Kanban**

La metodología Kanban se trata de una metodología japonesa, esta metodología se basa en fundamentalmente en el etiquetado de los procesos.

Se trata de una metodología muy simple de implementar. Sus principios básicos son:

* **Calidad.** La metodología Kanban antepone la calidad a la velocidad. Es muy importante desarrollar un producto de calidad desde el comienzo.
* **Esfuerzo necesario.** En esta metodología se debe omitir todo trabajo que no sea necesario. Solo lo necesario y requerido es lo que se debe hacer, de esta manera se ofrece una calidad mayor de producto y se optimizan costos y tiempo.
* **Adaptabilidad.** La metodología Kanban no es únicamente una metodología de desarrollo software, también se puede aplicar a otros sectores.
* **Flexibilidad.** En el desarrollo mediante la metodología Kanban no es necesario seguir una línea de trabajo, por lo que se pueden priorizar las tareas y solucionar imprevistos de manera más dinámica.

A la hora de diseñar un tablero kanban para añadir las etiquetas de los procesos, se deben seguir los siguientes pasos:

1. Crear el tablero y seccionar mediante columnas cada proceso. Es muy común el uso de tres columnas: “To do”, “Doing” y “Done”.
2. Se debe asignar una estimación de horas a cada tarea.
3. Las tareas deben seguir el orden, por lo que no se hace una tarea hasta que la de arriba del tablero ya se haya realizado. De esta manera se priorizan tareas.
4. Es muy recomendable establecer reglas de desarrollo en otra de las columnas para que todo el equipo las siga a la hora de realizar procesos.

**Metodología Extreme Programming**

La metodología *Extreme Programming* (XP) es una de las metodologías ágiles más famosa actualmente, esto es debido a la gran capacidad de adaptación frente a imprevistos que posee.

Esta metodología no plantea una recogida inicial de requisitos, sino que recomienda seguir el desarrollo según van apareciendo los requisitos sobre la marcha.

Los valores fundamentales de la metodología XP son los siguientes:

* **Comunicación.** La comunicación con el cliente es esencial en el desarrollo del software. De la misma manera, el equipo debe tener una buena comunicación para avanzar correctamente en el proyecto.
* **Simplicidad.** Se trata del principal valor de la metodología *Extreme Programming*, es necesario realizar las tareas de manera simple para dotar de velocidad al proyecto. Para esto, es muy importante codificar de manera limpia y documentar únicamente sobre este código fuente para ahorrar tiempo.
* **Retroalimentación.** El cliente se encuentra activo en el proyecto, de tal manera que durante el desarrollo de este se van corrigiendo, agregando o eliminando facetas. Esto facilita la retroalimentación del proyecto.
* **Pruebas.** La inclusión de pruebas unitarias es fundamental en esta metodología ya que facilita el desarrollo y ayuda con el punto anterior.

**Metodología Scrum**

Esta metodología ha sido la elegida por el equipo de desarrollo de este trabajo de fin de grado. Además, hemos incluido algunas ideas de otras metodologías como el tablero *Kanban.*

La metodología Scrum se trata de una metodología en la que se aplican un conjunto de buenas prácticas para trabajar colaborativamente en equipo. Está especialmente diseñado para proyectos en los que se necesita obtener rápidamente resultados y donde los requisitos son muy cambiantes o poco definidos.

Un desarrollo Scrum cuenta con las siguientes características:

* **Desarrollo incremental.** En Scrum el desarrollo va aumentando con el tiempo, sin importar el orden en el cual se realicen los procesos.
* **Calidad del equipo.** La calidad del producto depende de la auto-organización y conocimiento de las personas que forman el equipo.
* **Prioridades.** Si un proceso necesita realizarse antes que un proceso anterior por necesidades del proyecto se realiza.
* **Comunicación.** Los miembros del equipo deben estar en comunicación constantemente durante el proyecto.

Los procesos de la metodología Scrum son los siguientes:

* **Product Backlog.** Se trata de una lista de funcionalidades que se deben desarrollar para el producto. Debe estar ordenado de mayor a menor prioridad.

Para realizar nuestro Product Backlog, hemos utilizado la herramienta online ***Trello****.*

* **Sprint Planning.** El Sprint Planning es una reunión que se realiza antes de llevar a cabo un Sprint. En esta reunión se definen los plazos y procesos que se deben efectuar para el proyecto establecido en el Product Backlog.

En este proyecto los Sprint duraban alrededor de un mes y realizábamos el Sprint Planning entre los tres componentes del equipo. En este Sprint Planning asignábamos las tareas que cada uno íbamos a realizar durante ese Sprint.

* **Daily.** Los daylis son reuniones diarias durante el sprint donde cada miembro del equipo cuenta qué hizo ayer, qué va a hacer hoy y qué problemas se está encontrando durante el desarrollo.

En este proyecto realizábamos daylis vía chat para que todos conociésemos que problemas teníamos y ayudarnos y, a la vez, tener en cuenta la evolución del proyecto.

* **Sprint Review**. Se trata de la reseña del Sprint. Una vez se finaliza el Sprint se realiza y aquí es donde se enseña algo tangible al cliente.

Durante nuestro trabajo, en las Sprint Reviews nos reuníamos y poníamos en común cada una de nuestras aplicaciones en ese momento para ver como avanzaba nuestro trabajo.

* **Sprint Retrospective.** Tras todos los pasos anteriores se realiza la Sprint Retrospective. Se trata de una reunión donde el equipo analiza los objetivos cumplidos, las dificultades que se han obtenido y se proponen mejoras para los siguientes Sprints.

En toda metodología Scrum deben aparecer los siguientes roles:

* **Product Owner.** Es principalmente el líder del proyecto. Será el encargado del proyecto, de hacer que se cumplan los objetivos y de la comunicación con el cliente.

En nuestro proyecto, la figura del Product Owner ha sido nuestro tutor.

* **Scrum Master.** Es la figura que ayuda a solucionar todos los problemas que se produzcan en el proyecto. Debe ser una persona con altos conocimientos técnicos. También marca las pautas de cada Sprint.

En este trabajo todos los integrantes del grupo hemos tomado durante algún tiempo este papel por mutuo acuerdo dependiendo de nuestra disponibilidad en cada momento.

* **Scrum Team.** Se trata del núcleo de la metodología Scrum. Se encarga de desarrollar mediante código el software y de cumplir objetivos propuestos por el Product Owner.

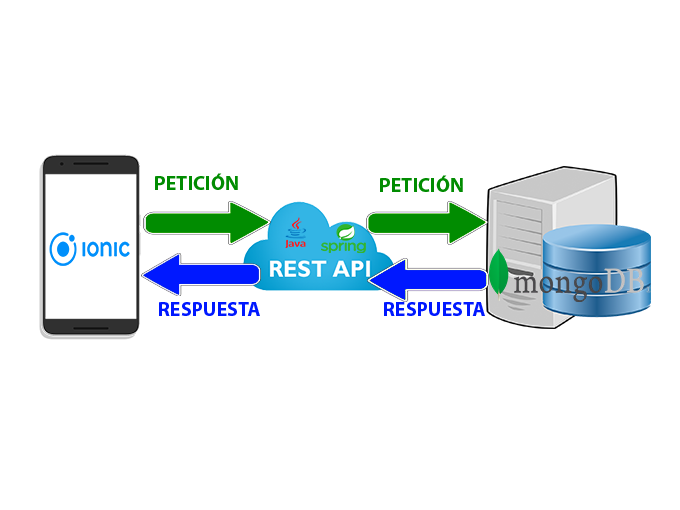
El Scrum Team de este proyecto hemos sido los tres alumnos que hemos realizado las tres diferentes aplicaciones en la gestión de ligas.

* **Cliente.** Aunque pueda parecer alguien externo, en la metodología Scrum el cliente debe formar parte del proceso. Se encarga de proponer nuevas ideas.

Durante este proyecto el papel de cliente ha sido ficticio ya que no existe un cliente real para nuestras aplicaciones, pero hemos realizado nuestro trabajo como si nuestro cliente fuera un árbitro, un comité de árbitro y un equipo con sus jugadores respectivamente para cada aplicación.

**Arquitectura del sistema**

Finalmente, la arquitectura del sistema queda de la siguiente forma: La aplicación móvil realizada con el uso de *Ionic Framework*, la base de datos en *MongoDB* y entre ambos una API REST desarrollado en *Java* con ayuda de *Spring Framework.*

****

Como he comentado anteriormente, la parte de la base de datos y la API REST se ha realizado de manera conjunta junto con mis compañeros Rubén y Jorge mientras que la aplicación móvil para el arbitraje la he realizado de manera individual.

**Herramientas utilizadas**

A continuación, presento las herramientas que he utilizado para realizar todas las partes del proyecto (base de datos, API REST y app).

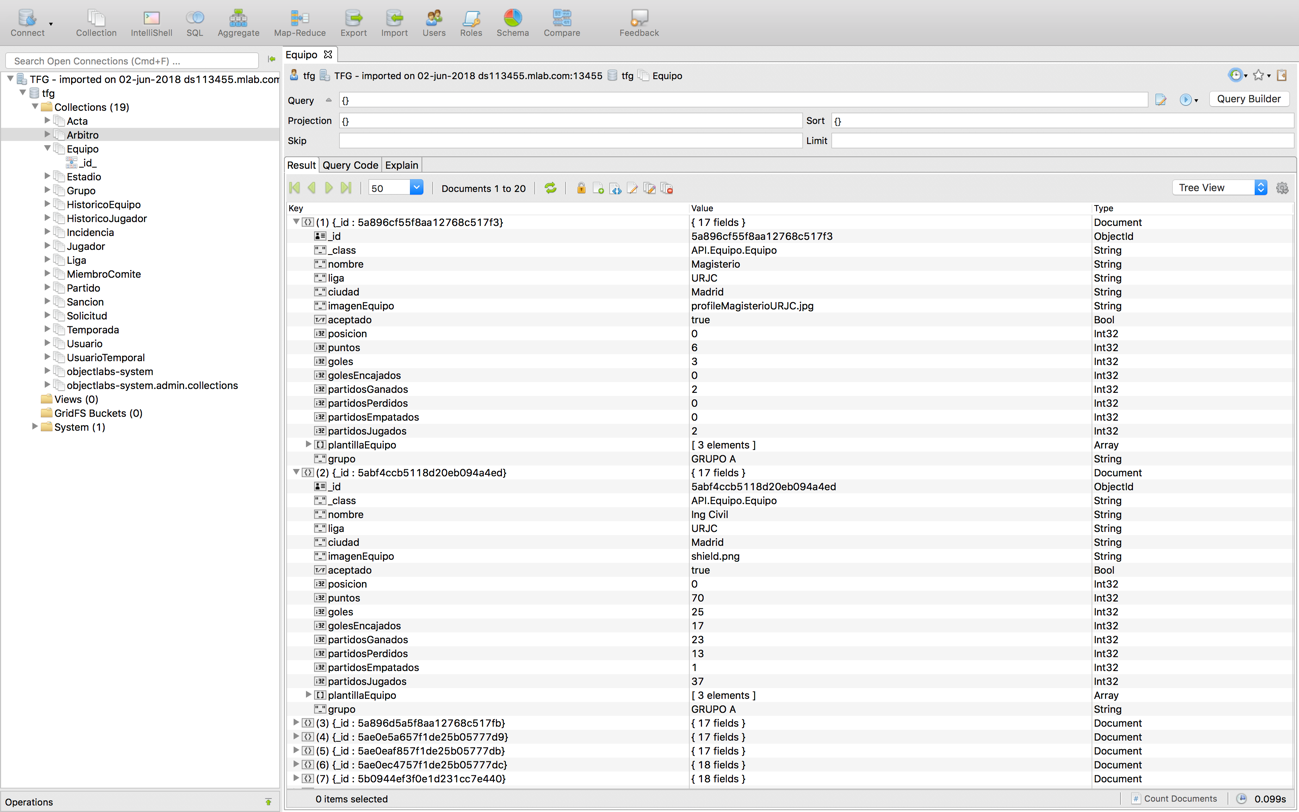
**Studio 3T**

Studio 3T (anteriormente conocida como Robomongo) es una interfaz gráfica para bases de datos en MongoDB.

Una interfaz gráfica no es una aplicación necesaria para la gestión de una base de datos, pero sí que la simplifica mucho.

Studio 3T ofrece muchas prestaciones para hacer que la gestión de bases de datos en MongoDB sea mucho más amena y sencilla. Tras mi uso, destaco principalmente las siguientes:

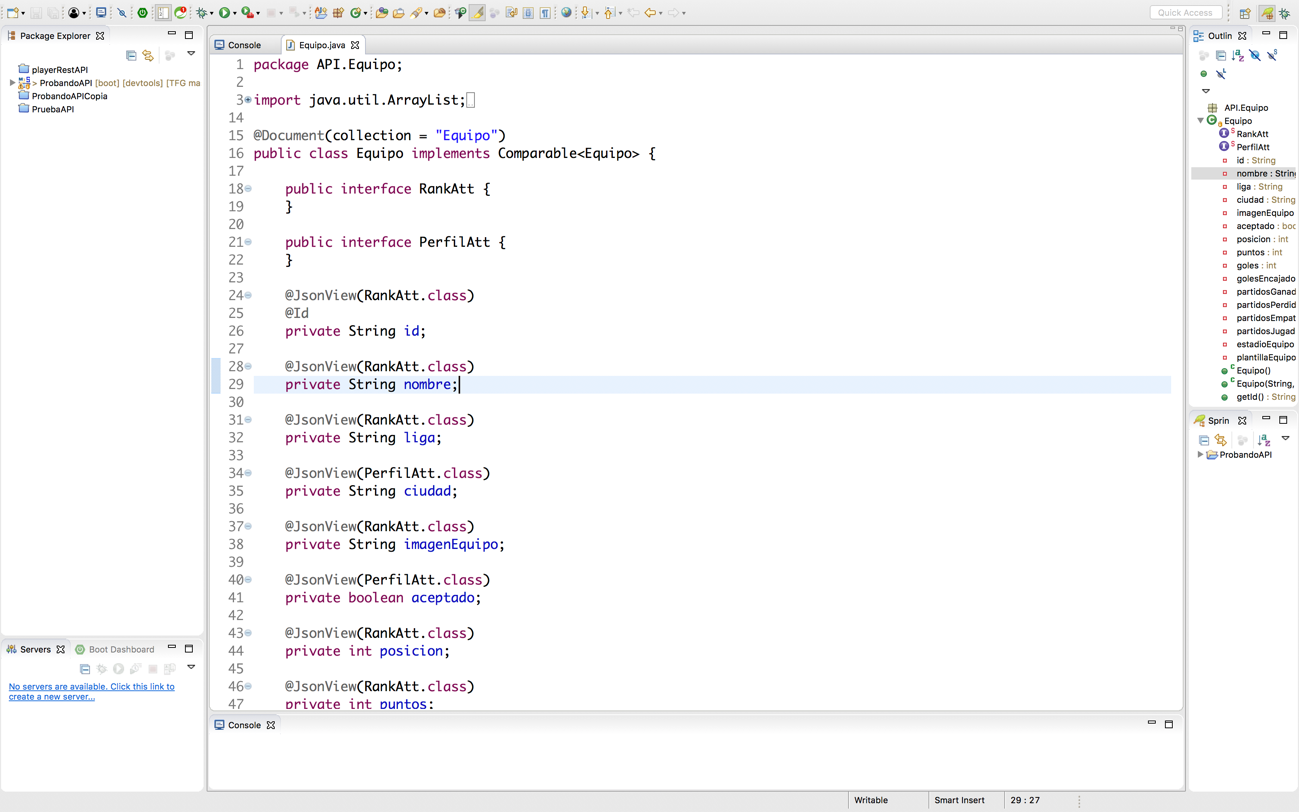
* **Query Builder**. Studio 3T proporciona una pequeña interfaz para realizar las queries, ya que realizar las queries en MongoDB no es tan sencillo como en SQL, por ejemplo.
* **In-Place Data Editing**. Studio 3T permite seleccionar un campo de cualquier entidad y cambiar el valor directamente.
* **Vista de tabla, árbol y JSON**. Mediante Studio 3T podemos visualizar los datos en distintas vistas que pueden ser en la forma típica de tabla, en forma de árbol y en formato JSON.
* **Intellishell**. Esta interfaz gráfica proporciona una consola directamente para poder realizar cualquier comando sobre la base de datos.



**Spring Tool Suite**

STS (Spring Tool Suite) es un entorno de desarrollo basado en Eclipse y que está personalizado para desarrollar apliaciones con Spring. Proporciona un entorno listo para usar para implementar, depurar, ejecutar y desplegar sus aplicaciones Spring e incluye integraciones para Git y Maven entre muchos otros.

Spring Tool Suite es compatible con la orientación de aplicaciones a servidores locales, virtuales y basados en la nube. Está disponible gratuitamente para el desarrollo y el uso de operaciones comerciales internas sin límites de tiempo, totalmente de código abierto y con licencia bajo los términos de la Licencia Pública de Eclipse.

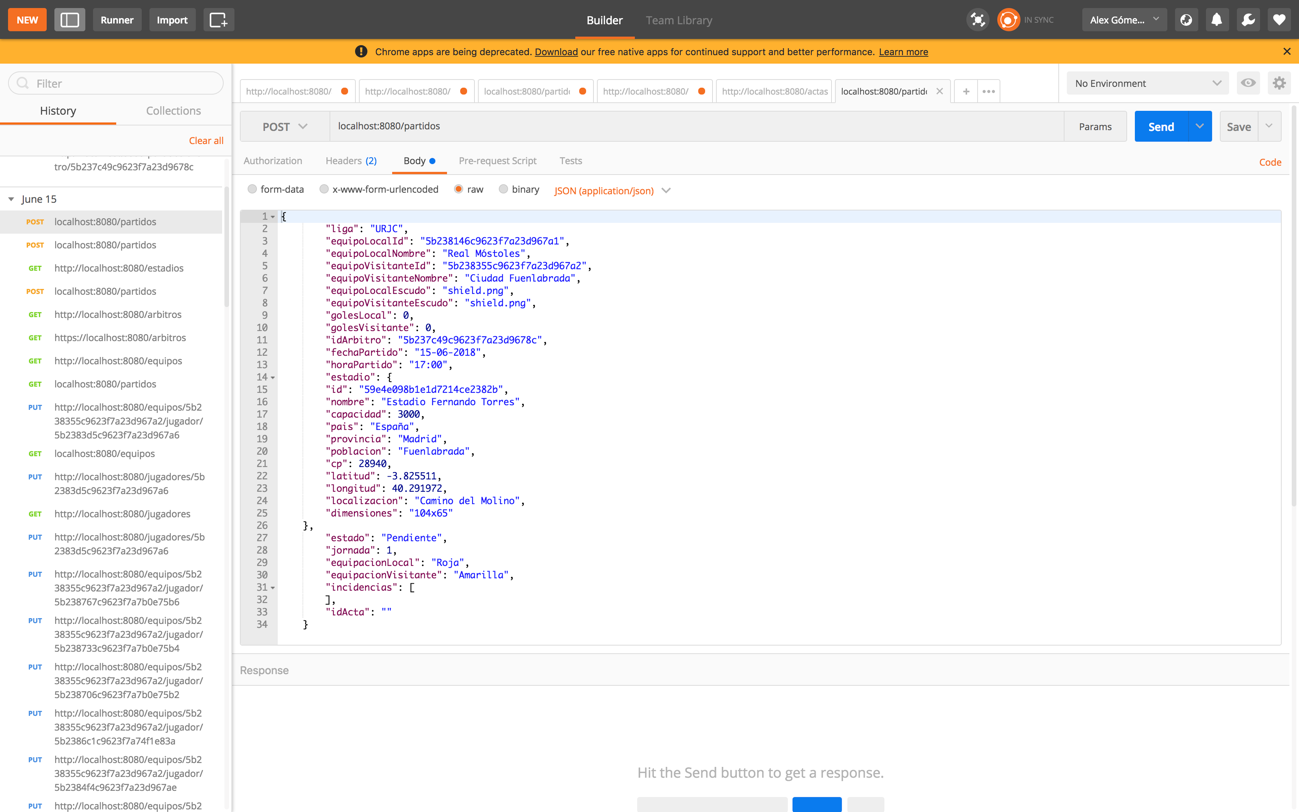
Elegí este entorno de desarrollo debido a que ya lo conocía debido a otros trabajos de la carrera y me parece muy intuitivo para desarrollar aplicaciones con Spring.

**Postman**

Se trata de una extensión de Google Chrome para realizar testing sobre API’s. Está diseñada pensando en desarrolladores y en facilitar realizar peticiones a estas.

En mi opinión, las características más interesantes que ofrece son:

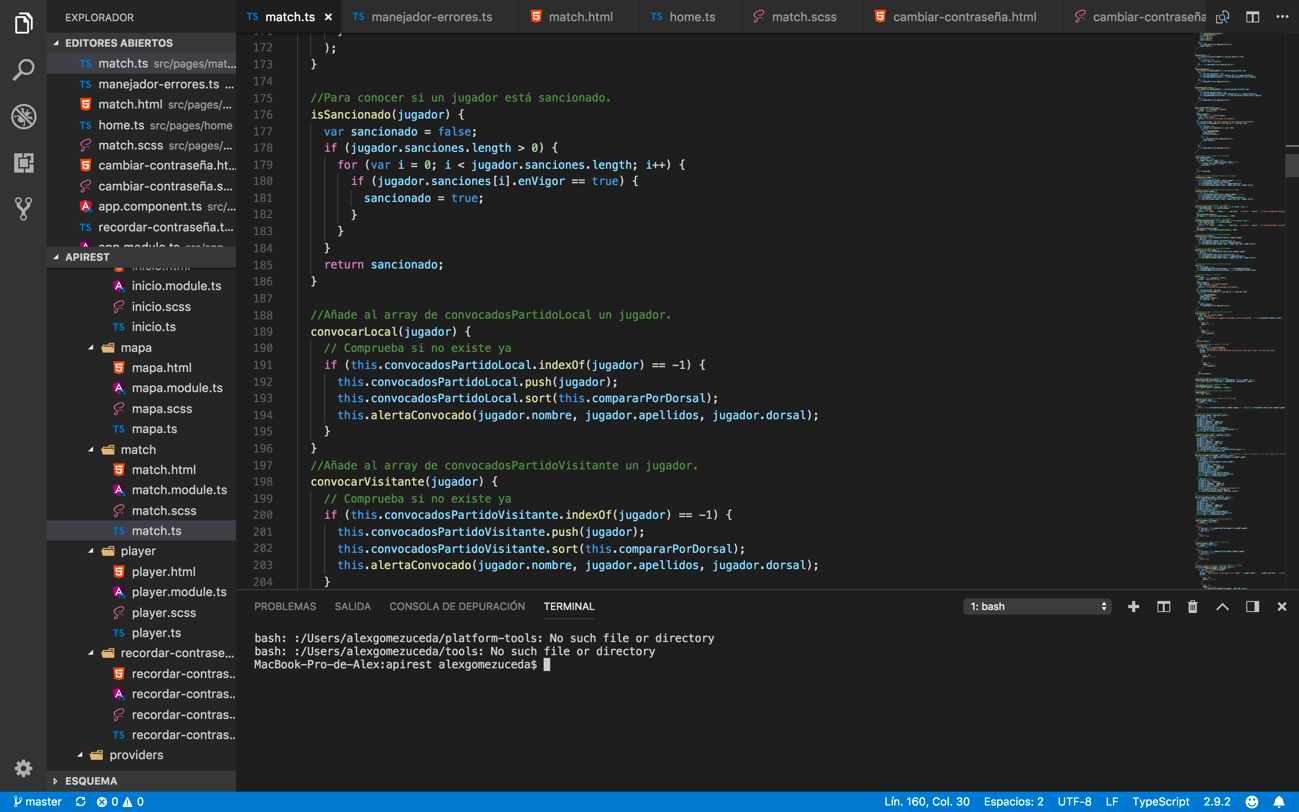
* Interfaz gráfica útil y sencilla.
* Creación ilimitada de colecciones de peticiones.
* Capacidad para guardar colecciones de peticiones.
* Historial diario de peticiones realizadas.
* Monitoreo de la API: tiempo de actividad, rendimiento y precisión.



**Visual Studio Code**

Visual Studio Code es un editor de código fuente desarrollado por Microsoft para Windows, Linux y macOS. Incluye soporte para la depuración, control integrado de Git, resaltado de sintaxis, finalización inteligente de código, fragmentos y refactorización de código.

También es personalizable, por lo que los usuarios pueden cambiar el tema del editor, los atajos de teclado y las preferencias. Es gratuito y de código abierto, ​aunque la descarga oficial está bajo software propietario.

Además, para mi caso, que se trataba de código escrito en Typescript proporciona herramientas de Debug. Es por este motivo añadido a su simplicidad, ayuda de escritura y su consola por lo que elegí utilizar este editor para crear mi aplicación en lugar de usar otros editores como Atom, que también he usado en otras ocasiones durante la carrera.

**Trello**

Trello se trata de un gestor de proyectos online que permite aclarar rutinas de trabajo, priorizar y generar avisos de citas entre otras tareas.

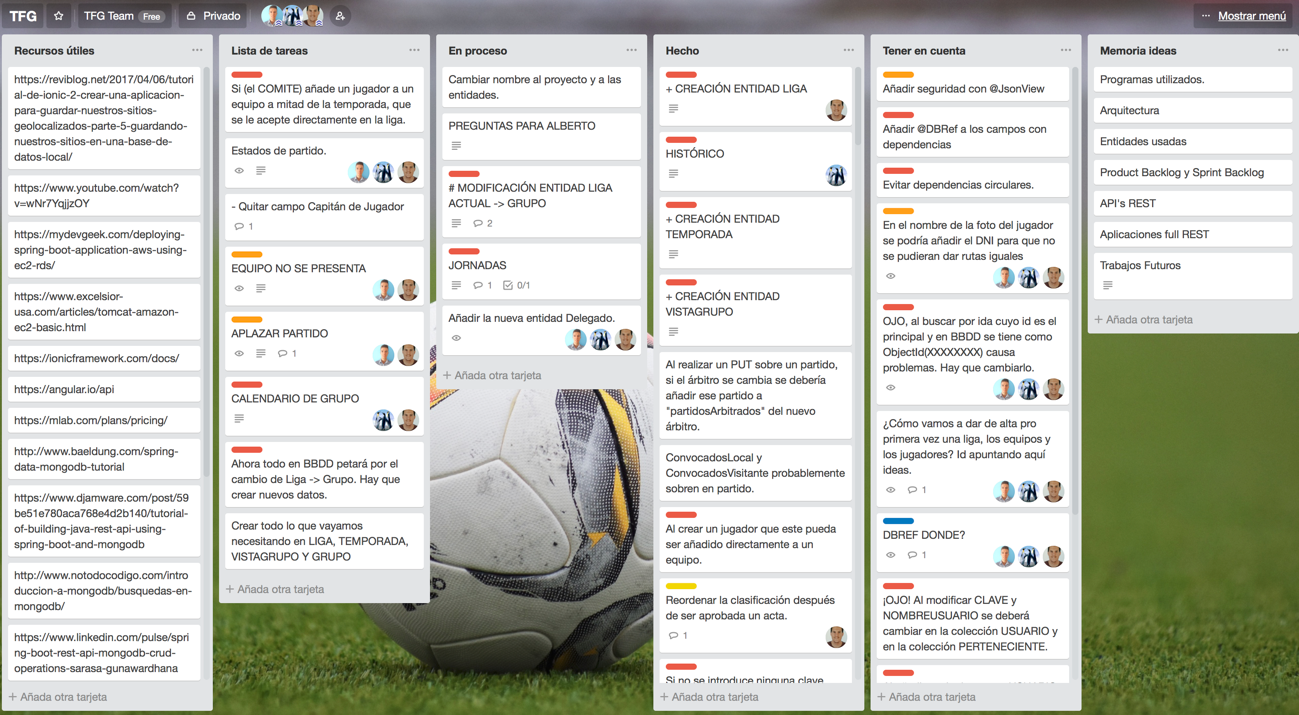
Trello se configura como un tablero, de manera sencilla e intuitiva. Este tablero se reduce a distintas columnas que se pueden crear y que, a su vez, se dividen en tarjetas. Estas tarjetas recogen las tareas o labores dentro de cada lista.

En estas tarjetas se pueden adjuntar archivos, agregar comentarios, fechas de finalización, añadir responsables e indicar nivel de prioridad entre otras características.

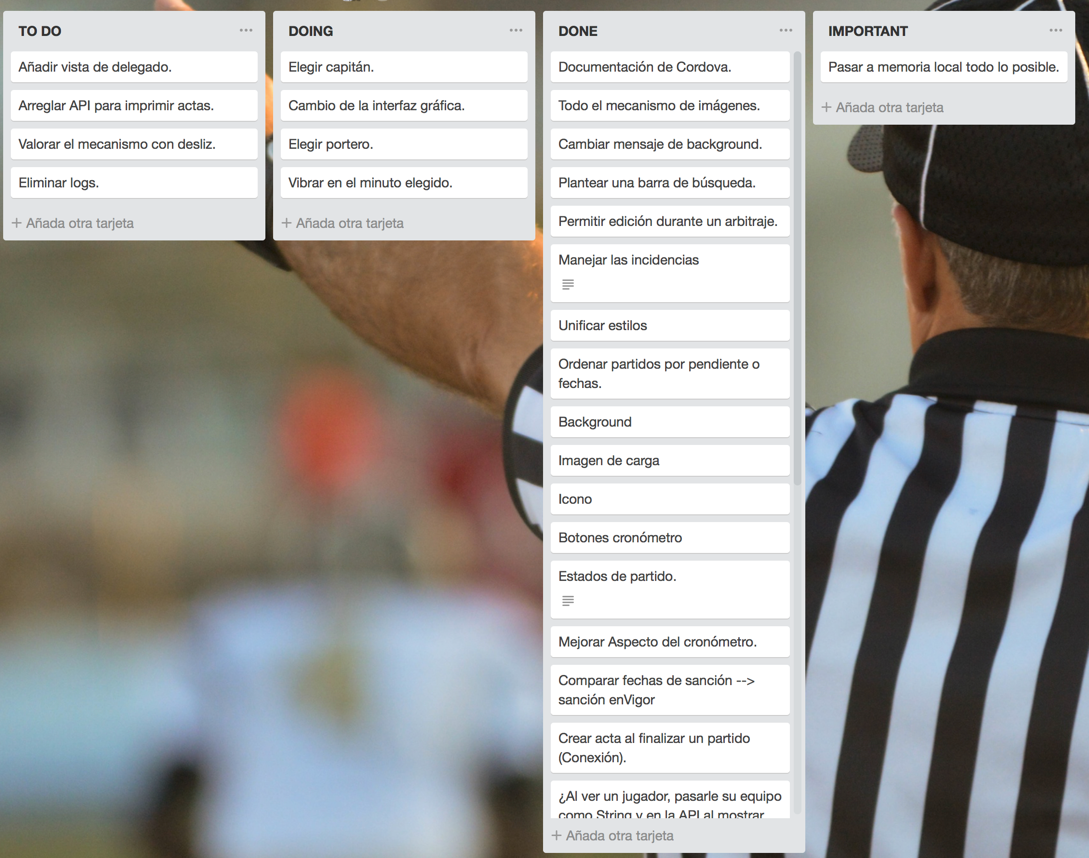
Además, Trello permite crear un tablero en conjunto con otros usuarios, lo que permite el trabajo en equipo de forma instantánea y sencilla.

En mi caso, he utilizado dos tableros. Uno de los tableros es compartido con mis compañeros Rubén y Jorge mientras que el otro lo he usado para organizar mis propias ideas sobre mi aplicación de arbitraje.

Esta es la forma que tienen ambos tableros:



Tablero grupal



Tablero individual

**Historias de usuario**

A continuación, muestro las historias de usuario de la aplicación con la siguiente estructura:

|  |  |  |
| --- | --- | --- |
| Historia de usuario | Prioridad | Estimación |
| Como <tipo de usuario> quiero <objetivo> (para <finalidad>) | Alta/Media/Baja | N días. |

Las historias de usuario de la aplicación son las siguientes

|  |  |  |
| --- | --- | --- |
| Historia de usuario | Prioridad | Duración |
| Como árbitro quiero realizar iniciar sesión en la aplicación para visualizar mi contenido. | Alta | 6 días |
| Como árbitro quiero recordar mi contraseña. | Alta | 2 días |
| Como árbitro quiero poder cambiar mi contraseña. | Alta | 2 días |
| Como árbitro quiero visualizar mis partidos pendientes para conocer estos en todo momento. | Alta | 2 días |
| Como árbitro quiero visualizar todos los datos de cada partido. | Alta | 3 días |
| Como árbitro quiero visualizar el estadio de cada partido mediante un mapa para poder encontrar mi ruta a este de forma fácil. | Media | 6 días |
| Como árbitro quiero visualizar todos los jugadores de la plantilla de cada equipo para poder confirmar su asistencia. | Alta | 3 días |
| Como árbitro quiero visualizar los datos de cada jugador y del delegado de manera individual para comprobar que todos los datos son correctos. | Alta | 2 días |
| Como árbitro quiero poder convocar y desconvocar jugadores para que estos aparezcan o no a la hora de arbitrar el partido. | Alta | 5 días |
| Como árbitro quiero seleccionar quien es el capitán de cada equipo en el partido. | Media | 1 día |
| Como árbitro quiero seleccionar quién es el portero de cada equipo en el partido. | Media | 1 día |
| Como árbitro quiero poder aplazar un partido. | Media | 2 días |
| Como árbitro quiero poder notificar que uno de los equipos no se ha presentado al partido. | Media | 2 días |
| Como árbitro quiero poder añadir tarjetas amarillas y rojas y goles a los jugadores. | Alta | 6 días |
| Como árbitro quiero poder controlar el tiempo mediante un cronómetro. | Alta | 2 días |
| Como árbitro quiero poder modificar los convocados de cada equipo durante su arbitraje. | Media | 4 días |
| Como árbitro quiero poder buscar a cada jugador por su dorsal para identificarlo de manera rápida. | Baja | 3 días |
| Como árbitro quiero generar un acta de manera automática al acabar cada partido | Alta | 5 días |

El total de duración es aproximadamente de **57 días**, suponiendo un presupuesto de **80€/día** tenemos un presupuesto final de **4560€.**

**Bibliografía**

* http://www.rfef.es
* <https://www.solbyte.com/blog/2014/07/21/tipos-de-aplicaciones-moviles-nativas-webs-hibridas/>
* <https://okhosting.com/blog/ventajas-desventajas-de-las-aplicaciones-moviles-nativas/#Ventajas_de_crear_aplicaciones_moviles_nativas>
* <https://www.raona.com/aplicacion-nativa-web-hibrida/>
* <https://es.wikipedia.org/wiki/Front-end_y_back-end>
* <https://serprogramador.es/que-es-frontend-y-backend-en-la-programacion-web/>
* <https://es.wikipedia.org/wiki/Framework>
* <https://jordisan.net/blog/2006/que-es-un-framework/>
* <http://www.phonegapspain.com/que-es-y-como-empezar-con-ionic-framework/>
* <https://es.wikipedia.org/wiki/Metodolog%C3%ADa_de_desarrollo_de_software#Otros_enfoques_de_desarrollo_de_software>
* <https://okhosting.com/blog/metodologias-del-desarrollo-de-software/#Cuales_son_las_Principales_Metodologias_Agiles>
* <https://okhosting.com/blog/metodologias-del-desarrollo-de-software/#Cuales_son_las_Principales_Metodologias_Agiles>
* <https://es.wikipedia.org/wiki/Kanban_(desarrollo)>
* <https://cordova.apache.org/docs/es/latest/guide/overview/>
* <https://es.wikipedia.org/wiki/Apache_Cordova>
* <https://spring.io>
* <https://studio3t.com>